**JELGAVAS TEHNIKUMS**

**Aleks Markuss Balodis**

**Datu ieguve un apmaiņa izmantojot mikro kontrolierus**

**Kvalifikācijas darbs**

**kvalifikācijas ieguvei**

**programmēšanas tehniķis**

**Darba izpildītājs:  410.gr.izgl. Aleks Markuss Balodis**

***paraksts, datums***

**Jelgava**

**2022**

ANOTĀCIJA

Balodis Aleks Markuss. Datu ieguve un apmaiņa izmantojot mikro kontrolierus: kvalifikācijas darbs. Jelgava: JT, 2022. 17 lpp., 1 att., 1 tab., 0 bibl. nos., 0 pielikumi..

Mikro kontrolieri plaši tiek izmantoti mūsdienās dažādās ierīcēs, lai veiktu vienkāršas funkcijas, kur nav nepieciešama liela veiktspēja. Darba mērķis ir ievākt datus par laikapstākļiem ar mikro kontrolieriem un izmantojot bezvadu komunikāciju nosūtīt tos uz serveri, kas tos saglabās datubāzē un attēlos lietotājam viegli pārredzamā veidā mājaslapā.

Mikro kontrolieri paši nespēj izmērīt temperatūru, gaismas daudzumu vai nosūtīt informāciju bez vadiem, bet vien apstrādāt loģiku, tāpēc tiek pievienoti dažādi moduļi vai komponentes. Komponentes kā *LED* tiek pievienotas izmantojot *GPIO* kontaktus, bet moduļi kā, piemēram *nRF24L01* bezvadu modulis, izmanto komunikācijas protokolu *SPI*.

Darba veikšanai nepieciešams būs izmantot dažādas programmēšanas valodas kā *C*. Koda rediģēšanai tiks izmantots *Visual Studio Code* un *Arduino IDE*. Versiju kontroles sistēmai tiks izmantots *GIT*.

ANNOTATION

Balodis Aleks Markuss. Data collection and exchange using microcontrollers – qualification work. Jelgava: JT, 2022, 17 pages, 1 image, 1 tables, 4 used information sources, 0 attachments.

Microcontrollers are widely used today in various devices to perform basic functions which do not require high performance. The purpose of my final year project is to harvest weather data using microcontrollers and then using wireless communication send the harvested data to a server, which would store the data in a database.

Microcontrollers by themselves cannot measure temperature, light or send information wirelessly. They can only process logic, that is why extra modules or components are added to them. Components like *LEDs* are added using *GPIO* pins, but modules like the *nRF24L01* wireless module use a communication protocol, in this case *SPI*.

For this project I will need to use a programming language like *C*. To edit the code, I will be using *Visual Studio Code* and the *Arduino IDE*. For version control I will be using *GIT*.

SATURS

[IEVADS 1](#_Toc105764467)

[1. PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA 2](#_Toc105764468)

[1.1 Ievads 2](#_Toc105764469)

[1.1.1 Dokumentācijas mērķis 2](#_Toc105764470)

[1.1.2 Produkta sfēra 2](#_Toc105764471)

[1.1.3 Dokumentācijas pārskats 2](#_Toc105764472)

[1.2 Vispārīgs apraksts 3](#_Toc105764473)

[1.2.1 Produkta perspektīva 3](#_Toc105764474)

[1.2.2 Produkta funkcijas 3](#_Toc105764475)

[1.2.3 Programmatūras gaita 3](#_Toc105764476)

[1.2.4 Mērķauditorija 3](#_Toc105764477)

[1.3 Funkcionālās prasības 4](#_Toc105764478)

[1.3.1 Datu ievākšana 4](#_Toc105764479)

[1.3.2 Datu pārraide 4](#_Toc105764480)

[1.3.3 Programmatūras saderība 4](#_Toc105764481)

[1.3.4 Datu ievietošana datubāzē 5](#_Toc105764482)

[1.4 Nefunkcionālās prasības 5](#_Toc105764483)

[2. IZSTRĀDEI IZMANTOTĀS TEHNOLOĢIJAS UN PROGRAMMATŪRA 6](#_Toc105764484)

[2.1 Programmatūras izstrādes vide 6](#_Toc105764485)

[2.2 Visual Studio Code izmantotie paplašinājumi 6](#_Toc105764486)

[2.2.1 PlatformIO 7](#_Toc105764487)

