nRF24LE1 firmware update over-the-air

nAN-18

Application Note v1.0

Key words

- Firmware update over-the-air protocol
- nRF24LE1
- Dedicated protocol
- Fully functional project and graphical user interface included with this application note

2011-09-06

Contents

1 Introduction

Being able to add new and better functionality to a product after it has been shipped out can extend its lifetime, and allow for better customer support.

This application note along with the included project files will enable you to implement over-the-air firmware updates for nRF24LE1. It contains a detailed description of the nRF24LE1 firmware and the protocol created and used for this application note's project. The application note also outlines the roles of the PC host application and the nRF24LU1+ USB-RF adapter. A project compiled in Keil™'s µVision with all the necessary code is available with the nRFgo Software Development Kit.

Moreover the application note explains how the host application on the PC, and the nRF24LU1+ USB-RF adapter work, and explain which roles they assume in the firmware update. The project files also contain a PC application with a graphical user interface created in C# using Windows Forms.

In our example the firmware to be updated resides in a remote device, only connected to a host by RF. This remote device consists of an nRF24LE1 RF System-on-Chip with flash memory. The host used in this application note is a Windows PC connected to an nRF24LU1+ over USB.

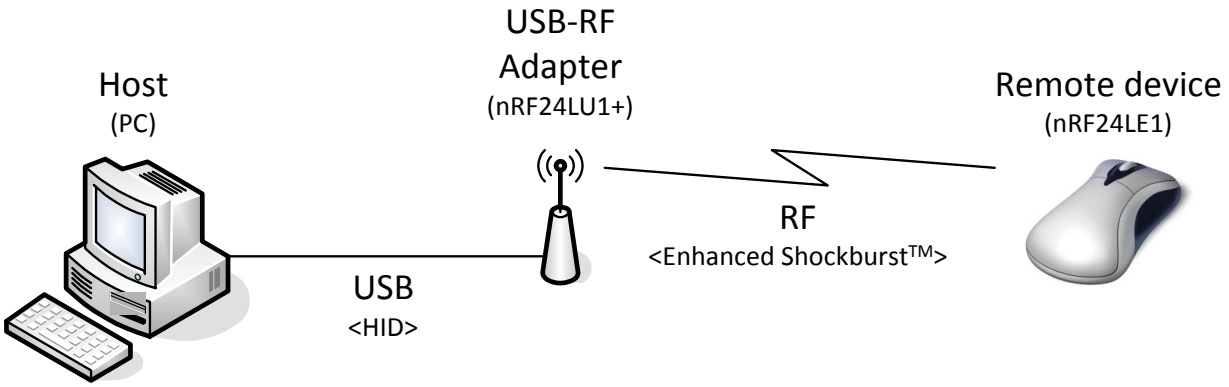


Figure 1. Communication from user to remote device

The remote device in Figure 1. represents the end product that you are developing. Its firmware will consist of two parts: the update firmware that executes as the boot loader, and the application-specific firmware that can be updated. The update firmware runs for a short period at each startup, before running the product-specific firmware. We refer to the product-specific firmware as *new firmware* in this application note, while update firmware will be referred to simply as *nRF24LE1 update firmware*.

A separate application note nAN-22 describes the USB communication between the PC host application and the nRF24LU1+ USB-RF adapter, using the HID interface. nAN-22 also describes the graphical user interface for the PC host application, created in C# using Windows forms.

2 Challenges with firmware updates

When firmware is stored in flash memory, you can re-program it to fix bugs or add new functionality to existing devices. There are two options for writing to the nRF24LE1 flash memory; through the nRF24LE1's MCU or directly over the nRF24LE1 SPI programming bus. In several types of devices the latter is impractical, especially in cases where the only available communication interface with the device is over RF.

When writing firmware to flash memory using the MCU, you must not overwrite the code that executes the update. If your update firmware crashes because of code overwrite, it is unlikely that it will be able to execute correctly again, and you may lose the re-programmability of the device. In the worst case, this may render your product useless.

Writing to flash memory can only be done after you first erase the location to which you want to write. On the nRF24LE1 you erase pages of 512 bytes of flash memory, which of course means that the nRF24LE1 update firmware cannot be located on the same 512 byte page to which the new firmware is written.

Another task involves initiating the update firmware. If you assign this role to the new firmware, the update firmware may become inaccessible if the new firmware contains errors. The update firmware could also become inaccessible if transmission of the new firmware is aborted while its code is being written to flash memory. You should therefore *not* let the new firmware initiate the nRF24LE1 update firmware.

3 Theory of operation

In this chapter we will explain necessary details related to the firmware update solution; the data format, and the protocol used to transfer it from host to remote device.

3.1 Data format

Firmware is often compiled to HEX files. While the format is often vendor specific, the HEX files generally comply with official standards. For this solution the Intel-HEX or HEX-80 format from Keil is used.

Start code ':'	Byte count	Memory address	Record type	Data	Checksum
1 byte	1 byte	2 bytes	1 byte	*n* bytes	1 byte

Table 1. Data structure for Intel HEX format

To avoid confusion about the format names, note that HEX-80 is the Intel-HEX format which is produced by the Keil tools.

Every HEX record starts with the ':' character. This is useful for parsing the HEX file, but does not provide any important information beyond that. The Byte Count value is the number of bytes of the Data field, and for HEX-80 records, this is not larger than 0x10 or 16 bytes.

The Memory Address specifies where in the flash memory the data should be written.

