nRF24L01 FIRMWARE DESCRIPTION

INTRODUCTION

This document describes the firmware for the nRF24L01-EVKIT. Contents of the source files and a briefly description of these are written here. Also, the Frequency Agility Protocol (FAP) is described in this document.

FIRMWARE, FILE DESCRIPTION

Below is a list of .c files included in this project.

main.c
Main program, USB commands decoder, nRF_IRQ source decoder (part of the FAP) and Start_Communication (init L01 for RX mode or TX mode).

nRF_API.c:
Basic SPI commands; read and write functions.

Protocol_API.c
High level L01 commands, based on the nRF_API SPI commands.

LL_API.c:
CPU dependent functions: Timers, port pins functions like LED's handling and SPI port pins control.

ISR.c:
Interrupt vectors for timers, external interrupt0 (nRF_IRQ pin) and USB.

ADC.c:
ADC functions for temperature and voltage monitor.

F320_FlashPrimitives.c and fwupgrade.c:
These files contain the functions used for firmware upgrade.

USB.c:
USB specific, and also contains the "USB Descriptor Information".

TestApp.c:
Part of the firmware testapplication

Below is a list of the .h files included in this project.

defines.h
This file contains global, general definitions used in the project.

nRF_API.h
This file contains the nRF24L01 register definitions, and nRF_API.c functions prototypes.

Protocol_API.h
This file contains the Protocol_API.c functions prototypes.

LL_API.h
This file contains the low-level (microcontroller dependent) interface definitions and LL_API.C functions prototypes.

ADC.h
This file contains the ADC definitions and the ADC.c functions prototypes.

F320_FlashPrimitives.h and fwupgrade.h
These files contain definitions for the firmware upgrade functions and the prototypes for these functions.

USB.h
This file contains definitions for the USB ISR function, and prototypes for USB functions.

TestApp.h
This file contains prototypes for the TestApp.c functions.

FIRMWARE DESCRIPTION, FREQUENCY AGILITY PROTOCOL

This chapter gives a briefly description of how this firmware works. This firmware has been written in a way that it can emulate a wireless mouse and keyboard. With this, it means that an operator can initialize two or more devices, one working like a USB dongle, the second working like a wireless mouse and the third working like a keyboard. The Frequency Agility Protocol (FAP) is running on the pipe that emulates the wireless mouse. In addition to this, the keyboard emulation can be set up on a second pipe. Refer to “nRF24L01 TEST SETUP” for detailed description of how to perform this.

The whole idea for the FAP is the ability to change channel in case of disturbance on selected channel. Figure 1 shows the state machine which is the basis of this firmware implementation.

