Nordic VLSI ASA

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

1. General

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

- Oversampling
- edge detection, and
- Use of a UART (Universal Asynchronous Receiver Transmitter.)

Whilst the nRF™ chips requires no coding of data, the fact that you are using wireless communication will necessitate the use of a receiving algorithm that can separate valid data from noise-created or spurious data. Therefore data has to be sent in packages, and a good start is to define the packet format.

2. Packet format

A packet has to begin with a preamble. A preamble with 2 bytes is suggested; the first byte should consist of: 10101010 (‘AA’), to cycle the receiver. The second byte of the preamble should be an identification to indicate the start of a packet. The rest of the packet containing address and data should follow.

As an example: One byte with an address followed by the packet payload (one or two bytes) and finally a checksum. In general, the best practice is to keep the length of each package as short as possible.

3. Oversampling

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 1 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

Table1: 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. Edge detection

If you choose to use bit edges as a way to achieve synchronisation, you should use the edges in the bit stream from the nRF™ chip to trigger an interrupt in the microcontroller. The interrupt routine can then calculate the right sampling time in the middle of each bit. This method can be implemented so it adjusts each time a bit edge occurs, and synchronisation will be achieved. However, using only one sample per bit will give a routine that is much more sensitive to noise than oversampling. A more complex version of this method is to let it be able to detect, and synchronise with different incoming data rates. If this functionality is not needed, oversampling should be used instead.

5. UART

Developing a system connected to a PC's serial port tempts to use the UART packet format directly on the air. Generally this is no problem when using a nRF™ chip, because it does not require any encoding of data. The challenge when using a UART is that some are more sensitive to noise, especially edge jitter. Therefore, in order to

ensure that the UART works correctly you must be able to detect its start and stop bits.
This means that a data packet format is required, even if a UART is chosen, but each byte in the data packet can be encoded in UART style (one start bit, the payload and a stop bit).
When using a UART it must be synchronised with the first start bit, by sending logic high for more than one byte duration of time before the first byte. The UART should be tested carefully to ensure that it would be able to detect each bit correctly even if the data has edge jitter.
For transmission the following protocol is suggested; first byte - 10101010 ('AA') to cycle the receiver. Then steady logic "1", long enough to flush and set-up the receiving UART for proper transmission - this should normally be more than ten bits duration. The receiving UART should then be in synchronisation with the transmitter and you can start transmitting the required datastream.

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.

7. Conclusion

On what criteria should the choice of technique be based ?

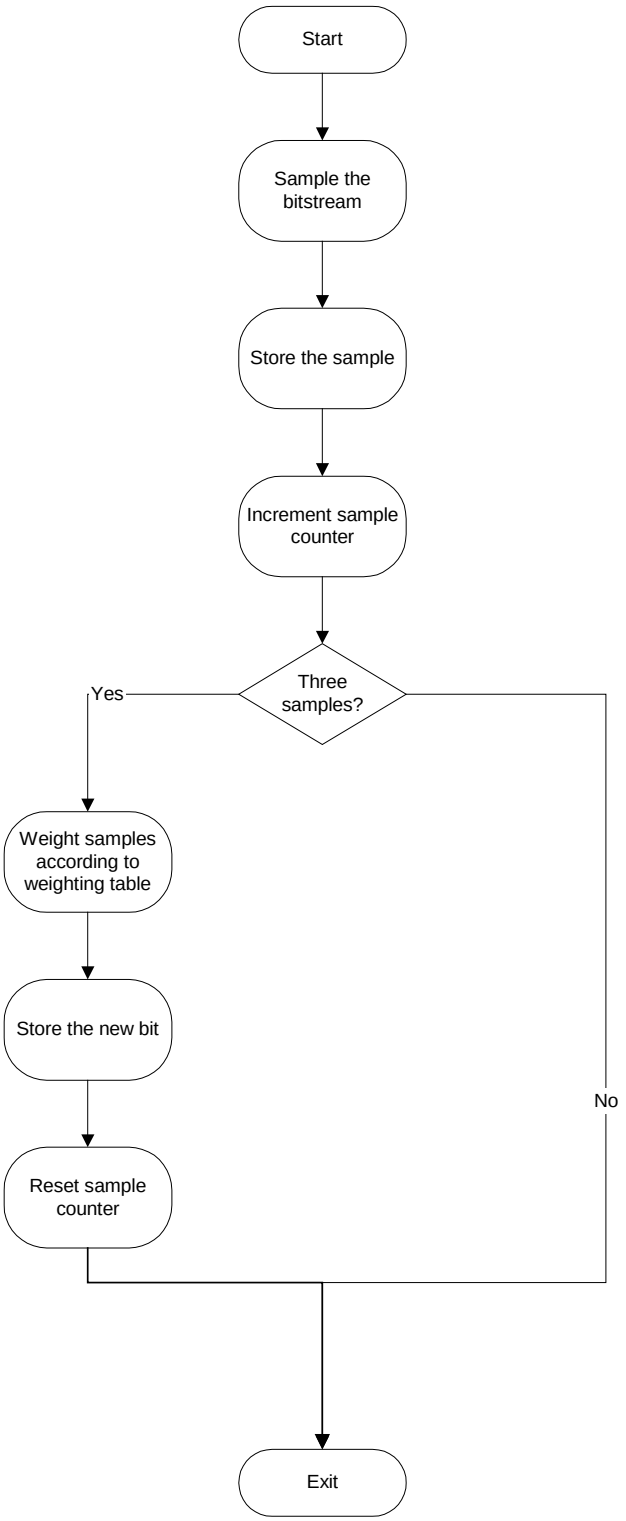
Well, if the link is required to be extremely reliable and resistant against noise, you should choose the over-sampling technique. This technique uses greater capacity of the microcontroller because it has to interrupt three times for each bit.