Note: The HEX-80 format does not require records to be sorted in increasing addresses, hence data is not necessarily written in order.

The Record Type field will usually be set to 0x00, indicating that this is a normal record. The only other option in HEX-80 is 0x01, which is the end-of-file indicator.

The Data field contains the code that should be written to the flash memory. This is the hexadecimal representation of the binary code of the new firmware. This field is of variable length and the length is specified by the Byte-Count field.

The Checksum field contains the least significant byte of the two's complement of the sum of the other fields, not counting the record mark field.

An example of this is ":0300300002337A1E". See Table 2.

Start code ':'	Byte count	Memory address	Record type	Data	Checksum
:	03	00 30	00	02 33 7A	1E

Table 2. Example of Intel HEX format

Add Byte-Count, Memory-Address, Record-Type, and Data field together:

03 + 00 + 30 + 00 + 02 + 33 + 7A = E2

Find the two's complement to this sum:

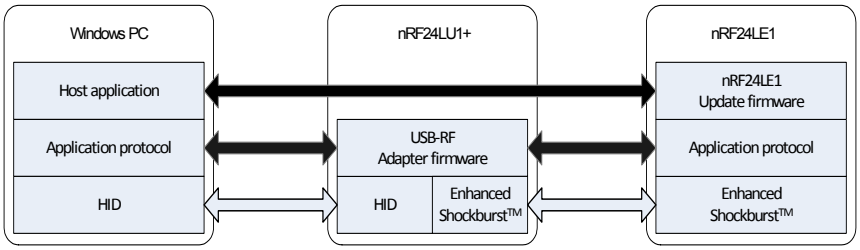
Bitwise-inverse (E2) + 1 = 1E

When checking the checksum you can simply add all the fields except the Start Code, and check that this number is equal to zero. Disregard any overflow the summation will cause.

03 + 00 + 30 + 00 + 02 + 33 + 7A + 1E = 00 (valid checksum)

3.2 Application protocol

The application protocol created for this project is used for end-to-end communication between the host and the remote device. In the included project it is built upon HID and Enhanced Shockburst™ as seen in Figure 2. but it can be built upon any transport protocol. An explanation of the application protocol follows below.



Note: The arrows with dark fill show a logical communication flow. The arrows with light-colored fill show an actual communication flow.

Figure 2. Communication abstraction layers

In the application protocol used between the host and the remote device, the communication is always initiated by the host. The host sends commands to the remote device, and the remote device executes the command and replies with an acknowledgement (`Ack`) if successful, or a not acknowledgement (`Nack`) if not.

3.2.1 Init

This command establishes the connection between the host and the remote device.

3.2.1.1 Functional description

The `Init` command is sent by the host to establish a connection with the remote device. It does not contain any content. If the remote device is listening on that channel, it will reply with an `Ack` containing its model number and the firmware version number of the firmware currently installed on it. When the host receives the `Ack`, the connection is established. There is no `Nack` for the `Init` command.

See Figure 3. on page 7.

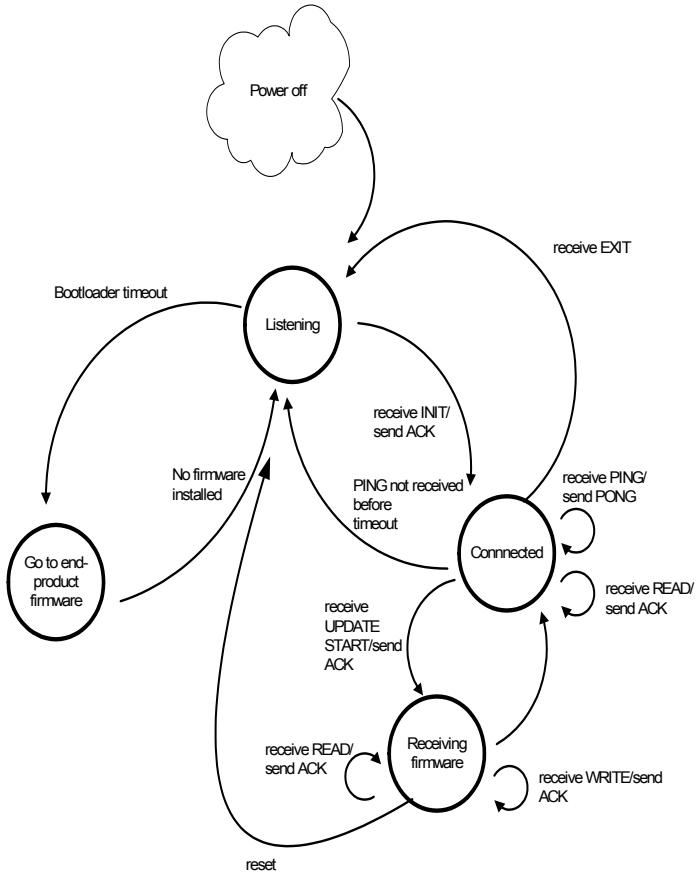


Figure 3. nRF24LE1 update firmware state diagram

3.2.1.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x02	Init command
Content			
No content			

Table 3. Data structure for Init command

3.2.1.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x07	Acknowledge
Content			
Product number	1		Number identifying product
Firmware version	1		Version number of installed firmware

Table 4. Data structure for Ack reply

3.2.2 Update start

This command initiates the transfer of new firmware.