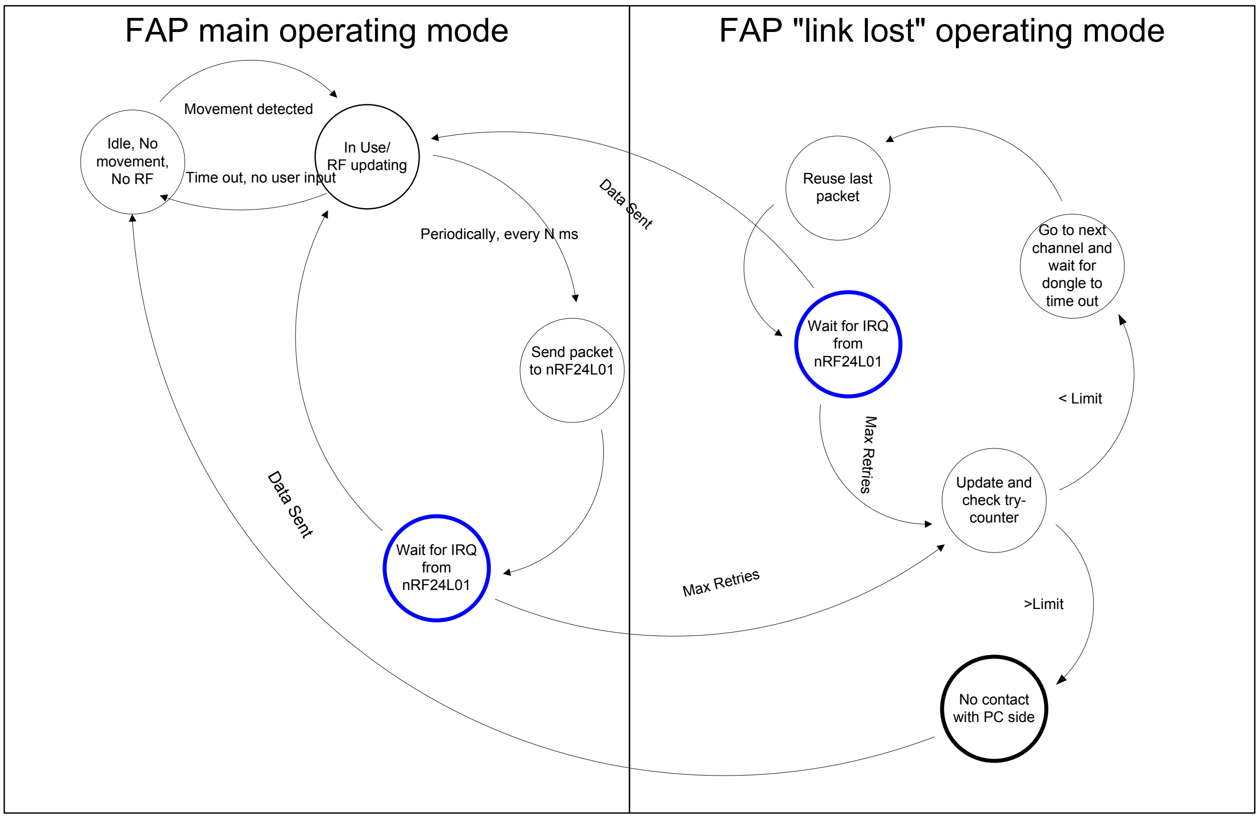

Figure 1: Implementation of the FAP in “active polling” mode

Looking at Figure 1, the description of the state machine flow will follow below.

- "Idle, No movement, No RF"; this is the state where the "active polling device" is doing nothing, and this is the state this device will stay most of the time.
- "In Use/RF updating"; this is the state where the transaction is initiated, i.e. have new updated data to transmit.
- "Send packet to nRF24L01"; new data is loaded into L01, and a transmission is initiated.
- "Wait for IRQ from nRF24L01"; last data packet have been transmitted, so the microcontroller is waiting for IRQ from nRF24L01, either "TX_DS" or "MAX_RT" interrupt.

    ➔ If data was sent successfully ("TX_DS" IRQ), the microcontroller goes back to "In use/RF updating" state, waiting for a new transmission.

    ➔ If data was NOT sent successfully ("MAX_RT" IRQ), the microcontroller goes to "update and check try-counter" state.

- "Updating and check try-counter"; a try-counter is incremented each time the channel table for FAP is wrapped. This counter has a fixed limit of 3, so if this maximum is reached the TX gives up and returns to "Idle, No movement, No RF" state. Note; in this firmware, to make it simple, the try-counter is not implemented for the "active polling" mode, but it is implemented for the "follow" mode.

    ➔ In case of "Try-counter > Limit"; the microcontroller goes to "No contact with PC side", and returns back to "Idle, No movement, No RF".

    ➔ In case of "Try-counter < Limit"; the microcontroller goes to the "Go to next channel and wait for dongle to time out".

- "Go to next channel and wait for dongle to time out"; this is the main part of the FAP. The nRF24L01 got no contact on current channel, so it then change channel according to channel table.

    ➔ If this is the first link loss since last successfully transmission, the nRF24L01 wait for T_{TX} until it tries to resend last packet. The reason for doing this is to give the RX a change to timeout and change channel, before TX tries to retransmit again.

    ➔ If this was not the first link loss since last successfully transmission, the TX change channel and resend its packet immediately, trying to catch up with the RX.

- "Reuse last packet"; this is the state where the nRF24L01 actually retransmit the last packet, with changed channel.
- "Wait for IRQ from nRF24L01"; data packet have been resent, and the microcontroller is waiting for an interrupt from nRF24L01, either "TX_DS" or "MAX_RT".

    ➔ In case of a "TX_DS" interrupt, the data was sent successfully and the microcontroller goes to the "In use/ RF updating" state, and are ready for new a transmission.

    ➔ In case of a "MAX_RT" interrupt, the microcontroller goes to back to the "Update and check try-counter", getting ready to retransmit last sent packet.