If the link needs to be able to handle different datarates, the edge detection algorithm should be chosen. This will be a more complex algorithm than the over-sampling algorithm, and will use even greater capacity of the microcontroller.

If the microcontroller has a hardware UART, this can be used to handle your receive and transmit routines. This technique uses the least capacity of the microcontroller, but success will be depended on the quality of the UART selected, therefore a test of the UART is recommended. The UART implementation will provide a very quick turnaround time for setting up and evaluating a RF-link. However the RF-link range will not be as long or reliable as a sampling algorithm which is more optimised for RF.

In most applications, oversampling will be the better choice of sampling technique, especially if long range and high reliability is required.

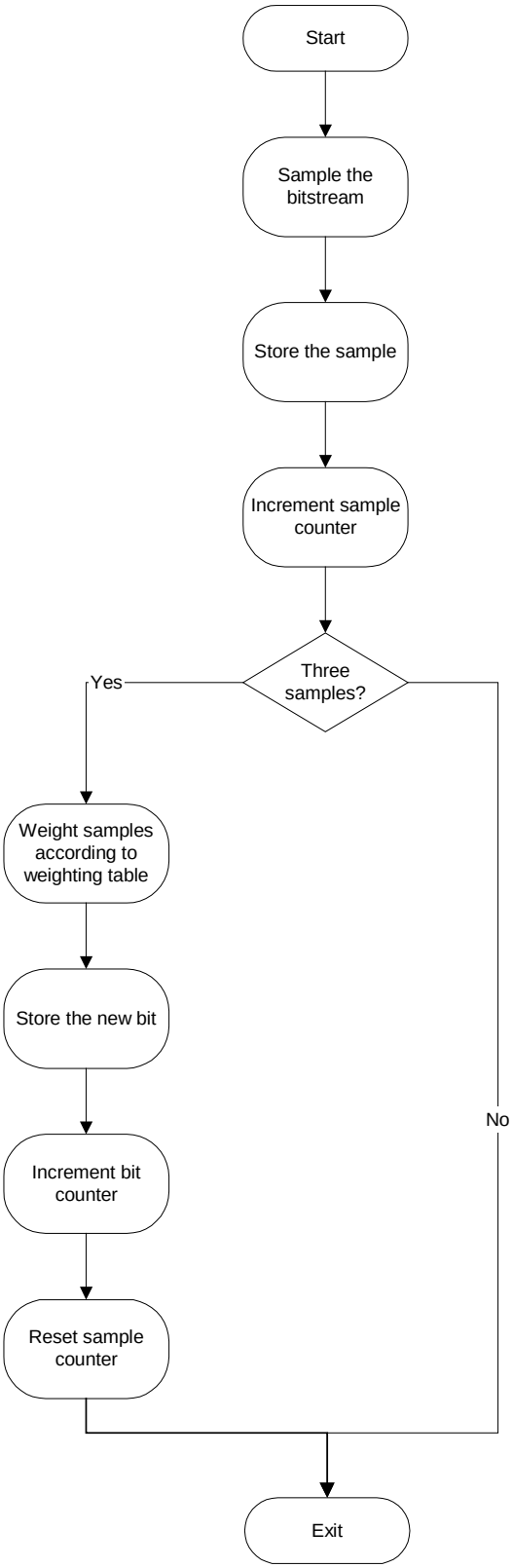
Flowchart for oversampling & interpretation of received data stream