3.2.2.1 Functional description

The `Update start` command is sent when the host wants to update the firmware on the remote device. The content contains the size of the new firmware in bytes, the reset vector of the new firmware, a specified version number used to recognize the firmware once it is installed, and a checksum to ensure that this command is correct. The checksum is calculated and checked in the same way as for the HEX format (see section 3.1 on page 6), using the sum of the other fields of the content.

If the remote device is ready to receive the new firmware it will reply with an `Ack`, otherwise it will reply with a `Nack` containing the error code identifying the error that has occurred.

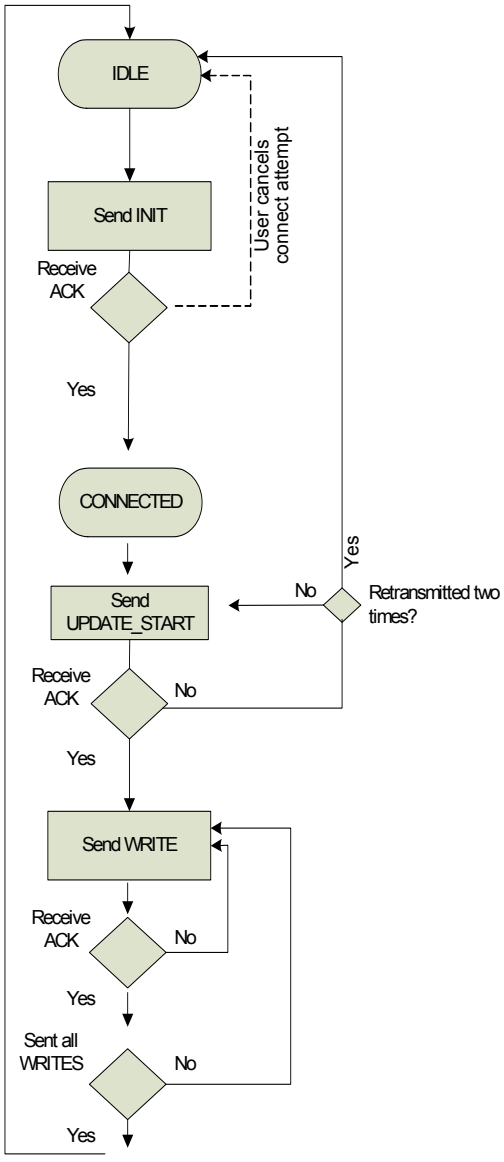


Figure 4. Host application flow chart

3.2.2.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x03	Update start command
Content			
Byte count total	2		Size of firmware in bytes
Reset vector opcode	1	(0x02)	Opcode part of new firmware's reset vector
Reset vector address	2		Address part of new firmware's reset vector
Firmware version	1		New firmware's version number
Checksum	1		Message checksum

Table 5. Data structure for Update-start command

3.2.2.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x07	Acknowledge (`Ack`)
Content			
No content			

Table 6. Data structure for Ack reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x08	Not acknowledge (`Nack`)
Content			
Error code	1		Error code (see Table 18. on page 15)

Table 7. Data structure for Nack reply

3.2.3 Write

The `Write` command contains one HEX record.

3.2.3.1 Functional description

The `Write` command contains one HEX record to be written to the remote device's flash memory. The Start-code field, see section 3.1 on page 6, of the HEX record has been stripped away before the host sends it. Before writing, the remote device should verify that the HEX record is correct and that the address it attempts to write to is legal. The replies are similar to those for the `Update start` command.

3.2.3.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x05	Write command
Content			
Byte count	1		The number of bytes (*N*) of the Data field
Address	2		The address in flash memory to which Data should be written
Record type	1	(0x00)	HEX record type
Data	*N*		Data to be written to flash memory
Checksum	1		HEX record checksum

Table 8. Data structure for Write command

3.2.3.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x07	Acknowledge (`Ack`)
Content			
No content			

Table 9. Data structure for Ack reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x08	Negative acknowledge (`Nack`)
Content			
Error code	1		Error code (see Table 18. on page 15)

Table 10. Data structure for Nack reply

3.2.4 Update complete

This command renders the transfer of new firmware complete.

3.2.4.1 Functional description

The `Update complete` command is sent when the host has sent all the HEX records of which the firmware consists. If the remote device has indeed received all the bytes of data that it should, according to the byte count total field in the `Update start` command, it marks the new firmware as bootable and can start executing it after the host terminates the connection. The replies are similar to those for the `Update start` command.

3.2.4.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x06	Update complete command
Content			
No content			

Table 11. Data structure for Update complete command

3.2.4.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x07	Acknowledge (`Ack`)
Content			
No content			

Table 12. Data structure for Ack reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x08	Not acknowledge (`Nack`)
Content			
Error code	1		Error code (see Table 18. on page 15)

Table 13. Data structure for Nack reply

3.2.5 Read

This command specifies the number of bytes from the address in flash memory.

3.2.5.1 Functional description

The `Read` command is used by the host to verify that the new firmware is installed correctly. In the content it specifies the number of bytes it wants to read, and the address in the remote device's flash memory. The host can check these bytes against the HEX file it has transferred. The reply is an `Ack` with the bytes the host has requested. To make sure that remote device does not start executing a faulty firmware, this command can be sent before sending the `Update complete` command. A `Nack` with an error code can be sent if the host requests a read of an illegal address.

