Nordic VLSI ASA

nRF™ Radio protocol guidelines **nAN400-07**

1. GENERAL

This application note describes two different methods of how to implement software routines to transfer data from one microcontroller to another via a nRF™ chip:

- Oversampling
- Use of a UART (Universal Asynchronous Receiver Transmitter.)

This document only describes the link-layer part of the OSI reference model. It is however enough information in this document to be able to build a radio link using the nRF devices.

2. PACKET FORMAT

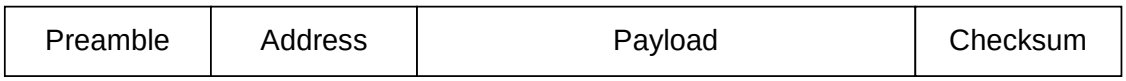


Figure 1 Reference packet format

2.1. Preamble

A packet has to begin with a preamble. The preamble has two tasks:
1: Stabilise the receiver.
2: Synchronise a receiving UART if used.
How long a preamble should be, and how it should look like will vary from nRF device to nRF device and if you are using a UART or not.
Please refer to Table 1 for recommended preambles.

Device	With UART	Without UART
nRF40x	55FFh	AAh
nRF903	CCCCCCCCF0h	CCCCCCCCh

Table 1: Recommended preambles

2.2. Address

The address in the packet is used by the receiver to identify a packet. It can be a system address or a device address. In the case of a system address, the device address will be part of the payload. The length of the address will depend on how many devices the system contains of and the wanted likelihood of misinterpretation of a packet. Normally one - two bytes are enough as address field.

2.3. Payload

The payload bytes contain the data that is intended for the next layer in the protocol stack. How many bytes the payload consists of will vary from application to application, but the golden rule is to keep the packet as short as possible because this will give the packet greater chance to survive through the link.

2.4. Checksum

The checksum is used to validate the packet. This is calculated from the address and payload bytes. Never use the preamble bytes when calculating checksum. Normally one byte with checksum is enough. A typical checksum routine is the cyclic redundant check (CRC) routine. But also a much simpler XOR routine can be used.

For all nRF devices except the nRF240x in ShockBurst™ mode, the control unit has to perform preamble generation, address detection and checksum generation/validation. For the nRF240x family in ShockBurst™ mode, this is done automatically.

3. OVERSAMPLING

The oversampling method is a sampling method implemented in software and uses only the microcontrollers parallel port pins as interface to the nRF™ device. The method requires much CPU time and the data rate it is capable of handling is dependent on the CPU clock speed. A microcontroller that runs on 4MHz clock and is capable of doing most instructions on one cycle will not be able to handle data rates higher than 15kbit/sec.
The oversampling method is only recommended on the nRF40x family. For the nRF903 the use of UART is the preferred method.

Oversampling will result in a reliable link. Oversampling with a rate of 3 times the bit-rate with weighting of the samples, will be noise resistant. However timing is critical, as the period between each sampling must be consistent.

If you are using a microcontroller with an internal timer with overflow interrupt, you should use this to handle the timing. To implement a send routine is easy. The only pitfall is to forget about timing, and create bit edge jitter on the outgoing bit stream.

In order to prevent this, the interrupt routine must be able to run as soon as possible after the interrupt occurs. This means that interrupt disabling should be used only where it is strictly necessary.

The receive routine is more complex. It has to be able to determine if the received bit is a "one" or a "zero". When using oversampling, the receive routine should sample the received bit stream at least three times the bit-rate. Then the samples must be weighted according to a weighting table to determine the result. (An example of such a weighting table is shown in Table 2 below.) In addition to the three samples of the present bit, the last sample of the previous bit should be used. This will make the routine more resistant against edge jitter.

Sample 3	Sample 2	Sample 1	Sample 0	Result
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Table 2: Weighting table using three samples per bit

"Sample 0" is the last sample. "Sample 3" is the last sample in the previous bit.
"Result" is the resulting value of the bit.

