nRF24L01 Library Documentation

Revision 1.1

Luka Gacnik

October 2023

Table of contents

Table of figures

Acronyms

MCU	Microcontroller Unit
PIO	Parallel Input Output
SPI	Serial Peripheral Interface
RF	Radio Frequency
PA	Power Amplifier
LNA	Low Noise Amplifier
PRX	Primary receiver
PTX	Primary transmitter
ISR	Interrupt Service Routine
ACK	Acknowledgement
RX	Receiver
TX	Transmitter
RX FIFO	Receiver First In First Out buffer
TX FIFO	Transmitter First In First Out buffer
SFR	Special Function Register

1 Library description

The library consists of basic functions intended for operating the nRF24L01 – a 2.4 GHz wireless transceiver with an embedded baseband protocol engine (Enhanced Shock-Burst), designed for ultra-low power wireless applications.

1.1 About the nRF24L01

The nRF24L01 transceiver is a stand-alone integrated circuit that can be controlled an MCU via SPI communication protocol. The latter is used to modify or read user-accessible configuration registers. The main features of the nRF24L01 device are the following ones:

- Can operate in the frequency range of 2.400 - 2.4835 GHz (126 of available RF channels)
- Up to 2 Mbps air data rate (increased sensitivity at lower rates)
- Programmable output power up to 0 dBm
- Offers Enhanced ShockBurst data link layer that is capable of automatic packet assembly and timing, automatic acknowledgement and retransmissions of packets
- A MultiCeiver feature where one receiver can handle data reception from up to six different transmitters
- Two stand-by modes for ultra-low power consumption
- SPI speed of max 10 Mbps data rate

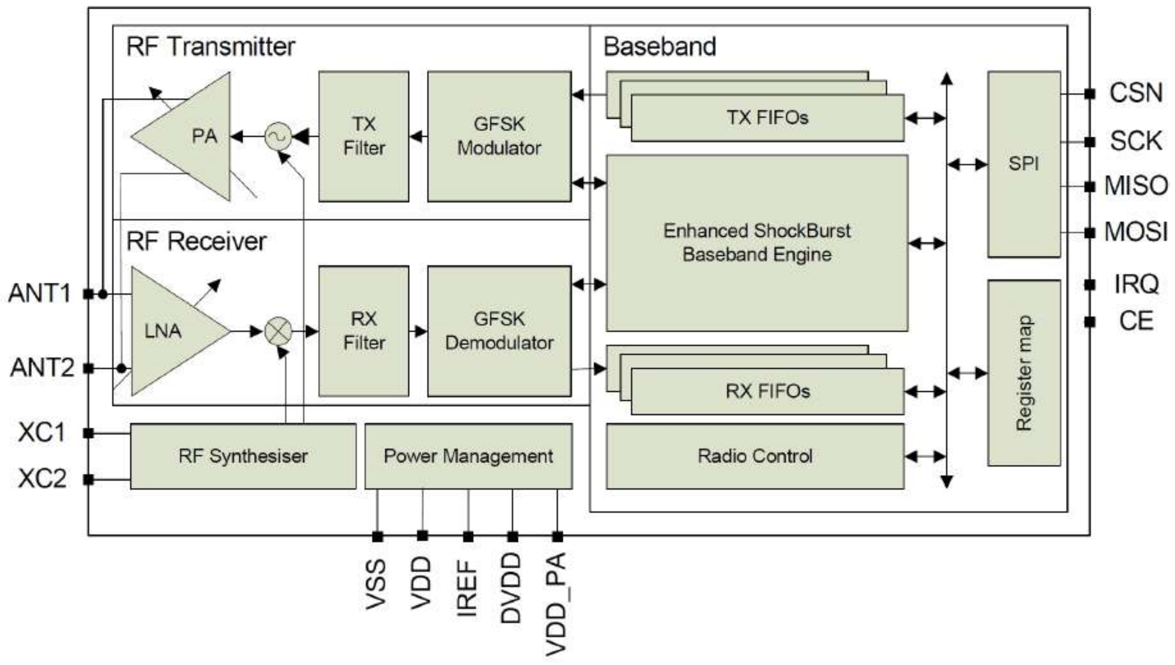

Figure 1.1: nRF24L01+ block diagram

For the sake of convenience a printed circuit board with the nRF24L01 integrated circuit and required external components was used throughout the library development phase. The MCU interfaces the nRF24L01 device using the following pins:

- **GND** - ground
- **VCC** - 3.3 V power supply
- **CE** - active-high chip enable for enabling transmit/receive mode
- **CSN** - active-low chip select for SPI accessing of the device
- **SCK** - SPI clock pulse
- **MOSI** - SPI input for the nRF24L01
- **MISO** - SPI output for the nRF24L01
- **IRQ** - interrupt pin for event handling

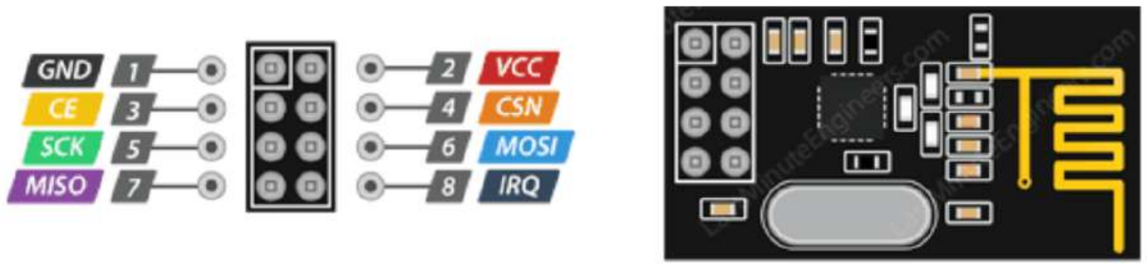

Figure 1.2: nRF24L01+ pin diagram

Such a circuit is said to have a range of 100 m in empty space. The range can be improved using nRF24L01+ PA/LNA circuit, which comes with an input low-noise amplifier and output power amplifier. It offers an improved range of up to 1000 m line of sight. The pinout of should be the same for both variants.