3.2.5.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x04	Read command
Content			
Byte count	1		Number of bytes (*N*) to read
Address	2		Flash memory address

Table 14. Data structure for Read command

3.2.5.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x07	Acknowledge (`Ack`)
Content			
Data	1		*N* bytes of data from address

Table 15. Data structure for Ack reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x08	Negative acknowledge (`Nack`)
Content			
Error code	1		Error code (see Table 18. on page 15)

Table 16. Data structure for Nack reply

3.2.6 Exit

This command terminates the connection.

3.2.6.1 Functional description

The `Exit` command terminates the connection between the host and the remote device. This command does not need `Ack` or `Nack`.

3.2.6.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x01	Exit command
Content			
No content			

Table 17. Data structure for Exit command

3.2.6.3 Reply

This command has no reply.

3.2.7 Ping

This command is a connection test.

3.2.7.1 Functional description

Note: This command is only sent by the USB-RF adapter to verify the connection between it and the remote device.

The `Ping` command is sent to verify that the USB-RF adapter and the remote device are still connected. The remote device replies to a `Ping` with a `Pong`. If the USB-RF adapter does not receive a `Pong` reply on a set amount of `Ping` sent, it will consider the connection as lost, and should try to reconnect to the device. The remote device also considers the connection lost if no `Ping` commands are received within a certain amount of time.

3.2.7.2 Message format

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x08	Ping
Content			
No content			

Table 18. Data structure for Ping command

3.2.7.3 Reply

Message field/parameter	Value size (bytes)	Data value	Description
Header			
Message type	1	0x09	Pong
Content			
No content			

Table 19. Data structure for Pong reply

3.2.8 Error codes

Byte value	Error description
0x01	Lost connection
0x02	Checksum failed
0x03	Illegal address
0x04	Illegal size

Table 20. Error codes

4 Implementation

4.1 Project setup

To set up the attached project correctly you will need the following hardware, software, and project structure:

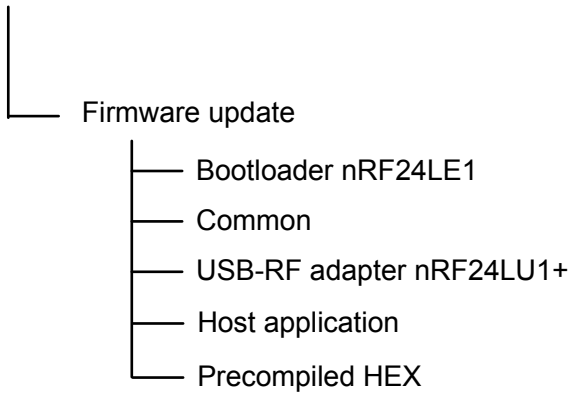
Required hardware

- nRFgo Development Kit nRF24LE1-F16Qxx-DK, nRF24LE1
- nRFgo Development Kit nRF24LU1P-FxxQ32-DK, nRF24LU1+
- 2 x nRFgo Starter Kit nRF6700
- PC workstation with USB

Required software

- Keil µVision V4
  - C51 Compiler
  - BL51 Linker
- nRFgo Software Development Kit (SDK) version 2.2
- Windows 7 (32-bit, 64-bit)
- Microsoft Visual C# 2010 Express

Project structure



If you have not already done so, you should install Keil µVision and the nRFgo SDK. Once you have done this, you can place the **Firmware_updater folder** in the **…\nRFgo SDK 2.2.0.270\source_code\projects\nrfgo_sdk** folder. This will ensure that the predefined project files will have the correct include paths to compiler- and hal-directives. If you wish to place the project at a different location you will have to set the include paths manually in Keil µVision. These are found under **Project – Options for Target '…' - C51 – Include Paths**.

You will find the Keil µVision project files for the nRF24LE1 update firmware in the Bootloader nRF24LE1 folder. A precompiled HEX file for the nRF24LE1 update firmware is found in the precompiled HEX folder. This HEX file can be flashed directly to the chip if you do not want to build the project files before you test out the functionality.

Demo firmware that can be used as the new firmware, is found in the precompiled-HEX folder.

4.2 Implementing the nRF24LE1 update firmware

Most developers will want to modify the nRF24LE1 to the purposes of their specific product. When doing so you should make the update firmware and the new firmware as independent as possible. To accomplish this you should:

1. Keep the update firmware and the new firmware separate by locating them on different pages in the flash memory. See section 4.2.1.
2. Build the nRF24LE1 update firmware as a boot loader. See section 4.2.2 on page 18.
3. Make sure that the update firmware cannot erase flash pages where its code is located. See section 4.2.3 on page 18.
4. Do not use interrupts in the nRF24LE1 update firmware. See section 4.2.4 on page 19.

Sections 4.2.5 on page 19 to 4.2.7 on page 21 describe other factors you need to consider when creating the nRF24LE1 update firmware.

4.2.1 Relocating the nRF24LE1 update firmware

Since the flash memory must be erased before a bit value is changed from 0 to 1, and it is only possible to erase entire pages, it will simplify matters considerably if you locate the update firmware at different pages than the new firmware. Our solution locates the update firmware at the last available pages in the flash memory.

To achieve this in Keil µVision 4, go to Project - Options for Target - Target – Off-chip Code Memory, and set **Start** to 0x3400 and **Size** to 0x0C00. These values are specific to this project and are decided based on the code size of the update firmware. If you make any changes to the project make sure to check that code size does not exceed 3 kB, and if so you must change the Start and Size value accordingly. You must then also remember to change the BOOTLOADER_PAGES define-macro in the header file for the nRF24LE1 update firmware to the correct number of pages.