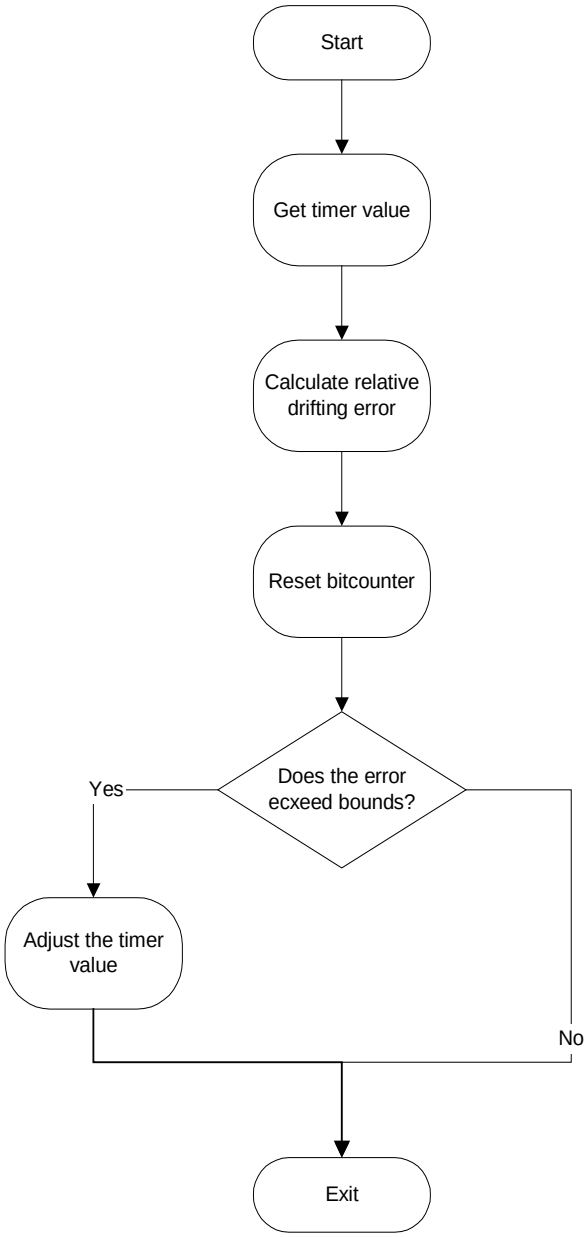
This algorithm must be implemented as a timer overflow interrupt routine in a MCU. Then the timer should be set to time out three times per bit periode.
It is important that this routine is implemented as short and efficient as possible, because even if a MCU runs at 4 MHz cyckle, sampling a 20 kbits/s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

Flowchart for edge sensing.