[2.2.2 C / C++ Syntax 7](#_Toc105764488)

[3. PRODUKTA KOMPONENŠU APRAKSTS 8](#_Toc105764489)

[3.1 Mikro kontroliera pirmkoda komponentes 8](#_Toc105764490)

[3.1.1 Bezvadu moduļa nRF24L01 bibliotēkas funkcijas 8](#_Toc105764491)

[4. LIETOTĀJA ROKASGARĀMATA 11](#_Toc105764492)

[5. NOBEIGUMS 12](#_Toc105764493)

[5.1 Kopsavilkums 12](#_Toc105764494)

[5.2 Secinājumi 12](#_Toc105764495)

[5.3 Izmantoto informācijas avotu saraksts 13](#_Toc105764496)

# **AKRONĪMI**

|  |  |
| --- | --- |
| *WEB* |  |
| *C / C++* |  |
| *nRF24L01* |  |
| *DHT-11* |  |
| *BIOS* |  |
| *GitHub* |  |
| *MCU Weather* |  |
| *Visual Studio Code / VSC* |  |
| *Arduino* |  |
| *Mongo DB* |  |
| *MCU* |  |
| *SPI* |  |

# **IEVADS**

Darba tēma “Datu ieguve un apmaiņa izmantojot mikro kontrolierus” tika izvēlēta, lai uzlabotu savas prasmes zema līmeņa programmēšanā. Mūsdienās programmētāji arvien vairāk izvēlas pilnveidot savas zināšanas *WEB* vai aplikāciju programmēšanas, bet arvien retāk zema līmeņa programmēšanā – *BIOS* programmēšanā, mikro kontrolieru programmēšanā un cita veida *embedded* programmēšanā.

Darba laikā tika izmantota *C++* programmēšanas valoda un netika izmantotas gatavas bibliotēkas, lai izprastu kā ir strādāt bez abstrakcijas.

Darba mērķis: Izveidot no nulles mikro kontroliera programmatūru, kas var ievākt datus par laika apstākļiem un tos nosūtīt uz citu mikro kontrolierī.

Darba uzdevumi:

* Izveidot bibliotēkas *nRF24L01* bezvadu modulim, *DHT11* temperatūras/mitruma sensoram un gaismas sensoram.
* Izveidot pārbaudi vai saņemtie dati ir neskarti.

# **PROGRAMMATŪRAS PRASĪBU SPECIFIKĀCIJA**

## Ievads

### Dokumentācijas mērķis

Programmatūras prasību specifikācijas nolūks ir aprakstīt prasības uzdevumu plānošanas informācijas sistēmā.

Dokuments paredzēts izstrādātājiem un programmatūras lietotājiem, lai varētu izprast programmatūras funkcijas, darbību un specifiku. Dokumentācija palīdzēs tās lietotājiem integrēt programmatūru veiksmīgi savā projektā.

### Produkta sfēra

Programmas produkta nosaukums ir “*MCU Weather*” – *C++* pirmkods, ko var izmantot savos projektos, lai ievāktu datus par laikapstākļiem un nosūtīt tos izmantojot bezvadu savienojumu.

Programmu var pielietot dažādos veidos, dažas aplikācijas ir termometrs, apkures sistēmas un gaisa mitrinātājs.

### Dokumentācijas pārskats

## Vispārīgs apraksts

### Produkta perspektīva

Izstrādātā programma (pirmkods) ir pieejams vietnē *GitHub*.

Šo pirmkodu var izmantot jebkurš, piemēram, hobijisti, lai integrētu, pielāgotu un izmantotu savām vajadzībām. Apskatot šo kodu arī rodas priekštats par to kā programmēt zemā līmenī.

### Produkta funkcijas

Izstrādātai programmatūrai ir vairākas funkcijas. Pirmā pamata funkcija ir iespēja vienkāršoti strādāt ar *nRF24L01* bezvadu moduli – to konfigurēt savām vajadzībām, sūtīt un saņemt datus. Otrā funkcija ir iespēja veikt temperatūras un gaisa mitruma mērījumus ar *DHT11* sensoru. Trešā funkcija ir veikt apgaismojuma mērījumus ar foto-pretestību. Ceturtā funkcija ir nosūtīt datus uz citu mikro kontrolieri kurš tos apstrādā un pārraida datoram/serverim.