4.2.2 Boot loader

To solve the problem regarding initiating the nRF24LE1 update firmware, you should implement it as a boot loader. This will ensure that you have the update capabilities of the device regardless of what new firmware is installed.

When the nRF24LE1 boots it executes whatever it finds at address 0x0000 in code space. This first instruction is called the reset vector and is a long jump to the start address of your program. You have to make sure that the reset vector pointing to the update firmware is not overwritten by a reset vector from the new firmware. If this happens, the boot loader with the update firmware would not be executed after booting the device, and it would jump to the new firmware instead.

4.2.3 Erasing flash pages

You must make sure that the update firmware only can erase pages not containing the update firmware. Since the update firmware is located in the last six pages, this means that you can only erase the first 26 pages. These pages available for the new firmware should be erased when the remote device receives an `Update start` command from the host.

The following code lines show how the software prevents the erasing of pages where the boot loader resides:

```
for (i = 1; i < FLASH_FW_PAGES; i++) {
  hal_flash_page_erase(i);
}
```

Moreover, “FLASH_FW_PAGES” is defined to limit the total pages that can be erased:

```
#define FLASH_FW_PAGES          FLASH_TOTAL_PAGES - FLASH_BL_PAGES
```

Since this will erase the first page containing the reset vector, you must make sure to write the reset vector back after the first page has been erased. This should be done preferably as soon as possible to minimize the risk of losing the reset vector.

4.2.4 Avoid using interrupts

Interrupts require an interrupt jump vector to be stored at locations specified in the nRF24LE1 Product Specification. This jump vector will point to your specified interrupt function. However, if you choose to use the RF interrupt for the communication, you will run into some problems.

Firstly, the new firmware is likely to be an application in need of the very same RF interrupt. This means that you have to find a way to jump to either the nRF24LE1 update firmware's function or the new firmware's function depending on which of them is currently executing.

Secondly, you would need to extract the new firmware's interrupt jump vector when writing to flash, in order to make sure that it does not try to overwrite the nRF24LE1 update firmware's interrupt vector. As mentioned earlier, you cannot overwrite a flash memory location, as it has to be erased before a write. We are doing something similar to this with the reset vector, but this is a more complicated case.

The simpler solution you should use is to poll the RF IRQ bit, and explicitly call the "interrupt"-function when the RF IRQ bit is set. This function should look just like a normal interrupt function, but without using the function-name macro specified in the device header.

4.2.5 Storing important variables

There are several variables that you need to preserve in between device power resets. The nRF24LE1 update firmware needs to know if there is firmware installed on the device, so that the boot loader does not start executing unverified code. The reset vector to the new firmware needs to be stored so that the boot loader knows the entry point to the new firmware. Each new firmware also has a firmware version number associated with it.

For code reference to this section, please see the function "startFirmwareUpdate()" in the file "main.c" which is located in the project "bootloader_nRF24LE1".

You should store these variables so that they are kept between power resets. It is therefore advisable to store these in the non-volatile flash memory. One drawback with this solution is that it puts additional limitations on the new firmware. We have placed these variables in the first bytes of the last page of flash memory, page number 35. This means that if the new firmware erases this page for any reason, and does not write these bytes back, the boot loader will not jump to/find the new firmware. If this limitation is too severe for your application, you have to find another way to store these variables.

4.2.6 Channel switching and connection timeout

The nRF24LE1 update firmware and the nRF24LU1+ USB-RF adapter should be able to use several channels for their RF communication. If a channel cannot be used for communication because it is jammed by other sources, a connection should be established on another channel instead.

You should select a set amount of channels you wish to operate on. In the included project files we have chosen three channels, but you may choose more than that. Before a connection has been established the nRF24LE1 update firmware on the remote device should listen to a channel for a short period of time, before it changes to the next channel.

```
// Go to next channel
ch_i = (ch_i+1)%3;
hal_nrf_set_rf_channel(default_channels[ch_i]);
```

Once a connection has been established the channel should be kept until the connection is terminated or lost.

The USB-RF adapter will consider the connection between it and the nRF24LE1 update firmware lost, if it does not receive a `Pong` reply to its `Ping` command. Likewise, you should implement a connection timer in the nRF24LE1 update firmware, that when it times out changes the nRF24LE1 update firmware's state from Connected to Listening. This timeout must be longer than the longest interval in between messages sent received from the USB-RF adapter.

```
else if (state == CONNECTED) {
  connection_timer++;
    if (connection_timer > CONNECTION_TIMEOUT) {
      state = LISTENING;
  }
}
```

4.2.7 Resetting RF parameters before leaving the boot loader

Table 21. lists the different parameters used for the Enhanced Shockburst™ communication in the included project. These values can be used as is, but you will most likely want to choose your own.

It is a good practice to reset these values back to their reset values, shown in the last column in Table 21. before you exit the boot loader. If this is not done you could end up with mysterious errors in the new firmware. This is yet another way to make the nRF24LE1 update firmware and the new firmware more independent.