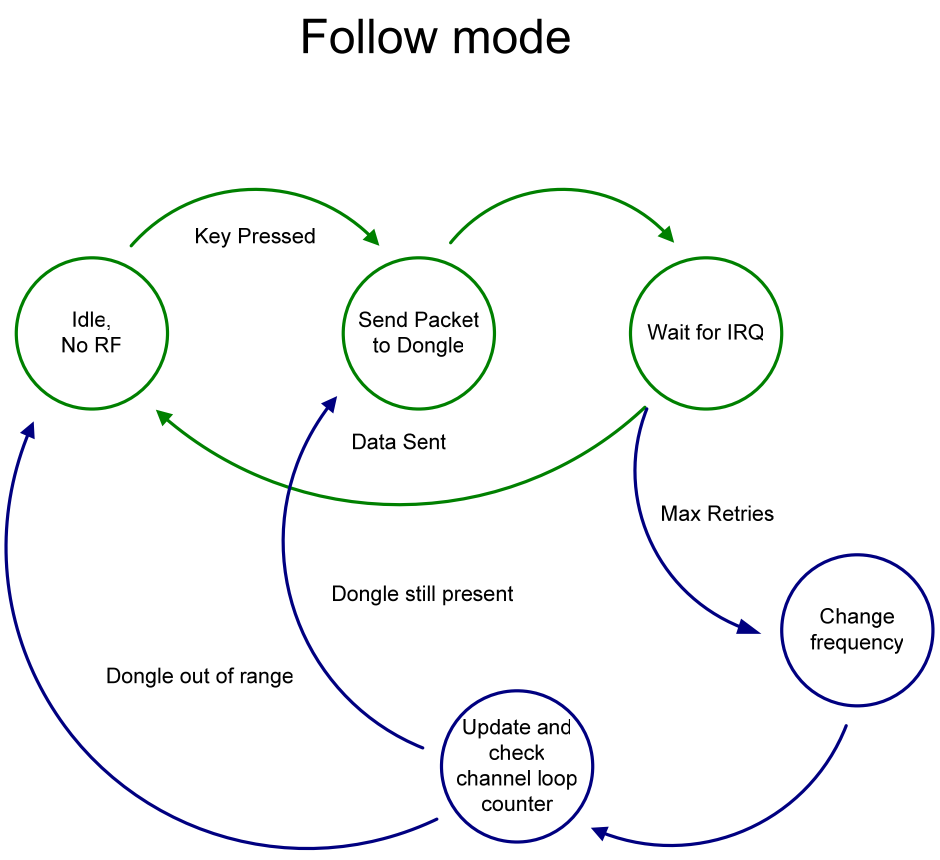

Figure 2: Frequency Agility Protocol in "follow" mode

Figure 2 shows the implementation of the frequency agility for "follow" mode. For this firmware, this is reflected to the TX using Button 1 to send data packets.

- "Idle, No RF"; the board is waiting for a button press.
- "Send Packet to Dongle"; Button 1 have been pressed, so the microcontroller loads the nRF24L01 with the data payload, and initializes a data transmission i.e. tries to send the packet.
- "Wait for IRQ"; the data packet have been transmitted, so the microcontroller waits for interrupt from nRF24L01, either "TX_DS" or "MAX_RT".
  ➔ If data was sent successfully, the microcontroller goes back to "Idle, No RF" state and is ready for a new transmission.
  ➔ If data was NOT sent successfully, the microcontroller goes to the "Change frequency" state.

- "Change frequency"; last packet was lost, so the nRF24L01 changes channel, according to the channel table, and goes to the "Update and check channel loop counter".

- "Update and check channel loop counter"; this state verifies that the try-counter is less that 3, i.e. verifies that the frequency table have wrapped less that 3 times.

  ➔ If try-counter = 3, the microcontroller goes to the "Idle, No RF" assuming the dongle is out of range.

  ➔ If try-counter < 3, the microcontroller goes back to the "Send packet to Dongle", and retry to transmit the data packet.