### Programmatūras gaita

Mikro kontrolierim uzsākot darbu tiks inicializēti visi nepieciešamie moduļi – *nRF24L01*, *DHT-11*, Gaismas sensors. Tālāk tiks periodiski ievākti dati – gaismas līmenis, gaisa mitrums, temperatūra un tiks nosūtīti centrālajam kontrolierim. Centrālais kontrolieris savukārt būs pievienots datoram un nosūtīs visus saņemtos datus tam. Pēc tam dators saņemtos datus ievietos datubāzē.

### Mērķauditorija

Programmatūra ir paredzēta hobijistiem kuri vēlas sava projekta ietvaros ievākt datus par laika apstākļiem un tos nosūtīt izmantojot bezvadu savienojumu uz citu mikro kontrolieri.

Projekta pirmkods ir brīvi pieejams ikvienam *GitHub* un to var rediģēt, lai pielāgotu savām vajadzībām

## Funkcionālās prasības

Programmatūras funkcionālās prasības ietilpst šāda funkcionalitāte un sistēmas darbības principi:

* Klienta mikro kontrolieris ievāc sekojošos datus par apkārtni: temperatūru, gaisa mitrumu un apgaismojuma spilgtumu.
* Klienta mikro kontrolieris šos ievāktos datus pārraida ar bezvadu savienojuma palīdzību caur *nRF24L01* moduli.
* Dati tiek nosūtīti periodiski.
* Centrālais mikro kontrolieris saņem datus no klienta.
* Centrālajam mikro kontrolieris ievieto šos datus datubāzē.
* Programmatūra ir saderīga ar *Arduino*.

### Datu ievākšana

Mikro kontrolierim ir jāspēj ievākt dažādi dati par apkārtni. Ievācamajos datos ietilpst:

* Gaisa temperatūra (celsijā)
* Gaisa relatīvais mitrums
* Apkārtnes apgaismojuma spilgtums (lukss)

Apkārtnes apgaismojuma līmenis tiks mērīts izmantojot foto pretestību, savukārt gaisa temperatūra un relatīvais mitrums tiks mērīti izmantojot DHT-11 sensoru.

### Datu pārraide

Mikro kontrolierim, kas ievāc iepriekš minētos datus ir jāspēj sūtīt ievāktie dati un mikro kontrolierim, kas ir pieslēgts pie datora ir jāspēj šie dati saņemt. Tādēļ tiks izmantot nRF24L01 2.4 GHz bezvadu modulis, kas ir spējīgs sūtīt un saņemt datus. Dati tiek nosūtīti un saņemti periodiski, lai nodrošinātu efektivitāti.

### Programmatūras saderība

Izstrādājot programmatūru, tai ir jābūt saderīgai ar *Arduino* platformu. Tas nodrošina ērtu lietošanu.

### Datu ievietošana datubāzē

Centrālajam mikro kontrolierim nosūtot datus uz datoru – datoram ir jāsaņem šie dati caur *Serial* interfeisu. Datoram šie dati ir jāsaņem un jāapstrādā. Pēc tam dati tiks ievietoti servera datubāzē, kura izmanto *MongoDB*.

## Nefunkcionālās prasības

# **IZSTRĀDEI IZMANTOTĀS TEHNOLOĢIJAS UN PROGRAMMATŪRA**

## Programmatūras izstrādes vide

Lai izstrādātu programmatūru *ATmega328P* (*Arduino Uno*) mikro kontrolierim esmu izvēlējies izmantot *Visual Studio Code (VSC)* ar paplašinājumiem *PlatformIO* un *C / C++ Syntax*. *VSC* izvēlējos, jo šis *IDE* pataisa kodu pārskatāmu un tam arī ir *IntelliSense*, kas atvieglo koda rakstīšanu. Paplašinājums *PlatformIO* pielāgo *VSC* darbam ar mikro kontrolieriem ļaujot augšupielādēt kodu uz tiem un izmantot dažas pamata bibliotēkas, kas atvieglo darbu.

Par cik rakstot kodu uz mikro kontrolieriem tiek izmantota *C / C++* valoda paplašinājums *C / C++ Syntax* ļauj *VSC* saprast *C / C++* sintaksi un izcelt to.