Parameter name	Parameter value		Reset value
Channel	0x02, 0x06, 0x51		0x02
TX address	0xBADA551337		0xE7E7E7E7E7
PIPEO address	0xBADA551337		0xE7E7E7E7E7
Auto retry	nRF24LE1: 5 retries, 500 µs wait	nRF24LU1+: 2 retries, 250 µs wait (searching) 8 retries, 500 µs wait (connected)	3 retries, 250 ms wait
Payload width	0x20 (32)		0x00
Operation mode	nRF24LE1: 1 (PRX)	nRF24LU1+: 0 (PTX)	0
Power mode	1		0

Table 21. Enhanced Shockburst™ communication parameters

4.2.8 Boot loader duration

Since the nRF24LE1 update firmware executes after each system reset, you should keep its duration short so the user does not notice the delay, but long enough so that it can receive the `INIT` command.

In the included project this duration is controlled by two timeouts: CHANNEL_TIMEOUT and BOOTLOADER_TIMEOUT. The first controls how long the update firmware listens to a channel before switching to the next, while the second controls how many times it changes channel before executing to the new firmware.

4.3 PC host application

To achieve correct transfer of the new firmware, the PC host application must do the following:

- Verify HEX file
- Initiate transfer correctly
- Optional: Verify the new firmware on the remote device by reading it back and comparing it to the input file.
- Send `Update complete` when done.

The update firmware on the remote device only checks the checksum, and that the memory address it writes to is within the available flash pages. If the host sends a HEX record that tries to overwrite another recently written HEX record, the update firmware will not detect this. You must make sure that the host application only sends valid HEX files to the update firmware.

To initiate an update, the host needs to send the size of the firmware in bytes. This is not the size of the HEX file but the sum of all the byte-count fields. It also needs to send the reset vector of the new firmware and a version number associated with the new firmware.

Verifying that the update has been successful can be done by reading back data from the remote device, according to the HEX file. If the data read from the device is the same as in the HEX file, the firmware update has been successful, and it can send the Update-complete command. If not, it can try to update again. By verifying the new firmware before you send the Update-complete command, you are sure that the nRF24LE1 update firmware does not start a faulty new firmware.

Regardless of verification or not, the nRF24LE1 firmware must receive an `Update complete` command to start executing the new firmware. The PC host application is described in further detail in the application note nAN-22.

4.4 nRF24LU1+ USB-RF adapter

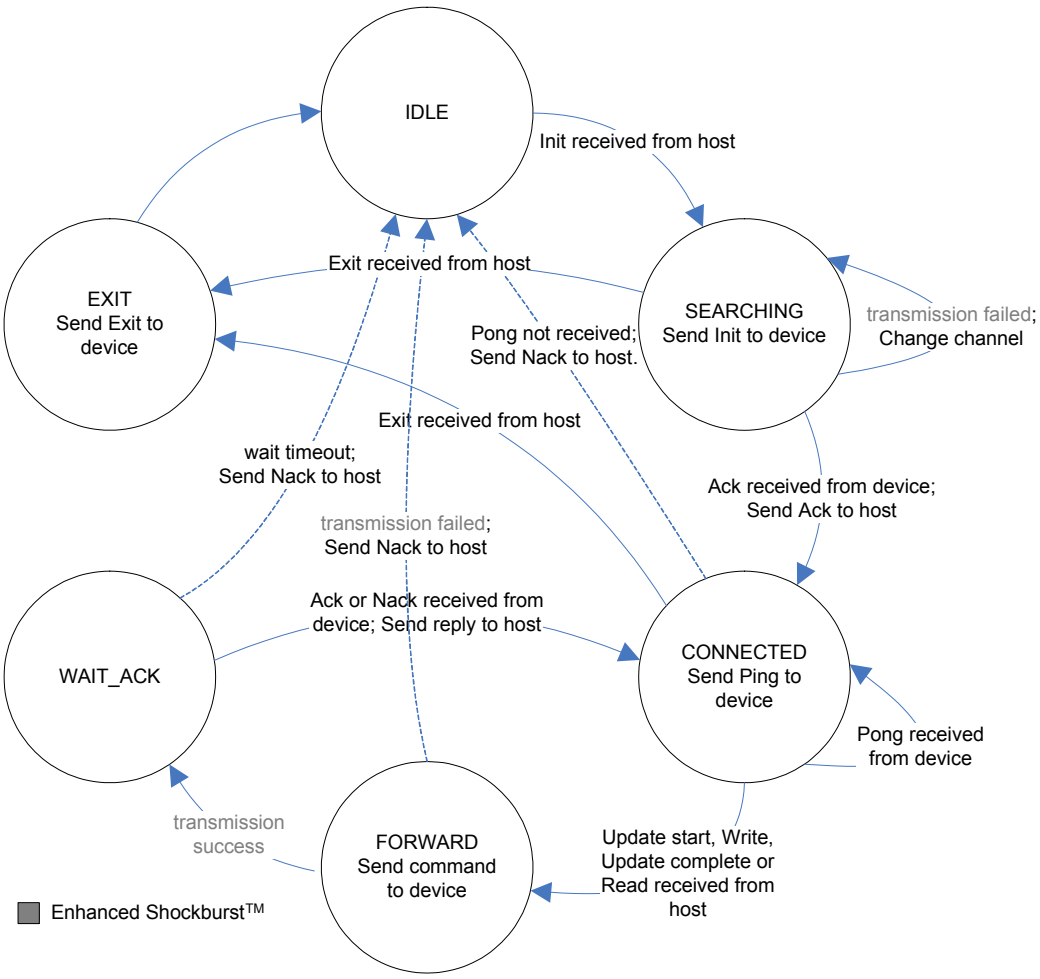


Figure 5. USB-RF adapter state diagram

The nRF24LU1+ chip acts as an adapter between the host application on the PC and the nRF24LE1 update firmware. Figure 5. shows the state machine of the adapter. It shows how the USB-RF adapter reacts to the commands sent by the host. The adapter uses Enhanced Shockburst™ to communicate with

the remote device. In the attached project the adapter uses HID to communicate with the host over USB. How this is done is explained in more detail in the application note nAN-22.

