SI4432 Packet Transmission

4 August, 2020
0:02

Doel: testen hoe ik zonder het pakket precies ken, de data van het weerstation uit te lezen.
Basis opzet:

- UHF Generator wordt uitgebreid met het zenden van een data package iedere seconde
- De initialisatie van de UHF generator ook gebruiken als de initialisatie (bijna) voor ontvangst, zodat frekwenties en bandbreedtes exact kloppen
- Generator testen op spectrum analyzer en statusregisters na zenden
- Receiver .... te testen, eerst wetende wat er komt, daarna stel dat ik niets weet ...

UHF Generator

Carrier 434 MHz, zodat een goede match voor de 470M modules wordt verkregen.
Packet:
lange preamble : 64 nibbles = 32 bytes

```
1814   SI4432_Write ( 0x34, 64 )       -- Preamble Length
```

4 sync bytes:

```
1807   SI4432_Write ( 0x33, 0x06 )     -- 0000_0110   Header Control 2
1808                                    -- 7    = 0    = skipsyn
1809                                    -- 654  = 000  = hdlen
1810                                    -- 3    = 0    = fxpklen
1811                                    -- 21   = 11   = synclen
1812                                    -- 0    = 0    = MSB of preamble length
```

```
1820   ;SI4432_Write ( 0x36, 0x2D )     -- Sync Word 3
1821   ;SI4432_Write ( 0x37, 0xD4 )     -- Sync Word 2
1822   ;SI4432_Write ( 0x38, 0x00 )     -- Sync Word 1
1823   ;SI4432_Write ( 0x39, 0x00 )     -- Sync Word 0
```

no header bytes
no datalength
10 bytes of data

De belangrijkste settings

```
SI4432_Write ( 0x30, 0xA8 )   -- Data Access Control (Packet handling)
                              -- 7  = 1  = Enable Rx Packet handling
                              -- 6  = 0  = LSB first
                              -- 5  = 1  = CRC Data Only
                              -- 4  = 0  = skip2ph
                              -- 3  = 1  = Enable Tx Packet handling
                              -- 2  = 0  = CRC Enable
                              -- 10 = 00 = CRC Polynome

SI4432_Write ( 0x32, 0x00 )   -- Header Control 1
                              -- 7654 = Broadcast address
                              -- 3210 = Received header check

SI4432_Write ( 0x33, 0x06 )   -- 0000_0110   Header Control 2
                              -- 7   = 0   = skipsyn
                              -- 654 = 000 = hdlen
                              -- 3   = 0   = fxpklen
                              -- 21  = 11  = synclen
                              -- 0   = 0   = MSB of preamble length

SI4432_Write ( 0x34, 64 )     -- Preamble Length
```

De zendroutine die iedere seconde wordt aangeroepen

```
procedure SI4432_Generate_FSK_Block_2 () is
  const byte Data_Len = 10
```

```
    var   byte i

    -- reset FIFO
    SI4432_Write ( 0x08, 0x03 )
    SI4432_Write ( 0x08, 0x00 )

    -- packet length
    SI4432_Write ( 0x3E, Data_Len )

    -- fill the FIFO
    for Data_Len using i loop
      SI4432_Write ( 0x7F, i+16 )
    end loop

    -- Start Transmission
    SI4432_Write ( 0x07, 0x0B )

    -- TEST: read EZMAC
    serial_hw_write ( SI4432_Read ( 0x31 ) )

    -- wait for transmission is ready, VERY IMPORTANT
    while SI4432_Read ( 0x07 ) != 0x03 loop
      serial_hw_write ( SI4432_Read ( 0x31 ) )
      delay_1ms (20)
    end loop

    -- when the FIFO gets empty, while loop above,
    -- the last byte has to be sent yet
    -- so depending on the Baudrate we've to wait a little while
    delay_1ms ( 2 )  -- needs to be at least 2 msec
    serial_hw_write ( SI4432_Read ( 0x02 ) )
    serial_hw_write ( SI4432_Read ( 0x03 ) )
    serial_hw_write ( SI4432_Read ( 0x04 ) )
    serial_hw_write ( SI4432_Read ( 0x31 ) )
    -- response should be:  20 24 02 01
    --                      | | | |__ package sent
    --                      | | |_____ Chip Ready
    --                      | |_______ ipksent, TX-FIFO almost empty
    --                      |_________ RX/TX Underflow
  end procedure
```