1.2 Library features

Currently, the nRF24L01 driver is capable of performing the following operations:

- Modifying both receiver (PRX) and transmitter (PTX) configuration registers for a user-defined operation, which is set via configuration structure
- Performing either polling-based or interrupt-based send and receive data operations for the PTX
- Initiating interrupt-based reception for the PRX (with optional acknowledgement payload respond)

2 Dependencies

The library consists of the following source and header files:

- `nRF24L01.c`
- `nRF24L01.h`
- `nRF24L01_sfr.h`

All of the external user-accessible function prototypes are provided in the `nRF24L01.h` file. Their definitions and other local functions like ISR handlers are contained in the `nRF24L01.c` file. The file `nRF24L01_sfr.h` contains all the macro definitions for register bit-fields and SPI commands.

The library depends on the following other libraries:

- `Spi.h`
- `Tmr.h`

The library `Spi.h` contains the SPI functions needed for interfacing the nRF24L01 from the MCU. The library `Tmr.h` is needed for the timeout feature of the nRF24L01 library. The rest of libraries which include other peripheral's functionalities are contained inside the `Spi.h` library.

Additionally, there needs to be one of the standard Microchip libraries included (since the compiler XC32 is the made by the Microchip), that is the `sys\attrib.h` library. It is used for its ISR macro definition that is needed for ISR function attribute definition.

Lastly, its important to provide the fact that this library was meant to be used with a specific MCU - the PIC32MX170Bxxx series made by the Microchip. As previously mentioned the XC32 compiler was used to compile the library files, meaning there are some specifics in the library which are unique to the XC32 compiler (such as ISR function attribute). In case any other MCU of the PIC32 family would be used, some peripheral libraries (used by this driver) might need modifications (like different register sets, etc.).

3 Notes and warnings

There are few constrains which limit the capability of the nRF24L01 library:

a) **Single device support only**

   The library doesn't support multiple nRF24L01 devices to be controlled by the same library on the same MCU. Doing so would require much more complex operation which is in fact not needed in most cases (from author's perspective). nRF24L01 can transmit data to any number of receivers, while one receiver can perform a reception for up to six different transmitters. One of the cases where multiple transmitters could be used is when transmit to many receivers would be required and would need to be parallelized for a faster performance of the system.

   One could bypass the limitation explained above by copying the same library under a different name and rename all the external functions – some other code would need to be renamed (such as `INTx_ISR_MACRO`) or deleted to prevent re-definition. In addition, different interrupt vector INTx would need to be used for this other nRF24L01 device. Finally, the amount of additional nRF24L01 devices used would be the same as the number of different copies of the same nRF24L01 library.

In addition, some some code related facts must be known prior to using the nRF24L01 library:

a) **Definition of an appropriate ISR vector**

   The header file `nRF24L01.h` contains a macro definition `INTx_ISR_MACRO` (where `x` can be in the range of `0` to `4`) which is used for the purpose of defining an appropriate interrupt-based operation parameters. These parameters are the following ones: number the of external interrupt vector used (in regards to the interrupt vector table of an MCU), interrupt flag and enable mask. Additionally, the (sub)priority level (referring to macros: `NRF_ISR_IPL`, `NRF_ICX_IPL` and `NRF_ICX_ISL`) of external interrupt vector need defined as well.

4 Future ideas

Current ideas for future development of the nRF24L01 library are the following ones:

- Providing function(s) for entering/exiting either of stand-by modes for ultra-low power consumption
- Re-formatting the library so that it would support multiple nRF24L01 device usage (this might not necessarily be needed for a casual transmission but only in case of communication with a huge amount of remote receiver devices where transmission parallelization would be desired)

5 Custom types

The library comes with a set of custom made structure or enumeration types. Since these are mentioned often they will be elaborated in depth in the following sections. Please note that the custom type `SpiSfr_t *` is not explained here since it is a part of the SPI library.

5.1 NrfPtxConfig_t

This structure-based type contains the properties for configuration of the PTX mode operation. Structure of such a type is intended to be passed into the `NRF_ConfigPtxSfr()` function. The type consists of the following properties:

- **`spiSfr`** (*`SpiSfr_t *`*) - the SPI module base address which controls the SPI bus to which the nRF24L01 is connected to
- **`isAck`** (*`NrfIsAck_t`*) - packet acknowledgement enabled/disabled
- **`retrDelay`** (*`NrfRetransmitDelay_t`*) - the amount of time the PTX waits for the response from the PRX (only applicable when ACK enabled)
- **`retrCount`** (*`NrfRetransmitCount_t`*) - the number of times the PTX needs to retransmit the packet before asserting `MAX_RT` (link lost) (only applicable when ACK enabled)
- **`rfChannel`** (*`NrfRfChannel_t`*) - the specific frequency channel used for the packet transmission
- **`rfPower`** (*`NrfRfPower_t`*) - the transmission power of the nRF24L01
- **`dataRate`** (*`NrfDataRate_t`*) - the data rate of the packet transmission
- **`pinConfig`** (*`NrfPinConfig_t`*) - the non-SPI pins used for controlling the nRF24L01 device

5.2 NrfPrxConfig_t

This structure-based type contains the properties for configuration of the PRX mode operation. Structure of such a type is intended to be passed into the `NRF_ConfigPrxSfr()` function. The type consists of the following properties:

- **`spiSfr`** (*`SpiSfr_t`* *) - the SPI module base address which controls the SPI bus to which the nRF24L01 is connected to
- **`isAck`** (*`NrfIsAck_t`*) - packet acknowledgement enabled/disabled
- **`rfChannel`** (*`NrfRfChannel_t`*) - the specific frequency channel used for the packet reception
- **`dataRate`** (*`NrfDataRate_t`*) - the data rate of the packet reception
- **`pipeAddr`** (*`NrfPipeAddr_t`*) - the addresses (up to 6) which identify the remote PTX device
- **`pinConfig`** (*`NrfPinConfig_t`*) - the non-SPI pins used for controlling the nRF24L01 device