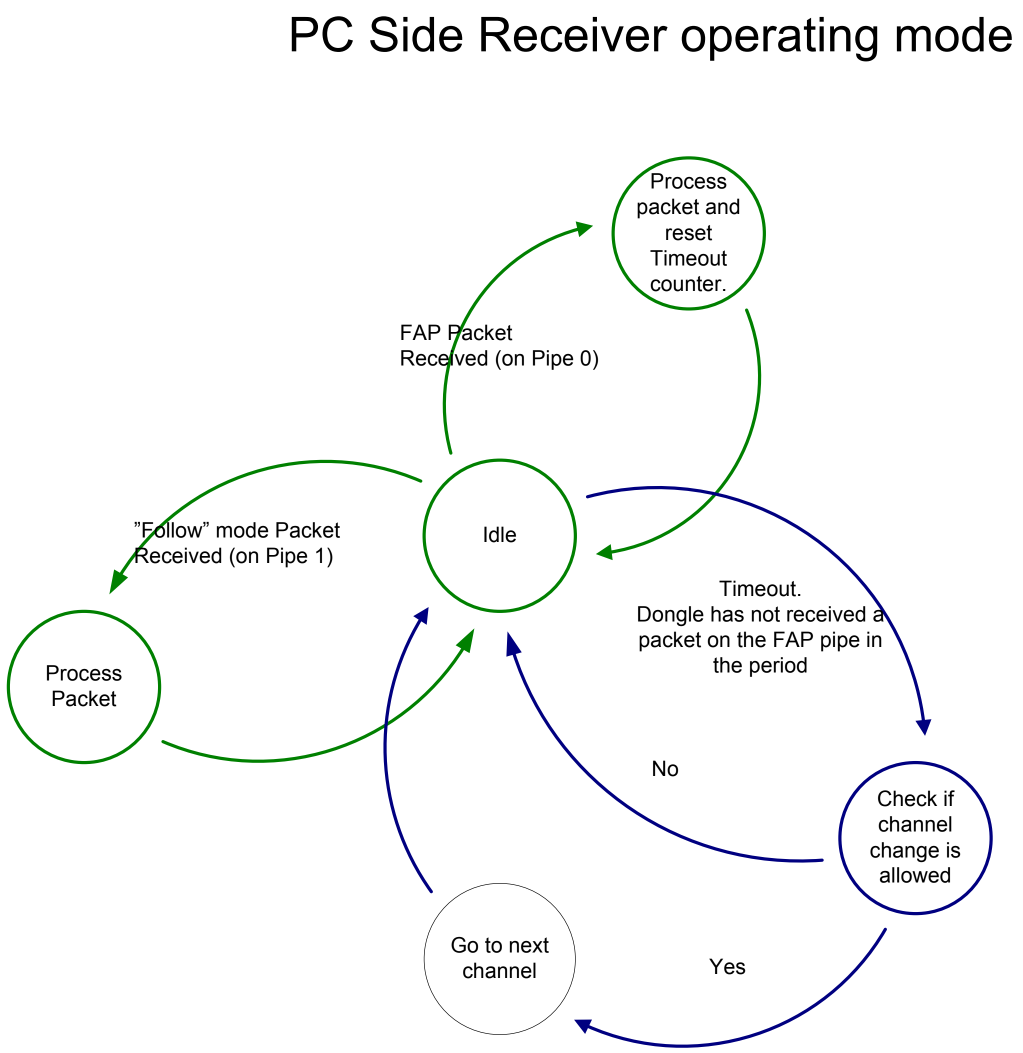
PC Side Receiver operating mode
Process packet and reset Timeout counter.
FAP Packet Received (on Pipe 0)
"Follow" mode Packet Received (on Pipe 1)
Idle
Process Packet
Timeout. Dongle has not received a packet on the FAP pipe in the period
No
Check if channel change is allowed
Go to next channel
Yes

Figure 3: Implementation of the FAP at the RX side

Figure 3 shows the implementation of the frequency agility for the RX device. This implementation accepts data on multiple pipes, but for only one pipe in "active" FAP mode. This means that if a FAP "active polling" mode is set for pipe 0, the pipe 1 can be set up for "follow" mode and so on.

- “Idle”; in this state the microcontroller is waiting for either an “RX_DS” interrupt from the nRF24L01, or a timeout. That is if the FAP “active poling” mode is activated.

  ➔ If a “RX_DS” interrupt occur, the microcontroller check if data was received on the “active polling” pipe. If so, the microcontroller processes this data, i.e. read and store this RX payload and reset the Timeout counter. If this data was received on a “follow” mode pipe, this data is processed as well, but the Timeout counter is not reset.

  ➔If a Timeout occur, the microcontroller goes to the “Check if channel change is allowed” state.

- “Check if channel change is allowed”; this state verifies that channel change is allowed before further processing. In this firmware, this state will always return “YES”, sending the microcontroller to the “Go to next channel” state.

  ➔If “NO”, the microcontroller goes back to the “Idle” state and reset the Timeout counter, waiting for new data reception.

  ➔If “YES”, the microcontroller goes to the “Go to next channel” state.

- “Go to next channel”; this state do exactly is it says, it changes channel according to the channel table, and returns to the “Idle” state, waiting for new data receptions.

This firmware has been compiled with the Keil C51 V7.50.
It uses the “SiLabs USBXpress Development Kit”, v.2.3 as the USB driver package.

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 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 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.

YOUR NOTES

Nordic Semiconductor - World Wide Distributor

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

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

Visit the Nordic Semiconductor ASA website at http://www.nordicsemi.no