Als we zenden, zien we netjes

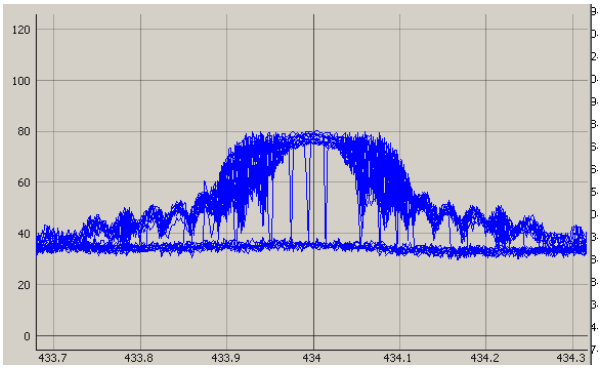
```
01 01 02 02 02 02 02 02 02
20 24 02 01
```

de eerste regel geeft

01 = package sent (van de vorige), daarna

02 = packet transmitting

de spectrum analyzer laat ook een mooi plaatje zien



Fix packet length is disabled, so the packet will contain the packet len.
Als ik het FIFO met 5 (ipv 10) bytes laad, krijg ik als response

```
01 01 02 02 02 02 02 02 02
60 A4 02 01
```

60 = Rx/Tx FIFO Underflow + Rx-FIFO empty
A4 = FIFO under/overflow error + Tx FIFO almost empty + Packet sent interrupt
klinkt redelijk logisch

Als ik het FIFO met 15 (ipv 10) bytes laad, krijg ik als response

```
01 01 02 02 02 02 02 02 02
20 24 02 01
```

niets aan de hand, want ik clear het FIFO tochbij iedere zendactie
Als ik overga op fixed packet len, krijg als respons en op de sspectrum analyzer precies hetzelfde.
fxpkln bepaald dus alleen maar of de lengte in het packet wordt opgenomen.
packet length en de hoeveelheid data in de FIFO bepalen of er een Underflow optreedt.
Dus eigenlijk heel logisch.

Veranderingen in 0x02, 0x03,0x04,0x07

20 20 02 07
20 20 12 07 RSSI
20 20 52 07 valid-preamble + RSSI
20 20 D2 07 valid-preamble + sync-detected + RSSI
20 20 92 07 sync-detected + RSSI
20 20 82 07 sync-detected
20 20 02 03
22 22 02 03 valid packet
Algemene Mode is van 07 (Receiving) gegaan naar 03 (standby).
Reg 03 springt naar Valid packet received.
maar meestal ziet het er zo uit
20 20 22 07
20 20 02 07
20 20 22 07
.... repeat of above
20 20 02 07
20 20 42 07 valid preamble
20 20 C2 07 valid preamble + sync-detected
20 20 82 07 sync detected
22 22 02 03 valid packet
Dus zou ik hem nu moeten kunnen uitlezen
....
20 20 82 07 sync detected
20 20 02 03
00 01 02 03 04 05 06 07 08 09 Data is ok
22 22 02 03 valid packet
01 01 01 01 01 01 01 01 01 01 try to read more data, but that's rubish
A2 A2 02 03 now underflow occures
01 01 01 01 01 01 01 01 01 01 and even more rubish data
Why "Valid packet" we only received 10 bytes and packlen =12 ???
let's try packet len 6
still works,
OK: fxpklen=0 en dan doet reg 0x3E (packlen) niets in receive mode
Ik weet op dit moment niet precies welke code er in de UHF-Generator zit, want data had 10,11,12,... moeten zijn, dus eerst UHF-Generator opnieuw laden.
Nu weet ik vrij zeker dat ik de volgende instellingen heb

- TX-Packet handling
- 64 nibbles preamb
- synclen = 4: 2D, D4, 00, 00
- no header
- fixpacklen reg(3E) = 10
- data 10, 11, ... 1A

Als ik nu lees met de volgende settings

- RX packet handling
- fxpklen = 1
- preamble = 4
- synclen = 4
- packet len = 12

Dan krijg ik het volgende
20 20 C2 07
20 20 82 07
20 20 82 03
10 11 12 13 14 15 16 17 18 19 << correcte data
22 22 02 03
3F FF F3 F3 F3 F3 F3 F3 F3 F3
A2 A2 02 03
eigenlijk verbazingwekkend dat dit goed gaat, terwijl er geen package len bekend is.

Ok, wat als we fxpklen=1 maken en correcte packlen opgeven
20 20 82 07
20 20 02 03
10 11 12 13 14 15 16 17 18 19 3F FF
22 22 02 03 <<< hier pas een valid packet, dus hier pas FIFO uitlezen
F3 F3 F3 F3 F3 F3 F3 F3 F3 F3 F3 F3
A2 A2 02 03
Ok 12 uitlezen gaat goed zonder overflow.
En 13 uitlezen geeft een underflow as was to be expected.
20 20 82 07
20 22 02 03
10 11 12 13 14 15 16 17 18 19 3F FF F3
A2 A2 02 03