Receiving algorithm

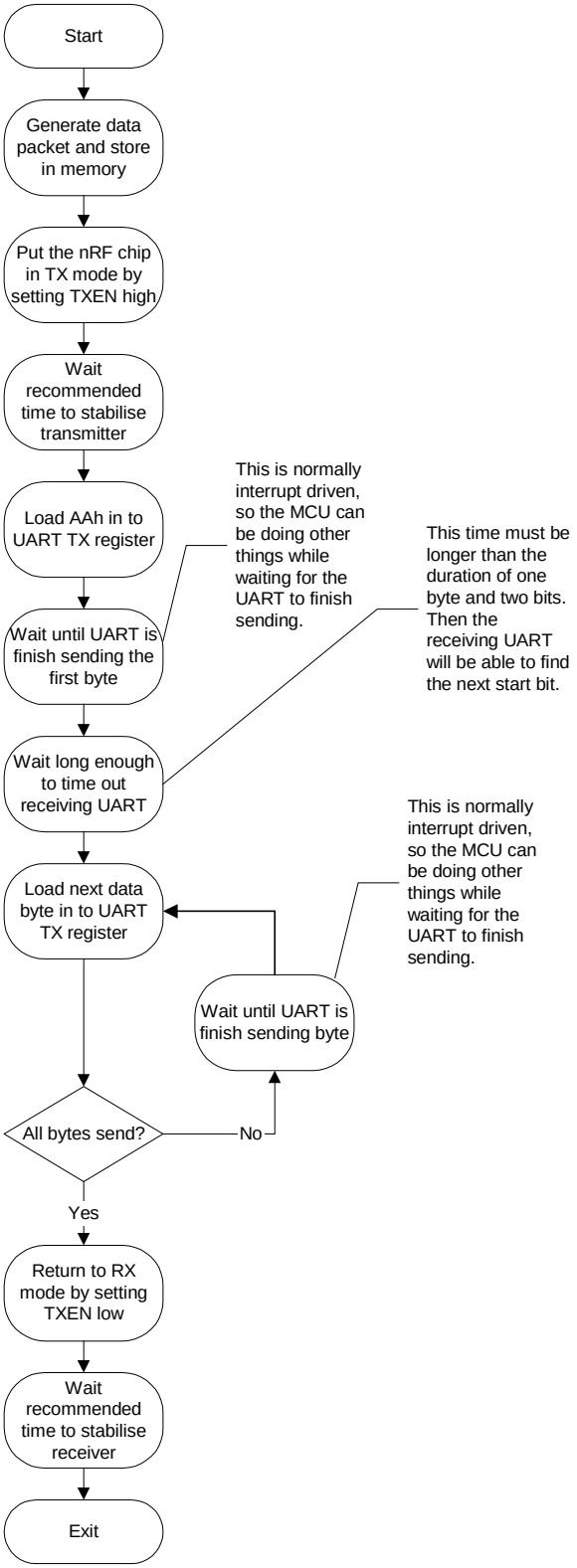


Edge sensing and adjustment algorithm



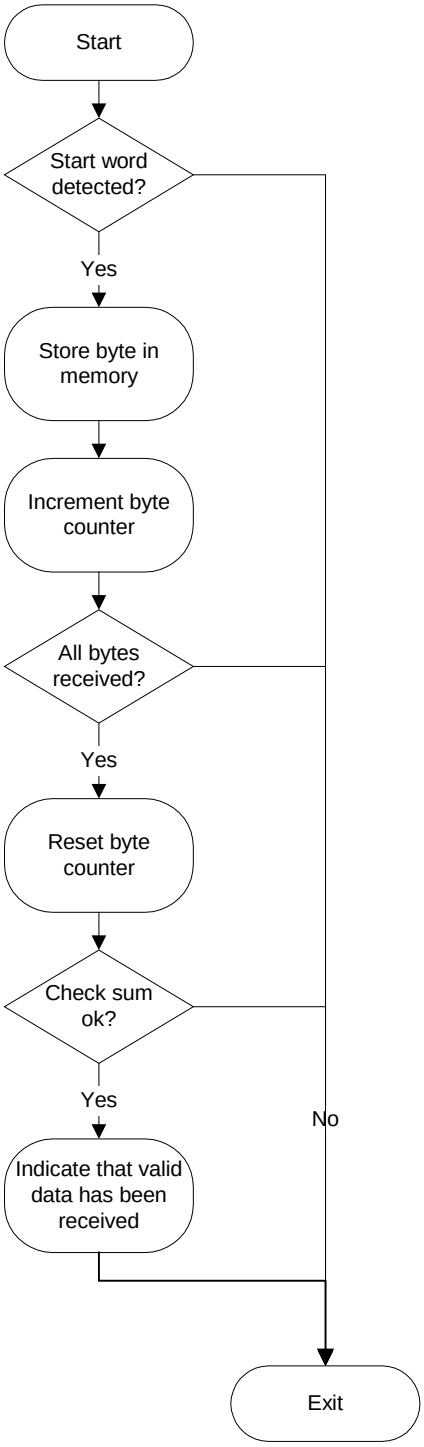
The receiving algorithm should be operated by a timer, as for the oversampling case. But to adjust the timer and avoid errors due to drifting in sampling time, the edge sensing algorithm should be run as a routine for the external interrupt source. This means that the incoming bitstream should be tracked to a pin on the MCU that can give interrupt from a change in the logical value of a signal.

Flowchart for UART transmission.



This flowchart shows how to send a data packet via hardware UART. To implement this routine you should consider use of the UARTs ability to make interrupts to the MCU. The parts of this routine that requires waiting for the UART to finish, should be implemented to run as an interrupt routine.

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: 22.10.2001.

Application Note order code: 221001-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
E-mail: nRF@nvlsi.no
Visit the Nordic VLSI ASA website at http://www.nvlsi.no