5.3 NrfPayloadConfig_t

This structure-based type contains the properties for accessing the nRF24L01 device. It is intended to be passed into reception/transmission functions. The type consists of the following properties:

- **`spiSfr`** (*`SpiSfr_t`* *) - the SPI module base address which controls the SPI bus to which nRF24L01 is connected
- **`pipeAddr`** (*`uint64_t`*) - the unique address of a specific PTX which is recognized by the PRX prior to prior to packet reception
- **`pinConfig`** (*`NrfPinConfig_t`*) - the non-SPI pins used for controlling the nRF24L01 device

Note that the property `pipeAddr` is only used when calling PTX related functions. It is ignored in the PRX related operations. Do not confuse the this property and its type with the property `pipeAddr` of a type *`NrfPipeAddr_t`* which is used for the configuration of the PRX mode.

5.4 NrfPipeAddr_t

This structure-based type contains the properties that inform the nRF24L01 device, operating in the PRX mode, about the unique PTX addresses through which the PRX recognizes the valid transmitter. Total of 6 pipe RX addresses are available for configuration of PRX mode. Each property of this type is the *`uint64_t`* type where all the addresses can be configured to a maximum of 5-byte long unique identifier.

5.5 NrfPinConfig_t

This structure-based type contains the properties used for recognition of the non-SPI pins that are used for interfacing a specific nRF24L01 on the SPI bus. The type consists of the following properties:

- **`cePin`** (*`uint32_t`*) - the chip enable pin which is used for triggering the nRF24L01 device into the reception/transmission
- **`csPin`** (*`uint32_t`*) - the chip select pin which is used for means of synchronisation between the MCU and the nRF24L01 device
- **`irqPin`** (*`uint32_t`*) - the pin for detecting an external event from the nRF24L01 device which might trigger the ISR on the MCU

6 Function description

The library includes the following functions:

- `NRF_ConfigPtxSfr()`
- `NRF_ConfigPrxSfr()`
- `NRF_ConfigPtxPayloadStruct()`
- `NRF_ConfigPrxPayloadStruct()`
- `NRF_SendReceivePayload()`
- `NRF_SendPayload()`
- `NRF_StoreAckPayload()`
- `NRF_StartReception()`
- `NRF_StopReception()`
- `NRF_FlushRxFifo()`
- `NRF_IsRxFifoLoading()`
- `NRF_ReadPrxPipeAddr()`
- `NRF_ReadStatus()`
- `NRF_SetUserCallback()`
- `NRF_ReleaseUserCallback()`

The following sub-sections describe the use and operationality of these functions.

6.1 NRF_ConfigPtxSfr()

PROTOTYPE

`NRF_ConfigPtxSfr(ptxConfig)`

DESCRIPTION

Configures non-SPI pins and modifies the nRF24L01 internal registers for the PTX (transmission) operation.

PARAMETERS

- **`ptxConfig`** (*`NrfPtxConfig_t`*) - contains the settings which determine the operation parameters of the nRF24L01 device being addressed

RETURNS

Returns `false` if no response from the nRF24L01 device was received during the power-up sequence. Otherwise, it returns `true`.

OPERATION

First, the necessary PIO registers are written to for the CE and IRQ pins (CS modified within the scope of the SPI module configuration). Then the power-up sequence of 1.5 ms period length is executed. Afterwards, a series of SPI non-ISR writes are executed to modify the nRF24L01 internal user-accessible registers. Lastly, the interrupt SFRs are configured.

NOTES

- The selection of an external interrupt vector `INTx_ISR_MACRO` macro needs to be set in the `nRF24L01.h` file prior to using any of the nRF24L01 library functions. In addition, the (sub)priority level (referring to macros: `NRF_ISR_IPL`, `NRF_ICX_IPL` and `NRF_ICX_ISL`) of external interrupt vector need to be set as well.

- The SPI module which was provided in the `ptxConfig` needs to be configured and operational prior to calling any of nRF24L01 functions that use the SPI bus (the

SPI clock speed must never exceed 10 Mbps).

- The IRQ pin used for detection of nRF24L01 events is set to have a weak internal pull-up resistor connected to it. If this pin is used for other intentions, make sure to disable the pull-up resistor if not required.

LIMITATIONS

None.

6.2 NRF_ConfigPrxSfr()

PROTOTYPE

`NRF_ConfigPrxSfr(prxConfig)`

DESCRIPTION

Configures non-SPI pins and modifies the nRF24L01 internal registers for the PRX (reception) operation.

PARAMETERS

- **`prxConfig`** (*`NrfPrxConfig_t`*) - contains the settings which determine the operation parameters of the nRF24L01 device being addressed

RETURNS

Returns `false` if no response from the nRF24L01 device was received during the power-up sequence. Otherwise, it returns `true`.

OPERATION

First, the necessary PIO registers are written to for the CE and IRQ pins (CS modified within the scope of the SPI module configuration). Then the power-up sequence of 1.5 ms period length is executed. Afterwards, a series of SPI non-ISR writes are executed to modify the nRF24L01 internal user-accessible registers. Lastly, the interrupt SFRs are configured.

NOTES

- The selection of an external interrupt vector `INTx_ISR_MACRO` macro needs to be set in the `nRF24L01.h` file prior to using any of the nRF24L01 library functions. In addition, the (sub)priority level (referring to macros: `NRF_ISR_IPL`, `NRF_ICX_IPL` and `NRF_ICX_ISL`) of external interrupt vector need to be set as well.

- The SPI module which was provided in the `prxConfig` needs to be configured and operational prior to calling any of nRF24L01 functions that use the SPI bus (the

SPI clock speed must never exceed 10 Mbps).