4. UART

A Universal Asynchronous Receiver Transmitter (UART) can be found as a peripheral on many microcontrollers. The UART will convert between serial and parallel data. So if a byte is written to the UART inside the microcontroller, it will come out as serial data on the UART TXD pin. To be able to read serial data correctly, the UART must have a start and a stop bit in addition to the actual byte. The start bit is always a "0", and the stop bit is always a "1." When the UART detects a negative transition, it will start sampling the next eight bits at the programmed data rate. The stop bit is always "1" so a negative transition is made when the next start bit occurs. Normally a UART packet will contain ten bits/byte, but there is a possibility to use eleven bits/byte, where the added bit can be used for parity check i.e.

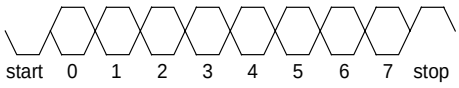


Figure 2 UART packet

When there are no transmitters present, a receiving nRF device will demodulate noise. This noise will be handled by the receiving UART. The receiving UART might be in the middle of sampling a byte when the first byte in the preamble arrives. Detecting the start bit is crucial to read the rest of the byte correctly. Therefore the preamble must be able to synchronise the UART so that when the first address byte arrives, the UART will detect the start bit correctly. A preamble with four bytes of CCh followed by a byte of hF0 will ensure that the receiving UART will be in sync and that the nRF903 receiver is dc levelled.

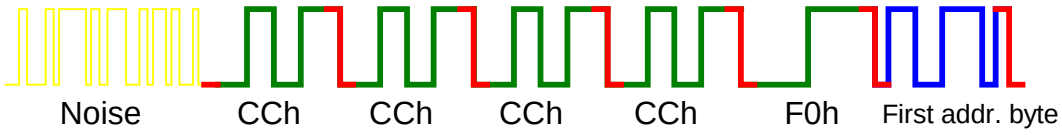


Figure 3 Preamble demodulated by a nRF903

Data passed trough the UART will be reversed. This means that on air, a preamble for the nRF903 will look like the one in Figure 3. Please note that noise is marked yellow, all start and stop bits are red, preamble bytes are green and first address byte is blue.

When using a UART together with the nRF903, the connection between the two data lines from the UART and the nRF903s bi-directional Data line can be done as shown in Figure 4.

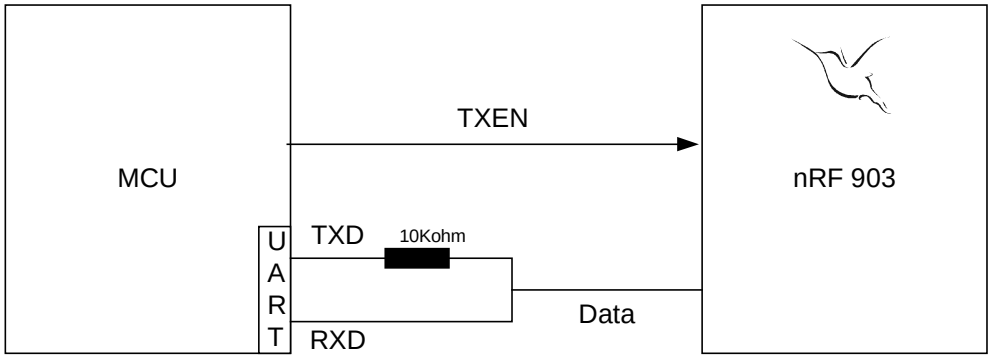


Figure 4 Connecting the UART to the nRF903

When using nRF transceivers with two separate data lines, the UART TXD shall be connected to the nRF DIN pin, and the UART RXD shall be connected to the nRF DOUT pin. **It is very important that the supply voltage for the microcontroller and the nRF device are on the same level.**

5. Scrambling

A data packet for RF communication should have as many transitions as possible to ensure a high performance RF link. The challenge is to manage this without adding too much overhead to your data packet like Manchester Coding does.