*Visual Studio Code (VSC)* ir Microsoft veidota teksta rediģēšanas programma, kurai ir iespējams uzstādīt dažādus paplašinājumus, kas nodrošina lietotājam nepieciešamo funkcionalitāti. *VSC* pamatā nodrošina šīs un citas funkcijas:

* integrētu komandas rindu;
* sintakses pārbaudītāju;
* *IntelliSense*, kas ļauj automātiski pabeigt rakstīto, dod informāciju par funkciju, tās parametriem un tipu un daudz ko citu.

## Visual Studio Code izmantotie paplašinājumi

Lai izstrādātu programmatūru esmu izvēlējies izmantot *Visual Studio Code* ar sekojošiem paplašinājumiem. Šie paplašinājumi padara programmēšanu vieglāku integrējot *VSC* tādas funkcijas kā:

* Koda augšupielāde mikro kontrolierī
* Seriālās konsoles integrācija *VSC*
* *C / C++* sintakses izgaismošana vieglākai koda pārlasāmībai

### PlatformIO

PlatformIO ir paplašinājums priekš Visual Studio Code, kas ļauj izstrādāt programmatūru priekš dažādām embedded platformām. Šī darba ietvaros šis paplašinājums tiek lietots, lai izstrādātu programmatūru Arduino platformai.

PlatformIO nodrošina šādu funkcionalitāti:

* Koda kompilēšanai izvēlētajai platformai
* Koda atkļūdošana
* Bibliotēku viegla pievienošana
* Projektu menedžeri

### C / C++ Syntax

C / C++ Syntax paplašinājums pievieno Visual Studio Code C / C++ programmēšanas valodas atbalstu, IntelliSense un atkļūdošanu.

# **PRODUKTA KOMPONENŠU APRAKSTS**

## Mikro kontroliera pirmkoda komponentes

### Bezvadu moduļa nRF24L01 bibliotēkas funkcijas

**void CS\_Select(void);**

Funkcija iestata *Chip Select* signālu zemajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CS\_UnSelect(void);**

Funkcija iestata *Chip Select* signālu augstajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CE\_Enable(void);**

Funkcija iestata *Chip Select Not (Chip Enable)* signālu augstajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void CE\_Disable(void);**

Funkcija iestata *Chip Select Not (Chip Enable)* signālu zemajā līmenī *SPI* interfeisā starp *MCU* un *nRF24L01*. Šī funkcija neatgriež neko un nepieprasa parametrus.

**void WriteRegister(uint8\_t reg, uint8\_t data);**

Funkcija ļauj rakstīt *nRF24L01* reģistros caur *SPI* interfeisu. Kā parametrus funkcija pieņem reģistra adresi un datus, kas tiks ierakstīti konkrētajā reģistrā. Funkcija neatgriež vērtību.

**void WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);**

Funkcija ļauj rakstīt konkrētā reģistrā vairākas vērtības. Funkcija neatgriež neko, bet kā parametrus pieņem reģistra adresi, rādītāju uz vērtību masīvu un masīva lielumu.

**uint8\_t ReadRegister(uint8\_t reg);**

Funkcija ļauj nolasīt *nRF24L01* reģistra vērtību caur *SPI* interfeisu. Kā parametru funkcija pieņem nolasāmā reģistra adresi. Kā vērtību funkcija atgriež veselu skaitli, kas atbilst reģistra vērtībai.

**void ReadRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);**

Funkcija ļauj nolasīt vairākas vērtības no konkrēta reģistra caur *SPI* interfeisu. Funkcija neatgriež neko, bet kā parametrus pieņem nolasāmo reģistru, rādītāju uz masīvu, kur ievietot nolasītās vērtības un masīva lielumu.

**void SendCommand(uint8\_t cmd);**

Funkcija caur *SPI* interfeisu ļauj nosūtīt komandu uz *nRF24L01*. Funkcija vērtību neatgriež, bet kā parametru pieprasa nosūtāmo komandu.

**void Init(void);**

Funkcija sagatavo *nRF24L01* bezvadu moduli darbībai. Vispirms konfigurācija tiek nodzēsta iestatot visas vērtības uz 0. Pēc tam tiek izslēgta *Auto-Acknowledgment* funkcija, izslēgtas visas datu saņemšanas līnijas, saņemšanas/sūtīšanas līniju platums tiek iestatīts uz 5 baitiem, *Auto-Retransmit* tiek iestatīts uz 250 mikrosekundēm, moduļa frekvence tiek iestatīta uz 2400 MHz un ātrums uz 2Mbps. Funkcija neatgriež vērtību un nepieprasa parametrus.