- The IRQ pin used for detection of nRF24L01 events is set to have a weak internal pull-up resistor connected to it. If this pin is used for other intentions, make sure to disable the pull-up resistor if not required.
- The pipe address register has always 5 bytes available for an individual address. If either of the `PIPE_2` to the `PIPE_5` is used for the reception, then its 4 uppermost bytes will always be copied from the `PIPE_1` by the nRF24L01 device. If the `PIPE_1` is not configured (equals 0), the other higher pipes will have 4 uppermost bytes configured to all-zeros (not recommended).
- The address all-zeros should never be used (`NRF_ConfigPrxSfr()` will detect this as the pipe not configured) and all-ones is not recommended by the nRF24L01 device documentation.

LIMITATIONS

None.

6.3 NRF_ConfigPtxPayloadStruct()

PROTOTYPE

`NRF_ConfigPtxPayloadStruct(ptxConfig, pipeAddr)`

DESCRIPTION

Copies specific members from the configuration structure of a type *`NrfPtxConfig_t`* to a payload structure of a type *`NrfPayloadConfig_t`*.

PARAMETERS

- **`ptxConfig`** (*`NrfPtxConfig_t`*) - contains the settings which determine the operation parameters of the nRF24L01 device being addressed
- **`pipeAddr`** (*`uint64_t`*) - the address which PRX uses to identify a specific PTX device

RETURNS

Returns the payload structure of a type *`NrfPayloadConfig_t`*.

OPERATION

Specific members from one structure are copied to another.

NOTES

None.

LIMITATIONS

None.

6.4 NRF_ConfigPrxPayloadStruct()

PROTOTYPE

```
NRF_ConfigPrxPayloadStruct(prxConfig)
```

DESCRIPTION

Copies specific members from the configuration structure of a type *`NrfPrxConfig_t`* to a payload structure of a type *`NrfPayloadConfig_t`*.

PARAMETERS

- **`prxConfig`** (*`NrfPtxConfig_t`*) - contains the settings which determine the operation parameters of the nRF24L01 device being addressed

RETURNS

Returns the payload structure of a type *`NrfPayloadConfig_t`*.

OPERATION

Specific members from one structure are copied to another.

NOTES

None.

LIMITATIONS

None.

6.5 NRF_SendReceivePayload()

PROTOTYPE

```
NRF_SendReceivePayload(payldConfig, rxPtr, txPtr, txSize)
```

DESCRIPTION

Sends a packet of data to the remote PRX with a possible reception of ACK data from PRX. The operation is polling-based; therefore, the status of operation is known as the function execution concludes.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address

- **`rxPtr`** (*`void` **) - a memory address of the MCU where the ACK response data (if any) will be stored at

- **`txPtr`** (*`void` **) - a memory address of the MCU where the packet data to be transmitted to the remote PRX is stored at

- **`txSize`** (*`uint32_t`*) - the size (in bytes) of a packet to be transmitted

RETURNS

Returns `true` if a payload was successfully transmitted (with or without ACK enabled). Otherwise (link lost or unresponsive device) returns `false`.

OPERATION

Initially, TX and RX FIFO are flushed to ensure a straightforward operation. Then the PTX address, and packet data are written to the nRF24L01 device. Afterwards, the PTX transmission starts and the MCU is polling the IRQ pin. After some time, the response is received from the PTX, resulting in either a successful or a failed transmission. A timeout is also provided (by the core timer) to prevent the possibility of MCU polling for an event of an unresponsive device.

NOTES

- The width of packet data loaded to the nRF24L01 device is always an 8-bit width
- The PTX identification address and a specific pipe address on the remote PRX side must be the same
- If the size of a receiving ACK payload (from the remote PRX) is unknown, it is best to reserve a space of exactly 32 bytes at the memory address of the `rxPtr` pointer. Otherwise, the MCU might generate an exception for writing the data into inaccessible memory space

LIMITATIONS

The maximum length of a packet, determined by the `txSize` parameter, is always 32 bytes (the same applies for the ACK packet from the PRX).

6.6 NRF_SendPayload()

PROTOTYPE

`NRF_SendPayload(payldConfig, rxPtr, txPtr, txSize)`

DESCRIPTION

Sends a packet of data to the remote PRX with a possible reception of ACK data from PRX. The operation is interrupt-based, therefore the current operation status can be read by calling the `NRF_ReadStatus()` function.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address
- **`rxPtr`** (*`void`* *) - a memory address of the MCU where the ACK response data (if any) will be stored at
- **`txPtr`** (*`void`* *) - a memory address of the MCU where the packet data to be transmitted to the remote PRX is stored at
- **`txSize`** (*`uint32_t`*) - the size (in bytes) of a packet to be transmitted

RETURNS

Always returns `true`.

OPERATION

Initially, TX and RX FIFO are flushed to ensure a straightforward operation. Then the PTX address, and packet data are written to the nRF24L01 device – here the function concludes, and SPI ISR takes over handling the write session. At the end of SPI write the function `ISR_NrfHandler_StartTransmission()` is called – here the PTX transmission starts, and timeout timer starts counting. After some time, the response is received from the PTX and the action is provided by the `ISR_NrfHandler_SendPayload()` ISR, where ACK payload might be read and status updated. Operation status can be verified at any time by the function `NRF_ReadStatus()`.

In the case of an unresponsive device or SPI bus malfunction, the timeout is triggered. The function `ISR_NrfTimeoutHandler_SendPayload()` handles this case and is by default set to trigger after 30 ms (by the local variable `timeoutVal`). This period might need to be extended if some other interrupt handler of a higher priority could possibly delay the `ISR_NrfHandler_SendPayload()` execution.