Scrambling of data is one way that does not add any overhead to your data packet at all. Scrambling is simply a logical XOR between the data byte and a code byte.
If we choose 10101010 as our code byte and 11110000 as the data byte and perform scrambling we get 10101010 XOR 11110000 = 01011010 as result. This is an increase from one to six transitions! To de-scramble data at the receiver side, we just take the received byte; 01011010 and perform XOR with the code; 10101010 and get the original data byte; 11110000 as the result. (01011010 XOR 10101010 = 11110000)

5.1. Synchronization

A UART packet contains of one start bit (always zero), eight data bits and one stop bit (always one.)

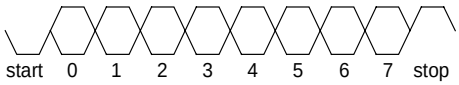


Figure 5 UART packet

The receiving UART will use the negative transition of the start bit as synchronisation when reading one byte of data. This is good since de-scrambling of received data requires synchronisation.

5.2. Dynamic scrambling

A data packet may contain bytes that are not suited for scrambling. In example the byte 10101010 will when scrambled with 10101010 give 00000000 as result. Clearly this is a situation where use of scrambling will *remove* transitions from the packet, and scrambling should not be used. For link protocols that shall be able to handle any byte value, and still get the maximum possible transitions, we need a way of switching the scrambling on and off for each byte in the data packet. This is called dynamic scrambling.

To be able to do this, we need one bit per UART packet that tells the receiving microcontroller if the byte is scrambled or not. Luckily, it is possible to add an extra bit to the UART packet. This bit was used as a parity bit option in RS232C.

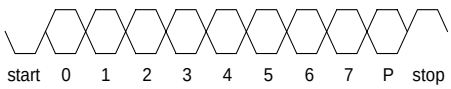


Figure 6 UART packet with extra bit (P)

We also need rules for when to use scrambling or not.
By using 10101010 as the code byte, we will be able to get maximum 7 transitions in one scrambled byte. If a data byte has n number of transitions before scrambling, the result will have 7-n transitions. So if a data byte has less than 4 transitions, it will be wise to scramble the byte.

5.3. Implementation

How to count number of transitions:

- Let the data byte be X.
- Make a copy of X, Y.
- Left shift Y one bit.
- Let Z = X XOR Y
- Count number of ”1” in Z. It tells you how many transitions X has.
- If Z has less then 4 ”1”s, scramble X before transmission.
- If scrambling, remember to set bit 9 in the UART packet.

6. Noise considerations.

Range and quality of service is dependent on the way the incoming data stream is interpreted. An oversampling algorithm already has a built in filter for noise or multipath effects. Any additional error detection and correction will provide an even better margin towards noise or distortion of the data stream. To be able to perform forward error correction a packet has to contain a lot of overhead. It is not recommended that such functionality is implemented unless it is strictly needed.