4.4.1 Establishing a connection

If the host wishes to establish a connection it sends an Init command to the USB-RF adapter over USB. The USB-RF adapter should then send the `Init` command over Enhanced Shockburst™ to the nRF24LE1 update firmware on the remote device. Enhanced Shockburst™ will automatically acknowledge this message if the remote device is listening on that channel. If the USB-RF adapter receives an automatic acknowledge, it should wait for an `Ack` reply from the remote device and send this reply back to the host over USB.

If the update firmware on the remote device is not listening on the channel that the `Init` command was sent over, the USB-RF adapter will not receive any automatic acknowledge. It should then change to the next channel and send the `Init` command there.

Once in the CONNECTED state the channel will remain static until the connection is terminated by the host or is lost.

4.4.2 Forwarding commands from host to remote device

The commands received from the host while the USB-RF adapter is in the CONNECTED state, are forwarded to the remote device. The reply from the remote device is sent to the host. If either the transmission fails or the wait timer expires before a reply is received, the USB-RF adapter sends a Nack with the error code for *Connection lost* to the host.

4.4.3 Connection termination

The correct/proper way of terminating a connection is to receive an `Exit` command from the host while being in the CONNECTED state. If the USB-RF adapter is in the SEARCHING state, and has not yet connected to the remote device, it will not stop trying to connect unless it receives an `Exit` command from the host.

While being in the CONNECTED state, the USB-RF adapter sends `Ping` commands to check that it is still connected to the remote device. If it does not receive a `Pong` reply within the set amount of time, it will consider the connection lost and send a `Nack` message to the host over USB to notify the host application of the connection lost.

The connection between the USB-RF adapter and the nRF24LE1 update firmware will be considered lost if the USB-RF adapter fails to receive the automatic acknowledge from nRF24LE1 update firmware when sending over Enhanced Shockburst™.

4.5 New firmware considerations

By having the nRF24LE1 update firmware as a boot loader on the nRF24LE1 chip you place some limitations on the new firmware, if they are to co-exist peacefully. Normally the new firmware can occupy 16 kB of flash memory, but because the boot loader occupies the last six pages, the total size of the new firmware cannot exceed 13 kB.

You should also make sure that the linker does not try to locate any of the new firmware's code on these last six pages. The boot loader should not accept the HEX file, if it tries to write an address above 0x3400.

If the new firmware enters a mode of operation that causes a system reset, any data variables stored to DataRetentive memory will be overwritten by the nRF24LE1 update firmware. The new firmware should therefore not rely on this feature.

The nRF24LE1 update firmware also stores some variables in non-volatile memory. These should not be erased by the new firmware, unless you are sure that they are written correctly back in the same location as before.

Needless to say the firmware should not attempt to erase any of the flash pages in code space that the boot loader depends on. More specifically page 0 and pages 26-31.

With that said, the nRF24LE1 update firmware is designed such that the new firmware can be created with as little knowledge of the boot loader as possible. As long as the new firmware complies with the restrictions outline above, the new firmware should work just as if the boot loader were not there.

5 Discussion

This application note together with the included project files enables you to create a product that is capable of over-the-air firmware updates. The files can be used as-is, as also the USB-RF adapter most likely will be, but typically, the host application on the PC and the nRF24LE1 update firmware should be tailored to your specific needs.

There are several ways to optimize or change the nRF24LE1 update firmware which have not been covered in the previous chapters of this application note. Below we outline some strategies to alter the update firmware if your specific product requires so.

5.1 Optimizing for size

If your product specific firmware requires more than the 13 kB of flash memory available, and you are unable to reduce its size, you can consider optimizing the nRF24LE1 update firmware so that it occupies fewer pages. Remember though, if you reduce the size of the nRF24LE1 update firmware, but it still occupies the same amount of pages, you have not gained any space for your new firmware.

5.1.1 HAL functions

The project employed in this application note uses several functions provided by the HAL to access flash memory and RF communication. These functions are used primarily because they are assumed to be familiar to the reader, and because of good readability. However, they are not highly optimized and could be replaced if size is crucial for your application. A more detailed description of each of the HAL functions is found in the nRFgo SDK documentation.

5.1.2 Reusing shared memory segments to reduce code redundancy

The project files already use a high level of automatic optimization provided by the compiler, so any optimization you want made must be performed manually.