**void TX\_Mode(uint8\_t \*address, uint8\_t ch);**

Funkcija iestata *nRF24L01* bezvadu moduli sūtītāja režīmā. Kā parametrus funkcija pieprasa rādītāju uz sūtīšanas adreses masīvu un moduļa frekvenci. (Frekvence = 2400 + ch).

**uint8\_t Transmit(uint8\_t \*dati);**

Funkcija pārraida datus, kas atrodas datu sūtīšanas buferī. Atgriež 1, ja dati ir veiksmīgi nosūtīti – citādi 0. Kā parametru funkcija pieņem rādītāju uz sūtāmo datu masīvu. Šim masīva lielumam ir jābūt 32.

**void RX\_Mode(uint8\_t \*address, uint8\_t ch);**

Funkcija iestata *nRF24L01* bezvadu moduli saņēmēja režīmā. Kā parametrus funkcija pieprasa rādītāju uz saņemšanas adreses masīvu un moduļa frekvenci. (Frekvence = 2400 + ch).

**uint8\_t IsDataAvailable(int pipenr);**

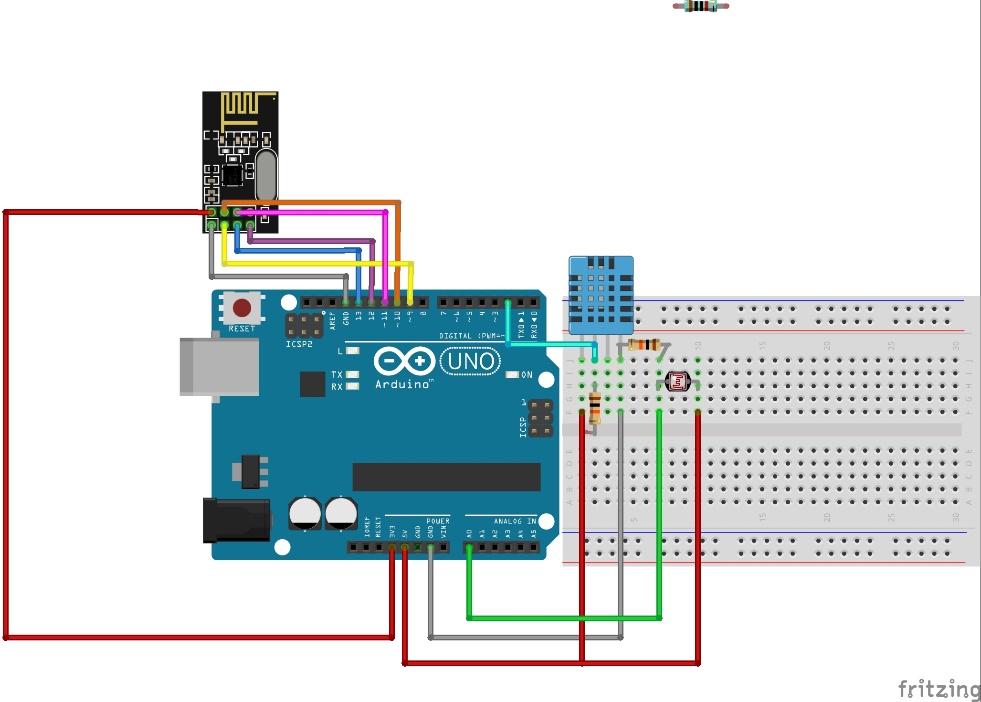
Funkcija pārbauda vai ir dati konkrētajā reģistrā. Ja ir dati konkrētā reģistrā funkcija atgriež 1, citādi 0. Kā parametru pieņem saņemšanas reģistra numuru.

**void Receive(uint8\_t \*data);**

Funkcija ļauj saņemt datus. Neatgriež vērtību. Kā parametru pieņem rādītāju uz masīvu, kur glabāt saņemtos datus. Šim masīva garumam ir jābūt 32.

# **LIETOTĀJA ROKASGARĀMATA**

1. Saslēgt komponentes vadoties pēc elektriskās shēmas 1. attēlā.



Attēls .