NOTES

- The width of packet data loaded to the nRF24L01 device is always an 8-bit width
- The PTX identification address and a specific pipe address on the remote PRX side must be the same
- If the size of a receiving ACK payload (from the remote PRX) is unknown, it is best to reserve a space of exactly 32 bytes at the memory address of the `rxPtr` pointer. Otherwise, the MCU might generate an exception for writing the data into inaccessible memory space

LIMITATIONS

The maximum length of a packet, determined by the `txSize` parameter, is always 32 bytes (the same applies for the ACK packet from the PRX).

6.7 NRF_StoreAckPayload()

PROTOTYPE

`NRF_StoreAckPayload(payldConfig, pipeNo, txPtr, txSize)`

DESCRIPTION

Stores an ACK payload into the TX FIFO of a PRX device for a specific pipe number.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address
- **`pipeNo`** (*`NrfRxPipeNo_t`*) - the number of a pipe whose address matches the PTX device where the ACK payload will be sent to
- **`txPtr`** (*`void *`*) - a memory address of the MCU where the ACK payload data is stored at
- **`txSize`** (*`uint32_t`*) - the size (in bytes) of an ACK payload

RETURNS

Always returns `true`.

OPERATION

The PRX reception is temporarily disabled, if it was previously active, until ACK payload is fully loaded into a nRF24L01 device. Reception is enabled in the ISR by the `ISR_NrfHandler_RestartReception()` function after the last SPI write is carried out. If the PRX reception wasn't active yet, when this function was called, then one shall not call the function `NRF_StartReception()` until the function `NRF_IsRxFifoLoading()` returns `false`. The latter is used in the former function as an additional protection (it is not known what would happen if the ACK payload would be partially loaded as the PTX would establish link and the PRX would need to respond). However, if the main program shall do other tasks while the ACK payload is being loaded, then the function `NRF_IsRxFifoLoading()` shall be used.

NOTES

- Up to three ACK payloads can be stored in the nRF24L01 device. These payloads can correspond to different pipe numbers (PTX addresses) or the same ones. In latter case, the selection of an ACK payload for the current transaction is determined in a FIFO manner
- Use the function `NRF_IsRxFifoLoading()` to check whether the RX FIFO is loaded by the ACK payload and the reception can start
- The TX FIFO of the PRX device can be flushed using `NRF_FlushRxFifo()`

LIMITATIONS

The maximum length of a packet, determined by the `txSize` parameter, is always 32 bytes.

6.8 NRF_StartReception()

PROTOTYPE

`NRF_StartReception(payldConfig, rxPtr)`

DESCRIPTION

Starts the reception for the PRX device. The operation is interrupt-based, therefore the current operation status can be read by calling the `NRF_ReadStatus()` function.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address
- **`txPtr`** (*`void`* *) - a memory address of the MCU where the packet data from the remote PTX will be stored at

RETURNS

Always returns `true`.

OPERATION

Initially, RX FIFO is flushed to ensure a straightforward operation. Then the device status is cleared so that normal operation can continue, and the reception starts.

When the PRX activates the IRQ line the ISR is triggered and the function `ISR_NrfHandler_ReadPayload()` executes. If there is data in the RX FIFO, the ISR-based SPI read begins and the function ends here. At the end of SPI operation the function `ISR_NrfHandler_ReadPayloadCont()` is called (the SPI handler), where the data is written at the `rxPtr` address and the reception automatically restarts. Operation status can be verified at any time by the function `NRF_ReadStatus()`.

NOTES

- If the size of a receiving payload (from the remote PTX) is unknown, it is best to reserve a space of exactly 32 bytes at the memory address of the `rxPtr` pointer. Otherwise, the MCU might generate an exception for writing the data into inaccessible memory space
- If the PRX operation needs to be reconfigured make sure that the function `NRF_StopReception()` is called prior to any configuration activity

LIMITATIONS

None.

6.9 NRF_StopReception()

PROTOTYPE

`NRF_StopReception(payldConfig)`

DESCRIPTION

Stops the reception for the PRX mode.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address

RETURNS

Always returns `true`.

OPERATION

Interrupt sources are disabled, and the device status is cleared.

NOTES

If the reception operation needs to be restarted, make sure that the function `NRF_StartReception()` is called once.

LIMITATIONS

None.

6.10 NRF_FlushRxFifo()

PROTOTYPE

`NRF_FlushRxFifo(payldConfig)`

DESCRIPTION

Empties all three levels of the RX FIFO. Intended for the PRX usage to flush previous ACK payloads.

PARAMETERS

- **`payldConfig`** (*`NrfPayloadConfig_t`*) - contains the information which determines a specific nRF24L01 device the MCU will be communicating with and the PTX identification address

RETURNS

None.

OPERATION

SPI write of a single byte is executed.

NOTES

None.

LIMITATIONS

None.

6.11 NRF_IsRxFifoLoading()

PROTOTYPE

`NRF_IsRxFifoLoading(void)`

DESCRIPTION

Checks whether the ACK payload was fully loaded into the RX FIFO. Intended for the PRX mode usage.

PARAMETERS

None.

RETURNS

Returns `true` if the operation of the ACK payload loading is in progress, otherwise `false`.

OPERATION

Local flag is read.

NOTES

At the beginning of the function `NRF_StartReception()` a flag is polled. However, if other tasks shall be carried out by the main program, then the function `NRF_StartReception()` shall be used afterwards the function `NRF_StartReception()` returns `false`.

LIMITATIONS

None.

6.12 NRF_ReadPrxPipeAddr()

PROTOTYPE

`NRF_ReadPrxPipeAddr(void)`

DESCRIPTION

Reads the pipe address of the latest packet received. Intended for the PRX mode usage.

PARAMETERS

None.

RETURNS

Returns a 5-byte address of the *`uint64_t`* type.

OPERATION

SPI write of a single byte is executed.

NOTES

This function is redundant for the PTX mode and therefore shall not be used because the transmission and the ACK payload reception is always processed via `PIPE_0`.

LIMITATIONS

None.

6.13 NRF_ReadStatus()

PROTOTYPE

`NRF_ReadStatus(void)`

DESCRIPTION

Returns the status of the latest nRF24L01 operation carried out in the PRX or the PTX mode.