7. Conclusion

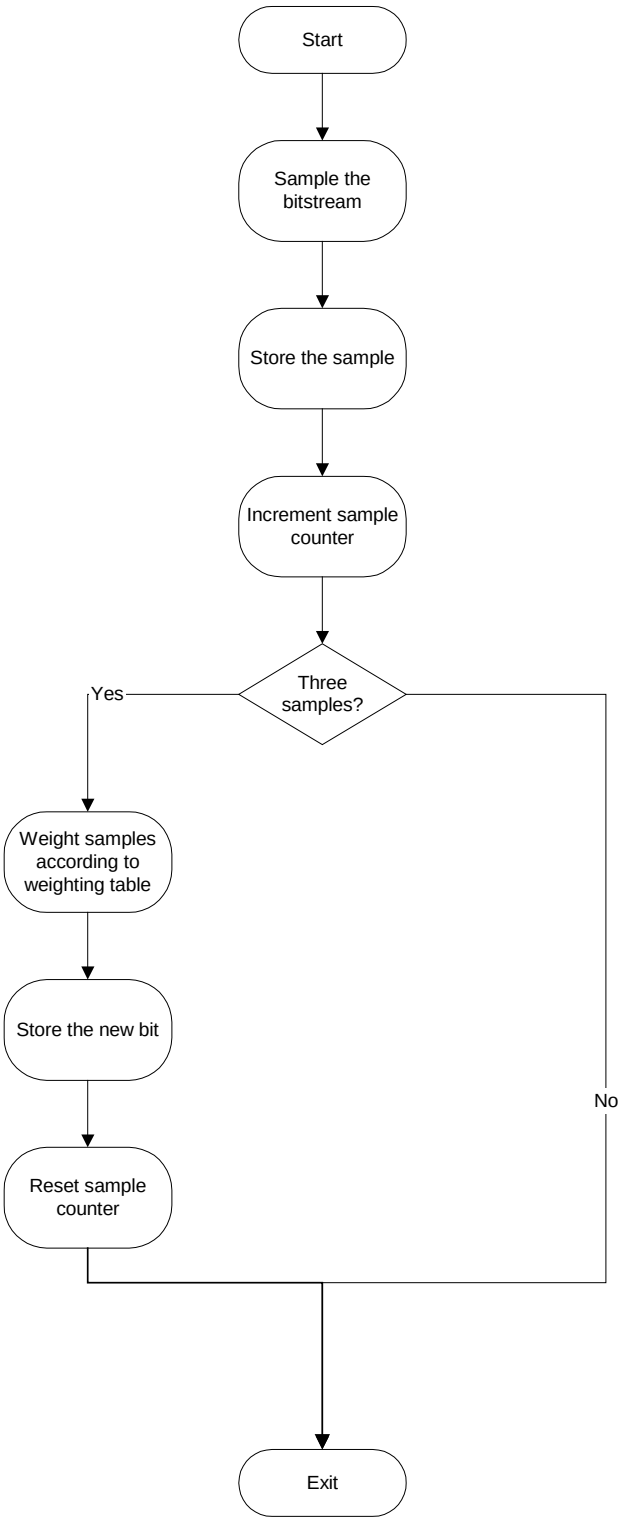
The nRF903 requires the UART method. It will run at 76.8kbit/sec and no low cost microcontroller can handle that data rate with the oversampling method. A high-speed microcontroller will be able to perform oversampling on 76,8kbit/sec, but the extra cost for this high-speed microcontroller will exceed the extra cost for a microcontroller with a UART. The total solution for the nRF903 design will be less expensive with the UART method.

For the nRF40x family, the use of the UART method will ease implementation of the protocol and minimise the workload for the microcontroller. If a low-cost application with low data rate is the design goal, a low-cost microcontroller can be loaded with the oversampling method and perform well.

The nRF401 demo kit is based on the use of a low-cost microcontroller with the oversampling method.
The nRF903 demo kit is based on the UART method, running at 76.8kbit/sec with use of scrambling.

