

Kari Kuivalainen

A PSoC API for NRF24L01

**Contents**

[1 Introduction 1](#_Toc356125112)

[2 Background 1](#_Toc356125113)

[2.1 The Radio 1](#_Toc356125114)

[2.2 The Platform 1](#_Toc356125115)

[3 Implementation 2](#_Toc356125116)

[3.1 The API 2](#_Toc356125117)

[3.2 The protocol 2](#_Toc356125118)

[4 Testing and results 3](#_Toc356125119)

[5 Evaluation and conclusion 3](#_Toc356125120)

[References 5](#_Toc356125121)

# Introduction

The goal of the project was to write an application programming interface (API) for the Nordic Semiconductor NRF24L01P radio frequency (RF) tranceiver, and the Cypress PSoC microcontroller platform. An API is a set of functions that make it simpler to program the target platform or device. Having an API for the RF chip makes it easier for others to implement wireless communication in their own projects, without having to laboriously study the chip datasheet.

# Background

## The Radio

The NRF24L01P is a low power RF tranceiver that works in the 2.4-2.525 GHz range. It is a half-duplex radio, i.e. it cannot receive and transmit simultaneously. One striking feature of the radio is its multiceiver function - the ability to listen to up to 6 channels at once - making it especially useful for wireless sensor networks. It supports bandwidths of 250 kbps, 1 Mbps, and 2 Mbps. It is operated using the four-wire serial peripheral interface (SPI) bus.

## The Platform

The platform the API is written for is the Cypress programmable system on chip (PsoC) micro controller. The PSoC is the platform of choice on the embedded systems programming course, serving as an introduction to embedded design. The designation 'system on chip' means there are different kinds of components built inside one chip. One can add timers, communication protocols, analog-to-digital converters (ADCs), and other building blocks that can be used in creating embedded system projects. The real power of the PSoC comes from the PSoC Developer integrated development environment (IDE). The IDE features a drag-and-drop interface, where the developer can quickly configure the various building blocks at their disposal. It also serves as a quick way to access all the documentation about the CPU itself and the available components.

# Implementation

## The API

When starting the project there was one clear goal: the API should be simple. Only the essential functions such as configuring the radio or sending a packet would be included in the main API source components, nrf24l01p.c and nrf24l01p.h.

At its core the API has functions for sending a command to the radio through SPI. All functions that deal with the various settings of the radio use nrf24SetRegister and nrf24GetRegister functions for reading and writing the various configuration registers.

The API has an interface for reading received packets and uploading and sending packets with the nrf24ReadPayload and nrf24SendPayload functions. The nrf24Transmit function is used to initiate the packet sending process after a packet has been uploaded to the radio.

A function for giving the radio basic configuration parameters that make the radio able to either receive or transmit packets is included. There are also quick functions to power down, sleep (standby), and power up the radio. The status of the interrupts and FIFOs can be queried using the nrf24GetStatus function.

The nrf24l01p.h header file includes all the register address and command byte definitions in abbreviations corresponding to the datasheet.

## The Protocol

The second part of the project was an attempt at building a basic protocol. The radio.h specifies a number of commands that the developer can use when creating a control interface for the radios. For example the command COMMAND\_DATA (0x00) indicates that the message only contains data where as COMMAND\_STANDBY (0x22) tells the receiver to sleep for a duration specified in the message. The protocol could be made to include a command that for example forces the receiving radio to change the channel it is listening on to one specified in the COMMAND packet.

The protocol consists of command name definitions and the packet structure. The packet is structured as a 32 byte (the maximum packet size the radio supports) structure consisting of: 1 byte packet id, 5 byte address, 1 byte timestamp, 1 byte command word, 1 byte indicating the payload size and the 23 byte payload containing the actual sent data. If a command supplies any configuration data, such as the time to sleep or the channel to switch to, it is placed in the payload area. Currently the protocol and packet structure only allow for 1 command per sent packet. This makes the protocol inefficient due to the amount of overhead in sending 32 bytes every time a command is sent to the target device.

For dealing with the packet structure, the radio.c part of the API contains a set of primitive functions that serve as testing prototypes. For example the sendPacket command takes a 32 byte packet and uses the nrf24SendPayload to upload it to the radio and then tries to transmit it until it succeeds.

# Testing and results

The testing was done on two CY29466 PSoC chips and NRF24L01P radios. One radio was designated primary transmitter (PTX) and the other one primary receiver (PRX). When turned on the PRX would listen to a packet continuously until receiving one. It then either prints the contents of a DATA packet or performs the instructions of the COMMAND packet. In the first scenario the PTX first sends a COMMAND packet ordering the PRX to standby for 2 seconds. Both the PRX and the PTX then enter standby mode and wake up after the specified time. After waking up the PTX sends a DATA packet whose contents are displayed on the receiver's LCD. In another scenario the PTX would continuously try to send data at set rate while the PRX would continuously print the contents of the packets on its LCD.

# Evaluation and conclusion

The tests show that the radios work reliably at short distances and that it's possible to create a basic control interface using the protocol and packet structures. All tests were done at very short distance between the transmitter and the receiver, at maximum power level.

The packet and protocol designs are not final and could be greatly improved. The timer system is not highly accurate because there is no clock synchronization between devices yet. This limits the accuracy to tens of milliseconds, which is still good enough for some applications that don't require exact timing.

# Sources

Ashby, Robert.

Designer's guide to the Cypress PSoC / by Robert Ashby. Amsterdam : Elsevier, cop. 2005. - (Embedded technology series.).

ISBN 9780750677806., ISBN 0750677805.