PARAMETERS

None.

RETURNS

Returns the status of the type `NrfStatusFlag_t` where one of the provided values by the list below can take place. The flag is reset after the function call has been executed.

The possible returns for the PTX mode:

- `NRF_FLAG_NO_STATUS` (no status) – there was no activity for the nRF24L01 device, or the processing of an event is currently in progress
- `NRF_FLAG_TX_DS` (TX data send) – a successful data transmission has been carried out whether configured for ACK or not
- `NRF_FLAG_ACK_PLD` (ACK payload) – a successful data transmission with ACK enabled has been carried out and the data has been read
- `NRF_FLAG_MAX_RT` (maximum retransmission) – a link with the remote PRX couldn't be established
- `NRF_FLAG_NO_RP` (no response) – the nRF24L01 device is unresponsive

The possible returns for the PRX mode:

- `NRF_FLAG_NO_STATUS` (no status) – there was no activity for the nRF24L01 device, or the processing of an event is currently in progress

- `NRF_FLAG_RX_DR` (RX data receive) – a successful data reception has been carried out and the data has been read
- `NRF_FLAG_ACK_PLD` (ACK payload) - a successful data reception has been carried out and the data has been read. The last ACK payload sent to the remote PTX was successfully received by it
- `NRF_FLAG_NO_RP` (no response) – the nRF24L01 device is unresponsive

OPERATION

Local flag is read and reset to the state `NRF_FLAG_NO_STATUS`.

NOTES

None.

LIMITATIONS

None.

6.14 NRF_SetUserCallback

PROTOTYPE

`NRF_SetUserCallback(cType, fPtr)`

DESCRIPTION

Set a user callback for a certain type of operation.

PARAMETERS

- **`cType`** (*`NrfUserCallback_t`*) - an enum type that determines which callback will be set
- **`fPtr`** (*`void (* ) (void)`*) - a function pointer to a user-defined callback

RETURNS

None.

OPERATION

One of the following callbacks may be set per function call:

- `NRF_CLBK_RX_PAYLOAD_RECEIVE` - callback after PRX payload receive operation
- `NRF_CLBK_TX_ACK_PAYLOAD_RECEIVE` - callback after PTX ACK payload receive from PRX operation
- `NRF_CLBK_TX_START` - callback after PTX start transmission operation
- `NRF_CLBK_TX_TIMEOUT` - callback after PTX send payload timeout operation

NOTES

If the `NULL` is passed as `fPtr` argument then the corresponding function pointer will be released.

LIMITATIONS

None.

6.15 NRF_ReleaseUserCallback

PROTOTYPE

`NRF_ReleaseUserCallback(cType)`

DESCRIPTION

Release a user callback for a certain type of operation.

PARAMETERS

`cType` (*`NrfUserCallback_t`*) - an enum type that determines which callback will be set.

RETURNS

None.

OPERATION

One of the following callbacks may be release per function call:

- `NRF_CLBK_RX_PAYLOAD_RECEIVE` - callback after PRX payload receive operation
- `NRF_CLBK_TX_ACK_PAYLOAD_RECEIVE` - callback after PTX ACK payload receive from PRX operation
- `NRF_CLBK_TX_START` - callback after PTX start transmission operation
- `NRF_CLBK_TX_TIMEOUT` - callback after PTX send payload timeout operation

NOTES

None

LIMITATIONS

None.

7 Examples

A set of examples is provided to demonstrate a simple operation of the nRF24L01 device. Examples provide only the main operation related code. Other processes such as the start-up of an MCU, programming of the device configuration register, setting up other peripheral modules, and similar is not provided here.

7.1 One packet reception

This example provides the code for setting up the nRF24L01 device as a receiver (the PRX mode). Initially, an SPI module is configured - its settings are contained within the `spiPrxConfig` structure which is used to modify the SFRs of the `SPI_2` module of the PIC32MX170B256 microcontroller. Also, the `prxConfig` structure is defined with the required setting and is later passed into appropriate function. The ACK payload is loaded once prior to the start of reception. Afterwards, an MCU waits for an external event from the nRF24L01 device. That is when a payload arrives and the rest is handled via ISR.

The main program can periodically check the status of the latest operation (if any) using the function `NRF_ReadStatus()` and respond appropriately. If further operation in the PRX mode is desired, a new ACK payload can be loaded right after the packet data from the remote PTX was stored at the address of the `prxRxData` array.

A time diagram of this operation's measurement is provided and explained in the appendix (A).

```c
/** Custom libs **/
#include "nRF24L01.h"

/** Macros **/
#define PRX_CS          GPIO_RPB4
#define PRX_CE          GPIO_RPB3
#define PRX_SDI         SDI2_RPA4
#define PRX_SDO         SDO2_RPB5
#define PRX_IRQ         INT2_RPB13
#define PRX_SPI_MODULE  SPI2_MODULE

int main(int argc, char** argv)
{
   /* SPI configuration parameters for nRF24L01 device */
   SpiStandardConfig_t spiPrxConfig = {
      .pinSelect = {
         .sdiPin = PRX_SDI,
         .sdoPin = PRX_SDO,
         .ss1Pin = PRX_CS
      /*   sckPin on RPB15   */
      },
      .isMasterEnabled = true,
      .frameWidth = SPI_WIDTH_8BIT,
      .sckFreq = 1000000,
      .clkMode = SPI_CLK_MODE_0
   };

   /* nRF24L01 configuration parameters for PRX mode */
   NrfPrxConfig_t prxConfig = {
      .spiSfr = &PRX_SPI_MODULE,
      .isAck = true,
      .dataRate = NRF_RF_DR_2000,
      .rfChannel = NRF_RF_CH_2,
      .pipeAddr = {
         .pipe0 = 0x7878787878,
         .pipe1 = 0xB3B4B5B6F1,
         .pipe2 = 0xB3B4B5B6CD,
         .pipe3 = 0xB3B4B5B6A3,
         .pipe4 = 0xB3B4B5B60F,
         .pipe5 = 0xB3B4B5B605
      },
      .pinConfig = {
         .cePin = PRX_CE,
         .csPin = PRX_CS,
         .irqPin = PRX_IRQ
      }
   };

   uint8_t prxTxData[32];
   uint8_t prxTxData[32];

   /* Initialise SPI modules for nRF24L01+ communication */
   SPI_ConfigSfrStandardMode(&PRX_SPI_MODULE, spiPrxConfig);

   /* Configure the device as a PRX */
   NRF_ConfigPrxSfr(prxConfig);

   /* Configure payload structure */
   NrfPayloadConfig_t prxPayloadConfig = NRF_ConfigPrxPayloadStruct(
   prxConfig);

   /* Store ACK payload */
   prxTxData[0] = 0x12;
   prxTxData[1] = 0x23;
   prxTxData[2] = 0x45;
   NRF_StoreAckPayload(prxPayloadConfig, NRF_RX_PIPE_5, prxTxData, 3);

   /* Continue when RX FIFO fully loaded */
   while( NRF_IsRxFifoLoading() )
   {
      /* Main program execution */
   }

   /* Start reception */
   NRF_StartReception(prxPayloadConfig, prxRxData);

   while(1)
   {
      /* Main program execution */
   }

   return 0;
}
```