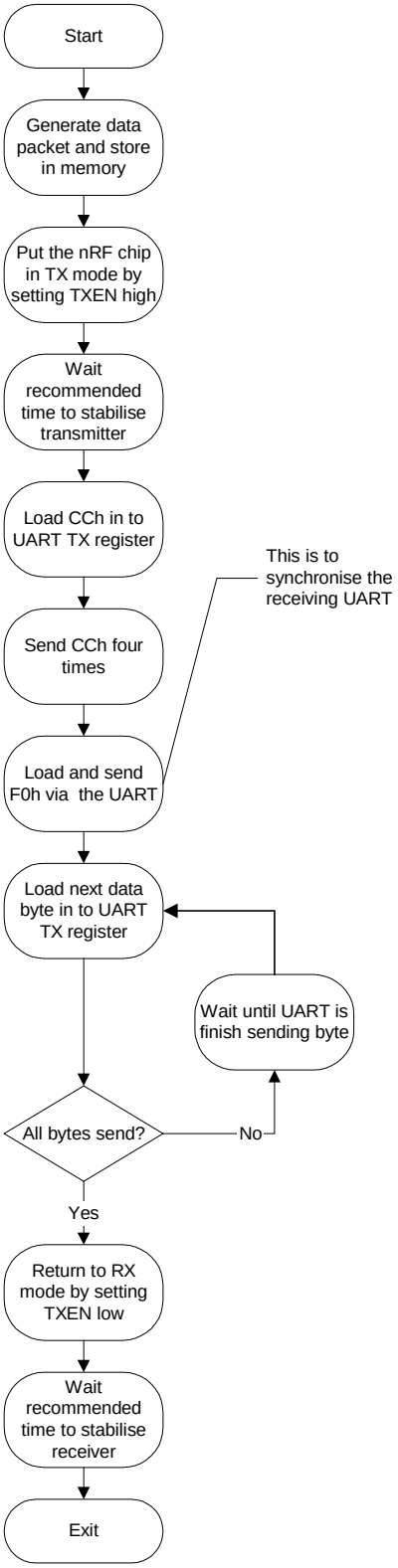
Appendix

Flowchart for oversampling & interpretation of received data stream



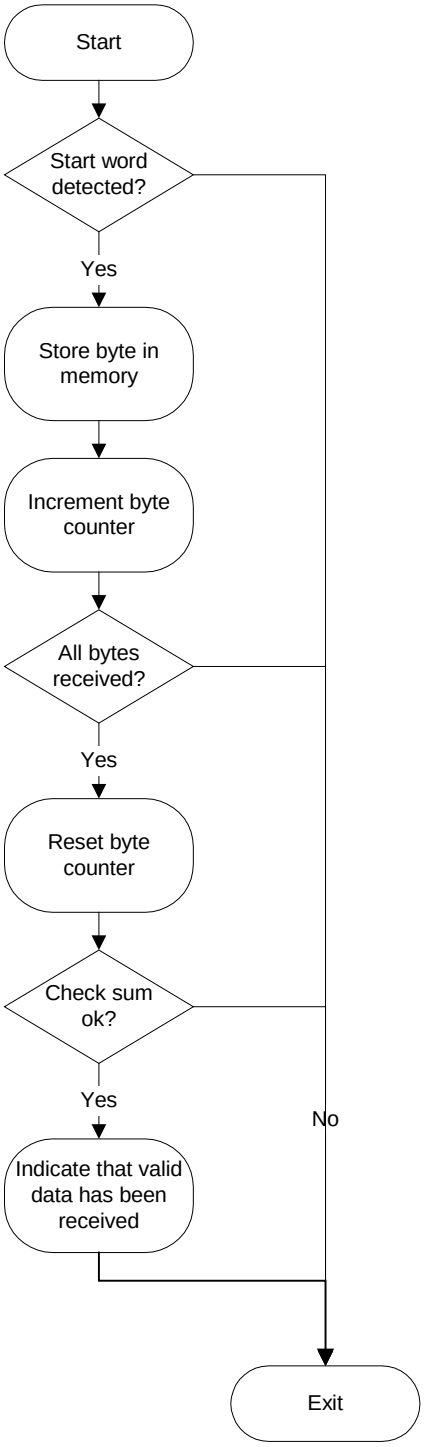
This algorithm must be implemented as a timer overflow interrupt routine in a MCU. Then the timer should be set to time out three times per bit periode.
It is important that this routine is implemented as short and efficient as possible, because even if a MCU runs at 4 MHz cyckle, sampling a 20 kbits/s bit stream three times each bit gives only 66 cycles to handle each sample.

Flowchart for UART transmission.



This flowchart shows how to send a data packet via hardware UART.

Flowchart for UART receive.



This routine must be run each time the UART has received a whole byte. In most microcontrollers with hardware implemented UART, an interrupt is send to the MCU each time a complete byte is received. Detection of start word can be done by setting a bit in a dedicated control register.

LIABILITY DISCLAIMER

Nordic VLSI ASA reserves the right to make changes without further notice to the product to improve reliability, function or design. Nordic VLSI does not assume any liability arising out of the application or use of any product or circuits described herein.

LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Nordic VLSI ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic VLSI ASA for any damages resulting from such improper use or sale.

Application Note. Revision Date: 16.12.02

Application Note order code: 131202-nAN400-07

YOUR NOTES

Nordic VLSI - World Wide Distributors

For Your nearest dealer, please see http://www.nvlsi.no

Nordic VLSI ASA

Main Office:
Vestre Rosten 81, N-7075 Tiller, Norway
Phone: +47 72 89 89 00, Fax: +47 72 89 89 89

Visit the Nordic VLSI ASA website at http://www.nvlsi.no