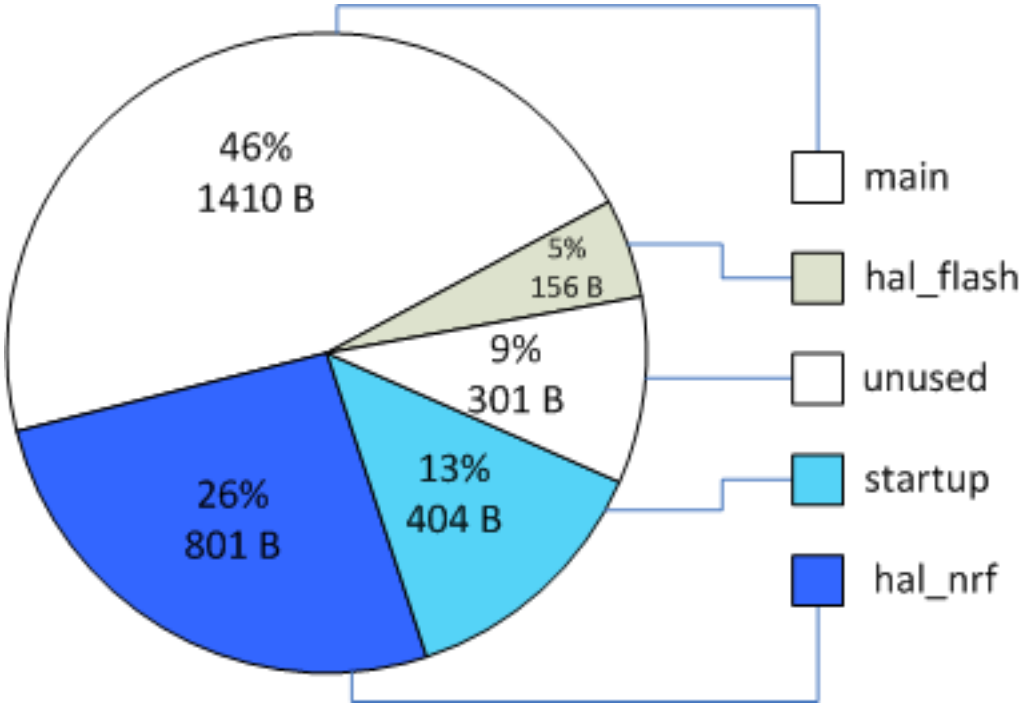


Figure 6. Size of nRF24LE1 update firmware's flash memory segments

Figure 6. on page 25 shows that the size of the HAL_NRF segment is well over one page in size, so if you use the same code in your product specific firmware, you can share this segment between the two firmwares to reduce redundancy. The same can be done for the HAL_FLASH segment, but considering its size it is most likely not worth it unless it is done in addition to the HAL_NRF segment, since they both can be located within two pages.

5.2 Security

This implementation makes no assumptions about security requirements. If you wish to protect your firmware from being known, or disable writing or reading from flash memory for anyone else, you will need to add security mechanisms to your implementation. If you use the implementation provided in the included project, anyone can access the firmware by using the application protocol.

The nRF24LE1 comes with an on-chip encryption/decryption accelerator, which can be used to protect the firmware. For more information on the accelerator see the nRF24LE1 Product Specification. The nRFgo SDK comes with a crypto library based on AES counter mode (CTR). For more information on lib_crypt see the SDK documentation.

If you wish to disable unauthorized access to the chip, you can require an authentication on write commands. The perhaps easiest solution to prevent read access to the firmware is to disable the Read command entirely. This does mean that another solution must be found, if you wish to verify the firmware before enabling it. It is strongly recommended that the firmware is verified before it is enabled, and can start executing.

5.3 Overcoming challenges to RF communication

The application protocol presented in this application note is built upon the Enhanced Shockburst™ protocol which is explained in the nRF24LE1 Product Specification. The goal is to create a reliable communication channel between the host application and the remote device. Enhanced Shockburst™ provides automatic packet handling, up to 32 bytes of payload, auto acknowledgment and retransmission, as well as CRC message integrity verification.

The 2.4 GHz RF range can be susceptible to disturbances from WLAN and other devices operating in that range. It is therefore desirable to be able to operate on more than one channel. The channels in the implementation were arbitrarily chosen, so you may wish to choose your own.

6 References

- nRF24LE1 Product Specification
- nRFgo Software Development Kit
- nAN-22

7 Glossary

Term	Description
Boot loader	A routine executed immediately after system start which is responsible for initiating the other functions of the device
Compiler	A compiler is a computer program (or set of programs) that transforms source code written in a programming language into another computer language, typically to create an executable program.
CRC	Cyclic redundancy check
Firmware	A program or set of instructions programmed on a hardware device. Usually stored in flash memory or EEPROM
Flash	Non-volatile computer storage chip that can be electrically erased and reprogrammed
HAL	Hardware abstraction layer
HEX record	One line in the Intel HEX format, containing the fields specified in Table 1. on page 5
Interrupt function	Function executed after an interrupt event has occurred. The MCU transfers program control to the interrupt function, and lets it execute fully before transferring it back to the previous, executing function.
IRQ	Interrupt request
Linker	A computer program that takes one or more objects generated by a compiler and combines them into a single executable program
MCU	Microcontroller
Reset vector	The first three bytes in flash memory, containing a long jump instruction to the firmware startup segment. The first command that is executed after a system reset
RF	Radio frequency
Segment	One of the sections of a program in flash memory

Liability disclaimer

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

Life support applications

Nordic Semiconductor's 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 Semiconductor ASA customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nordic Semiconductor ASA for any damages resulting from such improper use or sale.

Contact details

For your nearest dealer, please see http://www.nordicsemi.com.

Receive available updates automatically by subscribing to eNews from our homepage or check our website regularly for any available updates.

Main office:

Otto Nielsens veg 12
7004 Trondheim
Phone: +47 72 89 89 00
Fax: +47 72 89 89 89
www.nordicsemi.com

Revision History

Date	Version	Description
September 2011	1.0	