7.2 One packet transmission

This example provides the code for setting up the nRF24L01 device as a transmitter (the PTX mode). As in the case of the previous example (7.1) an SPI SFRs and a nRF24L01 device registers are configured prior to the start of the transmission. This operation is polling-based, therefore all the data processing is done within one function. The data of the received ACK payload (if any) is available for use immediately after the function `NRF_SendReceivePayload()` execution completes. In the addition this function's return state, the user check the completed operation status by evaluating the return value of the function `NRF_ReadStatus()`.

```c
/** Custom libs **/
#include "nRF24L01.h"

/** Macros **/
#define PTX_CS          GPIO_RPB7
#define PTX_CE          GPIO_RPB10
#define PTX_SDI         SDI1_RPB8
#define PTX_SDO         SDO1_RPB11
#define PTX_IRQ         INT2_RPB13
#define PTX_SPI_MODULE  SPI1_MODULE
#define PTX_ADDR        (0xB3B4B5B605)

int main(int argc, char** argv)
{
   /* SPI configuration parameters for nRF24L01 device */
   SpiStandardConfig_t spiPtxConfig = {
      .pinSelect = {
         .sdiPin = PTX_SDI,
         .sdoPin = PTX_SDO,
         .ss1Pin = PTX_CS
      /*   sckPin on RPB14   */
      },
      .isMasterEnabled = true,
      .frameWidth = SPI_WIDTH_8BIT,
      .sckFreq = 1000000,
      .clkMode = SPI_CLK_MODE_0
   };

```

```
    /* nRF24L01 configuration parameters for PTX mode */
    NrfPtxConfig_t ptxConfig = {
        .spiSfr = &PTX_SPI_MODULE,
        .isAck = true,
        .retrDelay = NRF_ARD_1000,
        .retrCount = NRF_ARC_15,
        .rfChannel = NRF_RF_CH_2,
        .rfPower = NRF_RF_PWR_MIN,
        .dataRate = NRF_RF_DR_2000,
        .pinConfig = {
            .cePin = PTX_CE,
            .csPin = PTX_CS,
            .irqPin = PTX_IRQ
        }
    };

    /* Initialise SPI modules for nRF24L01+ communication */
    SPI_ConfigSfrStandardMode(&PTX_SPI_MODULE, spiPtxConfig);

    /* Configure the device as a PTX */
    NRF_ConfigPtxSfr(ptxConfig);

    /* Configure payload structure */
    NrfPayloadConfig_t ptxPayloadConfig = NRF_ConfigPtxPayloadStruct(
    ptxConfig, PTX_ADDR);

    /* Test payload data */
    uint8_t ptxTxData[] = { 0x7A, 0x32, 0x3D, 0x01,
                            0xD9, 0x76, 0x63, 0x5F,
                            0x6D, 0x3F, 0xE2, 0xFD,
                            0x55, 0x8E, 0xEE, 0xE7,
                            0x90, 0xFC, 0x57, 0xE1,
                            0xE8, 0x6C, 0xFD, 0xAC,
                            0x04, 0xAE, 0x45, 0xF5,
                            0xD3, 0x61, 0x20, 0x0D};

    /* Send and receive the ACK payload */
    NRF_SendReceivePayload(ptxPayloadConfig, ptxRxData, ptxTxData,
    sizeof(ptxTxData));

    while(1)
    {
        /* Main program execution */
    }

    return 0;
}
```

The same operation can be ISR-based to reduce the MCU overhead. After SPI and nRF24L01 configuration process the function `NRF_SendPayload()` starts loading TX FIFO and control is returned to the main program immediately. After TX FIFO loading completes an ISR is triggered where transmission starts. After successful transmission and/or reception of ACK payload from the remote receiver, the data is read and the operation status can be manually read by using the function `NRF_ReadStatus()`.