# **NOBEIGUMS**

## Kopsavilkums

## Secinājumi

## Izmantoto informācijas avotu saraksts

1. nRF24L01+ Single Chip 2.4GHz Transceiver Preliminary Product Specification v1.0 [Datu lapa] Pieejams: <https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf> [Skatīts: 09.06.2022]
2. Arduino UNO R3 Datasheet [Datu lapa] Pieejams: <https://docs.arduino.cc/static/9982a40d35bb00ec42c0869106c40a0a/A000066-datasheet.pdf> [Skatīts: 09.06.2022]
3. DHT 11 Humidity & Temperature Sensor [Datu lapa] Pieejams: <https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf> [Skatīts: 09.06.2022]
4. Serial Peripheral Interface [Tiešsaistes resurss] Pieejams: <https://en.wikipedia.org/wiki/Serial_Peripheral_Interface> [Skatīts: 09.06.2022]

**PIELIKUMI**

1. Pielikums “nRF24Lib.h”

#include <Arduino.h>

#include <SPI.h>

/\* Registri\*/

#define CONFIG      0x00

#define EN\_AA       0x01

#define EN\_RXADDR   0x02

#define SETUP\_AW    0x03

#define SETUP\_RETR  0x04

#define RF\_CH       0x05

#define RF\_SETUP    0x06

#define STATUS      0x07

#define OBSERVE\_TX  0x08

#define CD          0x09

#define RX\_ADDR\_P0  0x0A

#define RX\_ADDR\_P1  0x0B

#define RX\_ADDR\_P2  0x0C

#define RX\_ADDR\_P3  0x0D

#define RX\_ADDR\_P4  0x0E

#define RX\_ADDR\_P5  0x0F

#define TX\_ADDR     0x10

#define RX\_PW\_P0    0x11

#define RX\_PW\_P1    0x12

#define RX\_PW\_P2    0x13

#define RX\_PW\_P3    0x14

#define RX\_PW\_P4    0x15

#define RX\_PW\_P5    0x16

#define FIFO\_STATUS 0x17

#define DYNPD       0x1C

#define FEATURE     0x1D

/\* Komandas \*/

#define R\_REGISTER    0x00

#define W\_REGISTER    0x20

#define REGISTER\_MASK 0x1F

#define ACTIVATE      0x50

#define R\_RX\_PL\_WID   0x60

#define R\_RX\_PAYLOAD  0x61

#define W\_TX\_PAYLOAD  0xA0

#define W\_ACK\_PAYLOAD 0xA8

#define FLUSH\_TX      0xE1

#define FLUSH\_RX      0xE2

#define REUSE\_TX\_PL   0xE3

#define NOP           0xFF

class nRF24Lib{

    public:

    uint8\_t CSNpin;

    uint8\_t CSpin;

    nRF24Lib(uint8\_t CSN, uint8\_t CS){

            CSNpin = CSN;

            CSpin = CS;

        };

    void CS\_Select(void);

    void CS\_UnSelect(void);

    void CE\_Enable(void);

    void CE\_Disable(void);

    void WriteRegister(uint8\_t reg, uint8\_t data);

    void WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);

    uint8\_t ReadRegister(uint8\_t reg);

    void ReadRegisterMulti(uint8\_t reg, uint8\_t \* data, int size);

    void SendCommand(uint8\_t cmd);

    void Init(void);

    void TX\_Mode(uint8\_t \*address, uint8\_t ch);

    uint8\_t Transmit(uint8\_t \*dati);

    void RX\_Mode(uint8\_t \*address, uint8\_t ch);

    uint8\_t IsDataAvailable(int pipenr);

    void Receive(uint8\_t \*data);

};

1. Pielikums “nRF24Lib.cpp”

#include "nRF24Lib.h"

SPISettings SPIrfsett(4000000, MSBFIRST, SPI\_MODE0);

void nRF24Lib::CS\_Select(void){

    digitalWrite(CSpin, LOW);

}

void nRF24Lib::CS\_UnSelect(void){

    digitalWrite(CSpin, HIGH);

}

void nRF24Lib::CE\_Enable(void){

    digitalWrite(CSNpin, HIGH);

}

void nRF24Lib::CE\_Disable(void){

    digitalWrite(CSNpin, LOW);

}

//Parbaudits

void nRF24Lib::WriteRegister(uint8\_t reg, uint8\_t data){

    uint8\_t buf[2];

    buf[1] = reg|1<<5;

    buf[2] = data;

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    SPI.transfer(buf[1]);

    SPI.transfer(buf[2]);

    CS\_UnSelect();