Change the program so it wait until valid package

we get different results
20 20 82 07
20 20 02 03
22 22 02 03
10 11 12 13 14 15 16 17 18 19
20 20 82 07
20 22 02 03
10 11 12 13 14 15 16 17 18 19
22 22 02 03
01 01 01 01 01 01 01 01 01 01
A2 A2 02 03
01 01 01 01 01 01 01 01 01 01
We never see 0x02:5 is RX-FIFO Empty go high ???
We see 0x02:2 is Reserved (RFM22 is Synthesizer Lock) go high ???
So this leads to the following protocol:
no, program error R02 was looking at reg 0x03
01 20 82 07
00 22 02 03
10 11 12 13 14 15 16 17 18 19
20 22 02 03
01 01 01 01 01 01 01 01 01 01
60 A2 02 03
01 01 01 01 01 01 01 01 01 01
YES 0x02:5 now is set to high when the Rx-register is empty

Wait for "Valid Package Received" becomes True

Reading status bits is tricky, because a bit can stay a long time high (not always clear when they are reset).]
So the next code waits until the Valid Package Received becomes set.

```
R03_Old = 0x02
forever loop
  R03 = SI4432_Read ( 0x03 )
  if ( R03 & 0x02 ) & ( ! ( R03_Old & 0x02 ) ) then
    for 10 loop
      serial_hw_write ( SI4432_Read ( 0x7F ) )
    end loop

    -- back to Receive mode
    SI4432_Write ( 0x07, 0x07 )    -- Operating Mode and Function Control 1
  end if
  R03_Old = R03
end loop
```

And we see, most of the time receiving is correctly.
We have to find out if we can detect the errors.
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
08 08 89 09 8A 0A 8B 0B 8C 0C
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19

10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
08 08 89 09 8A 0A 8B 0B 8C 0C
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19
08 08 89 09 8A 0A 8B 0B 8C 0C
10 11 12 13 14 15 16 17 18 19
10 11 12 13 14 15 16 17 18 19

Reading FIFO until empty

works, because the number of bytes in the FIFO will be exactly the same as the specified package len in reg 0x3E, the for loop is more elegant.

```
R03_Old = 0x02
forever loop
  R03 = SI4432_Read ( 0x03 )
  if ( R03 & 0x02 ) & ( ! ( R03_Old & 0x02 ) ) then

    -- read until RX-FIFO empty
    while  ( SI4432_Read ( 0x02 ) & 0x20 ) == 0x00 loop
      serial_hw_write ( SI4432_Read ( 0x7F ) )
    end loop

    -- back to Receive mode
    SI4432_Write ( 0x07, 0x07 )    -- Operating Mode and Function Control 1
  end if
  R03_Old = R03
end loop
```

setting package len to 16, will receive a package of 16 bytes
10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00
10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00
10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00
10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00
08 08 89 09 8A 0A 8B 0B 8C 0C 80 00 00 00 00 00
10 11 12 13 14 15 16 17 18 19 00 00 00 00 00 00
setting package len to 6, will receive a package of 6 bytes
10 11 12 13 14 15
10 11 12 13 14 15
10 11 12 13 14 15
10 11 12 13 14 15
20 22 24 4C 50 54
10 11 12 13 14 15
changing to receive 1 sync byte, will show the 3 last sync bytes as first data "D4 00 00"
D4 00 00 08 08 89 09 8A 0D 45
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
btw the error that occures in the first line, is a shift of 1 bit !!

Preamble Length

Depending on the settings, the worst case advised preamble detection is 8 nibbles.
3 nibbles: 30 errors
4 nibbles: 7 errors

5 nibbles: 0 errors
8 nibbles: 0 errors

here the results of 4 nibbles preamble
D4 00 00 08 08 89 09 8A 0A 8B <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 08 08 89 09 8A 0A 8B <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
AF FF EB 7B 5F 5C BB 79 7A B7 <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 08 08 89 09 8A 0A 8B <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 08 08 89 0C C5 05 45 <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 08 08 89 09 8A 0A 8B <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 08 08 89 09 8A 0A 8B <
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16
D4 00 00 10 11 12 13 14 15 16

RSSI Treshold + RSSI offset

No differences found with the following settings:

```
33   SI4432_Write ( 0x27, 0xA6 )     -- RSSI treshold
34   SI4432_Write ( 0x27, 0x11 )     -- RSSI treshold
35   SI4432_Write ( 0x27, 0x00 )     -- RSSI treshold
```

```
27  SI4432_Write ( 0x35, ( Preamble_Len << 3 ) | 7 ) -- Preamble Detection Control
28  SI4432_Write ( 0x35, ( Preamble_Len << 3 ) ) -- Preamble Detection Control
29                               -- 76543 = 4 = preamble detection treshold
30                               -- 210   = 2 = RSSI offset = 2*4dB
```

Delay before setting Receive mode

?????? otherwise nothing is received !!!

```
41  -- start Receiving
42  delay_10us ( 2 )               -- <<<<<<<<< Necessary !!!!
43  SI4432_Write ( 0x07, 0x07 )    -- Operating Mode and Function Control 1
```