A time diagram of this operation's measurement is provided and explained in the appendix (A).

```
/** Custom libs **/
#include "nRF24L01.h"

/** Macros **/
#define PTX_CS          GPIO_RPB7
#define PTX_CE          GPIO_RPB10
#define PTX_SDI         SDI1_RPB8
#define PTX_SDO         SDO1_RPB11
#define PTX_IRQ         INT2_RPB13
#define PTX_SPI_MODULE  SPI1_MODULE
#define PTX_ADDR        (0xB3B4B5B605)

int main(int argc, char** argv)
{
   /* All the required structures configured until here */

   /* Initialise SPI modules for nRF24L01+ communication */
   SPI_ConfigSfrStandardMode(&PTX_SPI_MODULE, spiPrxConfig);

   /* Configure the device as a PTX */
   NRF_ConfigPtxSfr(ptxConfig);

```

```c
    /* Configure payload structure */
    NrfPayloadConfig_t ptxPayloadConfig = NRF_ConfigPtxPayloadStruct(
    ptxConfig, PRX_ADDR);

    /* Test payload data */
    uint8_t ptxTxData[] = { 0x7A, 0x32, 0x3D, 0x01,
                            0xD9, 0x76, 0x63, 0x5F,
                            0x6D, 0x3F, 0xE2, 0xFD,
                            0x55, 0x8E, 0xEE, 0xE7,
                            0x90, 0xFC, 0x57, 0xE1,
                            0xE8, 0x6C, 0xFD, 0xAC,
                            0x04, 0xAE, 0x45, 0xF5,
                            0xD3, 0x61, 0x20, 0x0D};

    /* Send payload */
    NRF_SendPayload(ptxPayloadConfig, ptxRxData, ptxTxData, sizeof(
    ptxTxData));

    while(1)
    {
        /* Main program execution */
    }

    return 0;
}
```

A Demonstration time diagram

The captured time diagram shown by the figure (A.1) shows how the operation of one packet transmission and reception is handled on the receiver (upper diagrams) and the transmitter (lower diagrams) side. The orange-colored diagram under the name `PRX_MAIN_EXECUTION` and the purple-colored diagram under the name `PTX_MAIN_EXECUTION` indicate how the main program on both sides is interrupted as the process is carried out. A signal is toggled while a program is waiting for an ISR and is not toggled when ISR handler starts executing.

Note that when the non-ISR function (the `NRF_SendReceivePayload()`) is used, the shown diagram is practically the same but without `PTX_MAIN_EXECUTION` signal toggling until the end of the packet transmission.

Additionally, it can be noted that the main program execution gets interrupted during the PTX mode TX FIFO loading and during PRX mode RX FIFO reading. This is because the SPI buffer needs to be reloaded after fully loaded TX FIFO was emptied. And vice versa for the PRX mode where the fully loaded RX FIFO needs to be emptied completely before the SPI bus continues the operation.

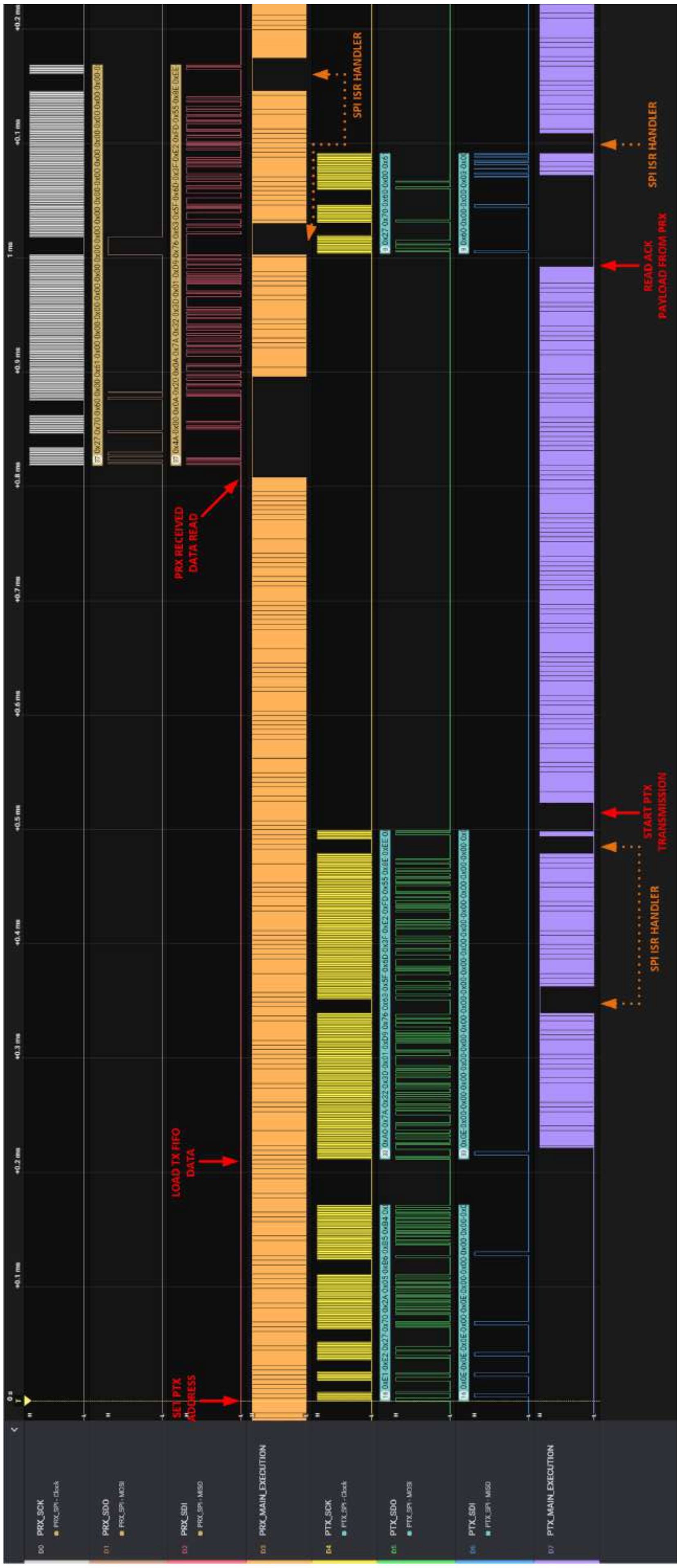

Figure A.1: One packet transmission and reception diagram

B Revision history

Revision 1.0 (October 2022)

This is the initial released version of this document.

Revision 1.1 (October 2023)

Added `NRF_SetUserCallback()` and `NRF_ReleaseUserCallback()` functions.