    SPI.endTransaction();

}

void nRF24Lib::WriteRegisterMulti(uint8\_t reg, uint8\_t \* data, int size){

    uint8\_t buf[2];

    buf[1] = reg|1<<5;

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    SPI.transfer(buf[1]);

    for(int i = 0; i < size; i++)

    {

        SPI.transfer(data[i]);

    }

    CS\_UnSelect();

    SPI.endTransaction();

}

//Parbaudits

uint8\_t nRF24Lib::ReadRegister(uint8\_t reg){

    uint8\_t dati;

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    SPI.transfer(reg);

    dati = SPI.transfer(0);

    CS\_UnSelect();

    SPI.endTransaction();

          //Serial.println(dati);

    return dati;

}

void nRF24Lib::ReadRegisterMulti(uint8\_t reg, uint8\_t \*data, int size){

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    SPI.transfer(&reg, 1);

    \*data = SPI.transfer(0);

    CS\_UnSelect();

    SPI.endTransaction();

}

void nRF24Lib::SendCommand(uint8\_t cmd){

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    SPI.transfer(&cmd, 1);

    CS\_UnSelect();

    SPI.endTransaction();

}

void nRF24Lib::Init(void){

    CE\_Disable();

    WriteRegister(CONFIG, 0);

    WriteRegister(EN\_AA, 0);

    WriteRegister(EN\_RXADDR, 0);

    WriteRegister(SETUP\_AW, 0x03);

    WriteRegister(SETUP\_RETR, 0);

    WriteRegister(RF\_CH, 0);

    WriteRegister(RF\_SETUP, 0x0E);

    CE\_Enable();

}

void nRF24Lib::TX\_Mode(uint8\_t \*address, uint8\_t ch){

    CE\_Disable();

    WriteRegister(RF\_CH, ch);

    WriteRegisterMulti(TX\_ADDR, address, 5);

    uint8\_t cfg = ReadRegister(CONFIG);

    cfg = cfg | (1<<1);

    WriteRegister(CONFIG, cfg);

    CE\_Enable();

}

uint8\_t nRF24Lib::Transmit(uint8\_t \*dati){

    //uint8\_t cmdsend = 0;

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    //cmdsend = W\_TX\_PAYLOAD;

    SPI.transfer(W\_TX\_PAYLOAD);

    for(int i = 0; i < 32; i++)

    {

     SPI.transfer(dati[i]);

    }

    CS\_UnSelect();

    SPI.endTransaction();

    delay(1);

    uint8\_t fifostatus;

    fifostatus = ReadRegister(FIFO\_STATUS);

    if ((fifostatus&(1<<4)) && (!(fifostatus&(1<<3)))){

            //cmdsend = FLUSH\_TX;

            SendCommand(FLUSH\_TX);

            return 1;

    }

return 0;

}

void nRF24Lib::RX\_Mode(uint8\_t \*address, uint8\_t ch){

    CE\_Disable();

    WriteRegister(RF\_CH, ch);

    uint8\_t enrxaddr = ReadRegister(EN\_RXADDR);

    enrxaddr = enrxaddr | (1<<1);

    WriteRegister(EN\_RXADDR, enrxaddr);

    WriteRegisterMulti(RX\_ADDR\_P1, address, 5);

    WriteRegister(RX\_PW\_P1, 32);

    uint8\_t cfg = ReadRegister(CONFIG);

    cfg = cfg | (1<<1) | (1<<0);

    WriteRegister(CONFIG, cfg);

    CE\_Enable();

}

uint8\_t nRF24Lib::IsDataAvailable(int pipenr){

    uint8\_t status = ReadRegister(STATUS);

    if((status&(1<<6))&&(status&(pipenr<<1))){

        WriteRegister(STATUS, (1<<6));

        return 1;

    }

    return 0;

}

void nRF24Lib::Receive(uint8\_t \*data){

    //uint8\_t cmdsend = 0;

    SPI.beginTransaction(SPIrfsett);

    CS\_Select();

    //cmdsend = R\_RX\_PAYLOAD;

    SPI.transfer(R\_RX\_PAYLOAD);

    for(int i = 0; i < 32; i++)

    {

        data[i] = SPI.transfer(0);

    }

    CS\_UnSelect();

    SPI.endTransaction();

    delay(1);

    //cmdsend = FLUSH\_RX;

    SendCommand(FLUSH\_RX);

}