Uređaj male potrošnje za nadzor kvalitete zraka

Fadiga, Ivan

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:300109

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-24**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Uređaj male potrošnje za nadzor kvalitete zraka

Diplomski rad

Ivan Fadiga

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 22.02.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Ivan Fadiga
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-17ARK, 05.10.2018.
OIB studenta:	44544125066
Mentor:	Doc.dr.sc. Tomislav Matić
Sumentor:	Doc.dr.sc. Ivan Vidović
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Ivan Aleksi
Član Povjerenstva 1:	Doc.dr.sc. Tomislav Matić
Član Povjerenstva 2:	Izv.prof.dr.sc. Tomislav Keser
Naslov diplomskog rada:	Uređaj male potrošnje za nadzor kvalitete zraka
Znanstvena grana rada:	**Arhitektura računalnih sustava (zn. polje računarstvo)**
Zadatak diplomskog rada:	U radu je potrebno izraditi i testirati uređaj koji mjeri razne parametre u okolišu koji se tiču kvalitete zraka kao što su klasični parametri temperatura, vlaga, tlak, ali i drugi kao što su: NO2, CO, PM2.5, PM10 itd. Uređaj se temelji na seriji STM mikroupravljača male potrošnje energije. Potrebno je ostvariti bežičnu komunikaciju s udaljenim poslužiteljem. Tema rezervirana za: Ivan Fadiga Sumentor s FERIT-a: Dr. sc. Ivan Vidović
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	22.02.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 02.03.2021.

Ime i prezime studenta:	Ivan Fadiga
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-17ARK, 05.10.2018.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Uređaj male potrošnje za nadzor kvalitete zraka**

izrađen pod vodstvom mentora Doc.dr.sc. Tomislav Matić

i sumentora Doc.dr.sc. Ivan Vidović

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
2. POSTOJEĆA RJEŠENJA ZA NADZOR KVALITETE ZRAKA 2
3. PARAMETRI KVALITETE ZRAKA 5
4. SKLOPOVLJE 6
4.1. Model uređaja 6
4.2. Senzori NO2 i CO plinova 7
4.3. BME680 senzor 8
4.4. PMS7003 senzor PM 1, PM 2.5 i PM 10 čestica 9
4.5. Mikrokontroler STM32L071KB 9
4.6. SARA N210 komunikacijski modul 10
4.7. TPL5010 mjerač vremena 10
4.8. Li-SOCl2 baterija 11
5. PROGRAMSKA PODRŠKA UREĐAJA 12
5.1. Generiranje glavnog programskog koda 12
5.2. Opis glavnog programskog koda 13
5.3. Programski kod za SARA N210 komunikacijski modul 15
5.4. Programski kod za senzore koncentracije NO2 i CO plina 16
5.5. Programski kod za PMS7003 senzor PM čestica 17
5.6. Programski kod za BME680 senzor 17
5.7. Programski kod za mjerenje stanja baterije 18
5.8. Programski kod za mjerač vremena 19
5.9. The things.io platforma 20
6. TESTIRANJE RADA UREĐAJA 22
7. ZAKLJUČAK 25
LITERATURA 27
SAŽETAK 29
ŽIVOTOPIS 31
PRILOZI 32
P.4.1. Shema uređaja 32
P.5.1. Zaglavna datoteka biblioteke za BME680 senzor 35
P.5.2. C datoteka biblioteke za BME680 senzor 38
P.5.3. JavaScript programski kod za aplikaciju u oblaku 43

1. UVOD

U današnje vrijeme kada postoji sve više izvora zagađenja zraka, ali i kada su informacije postale dostupnije, kvaliteta zraka je postala tema o kojoj se sve više priča i koja se sve više prati. Ovim diplomskim radom cilj je napraviti uređaj male potrošnje za nadzor kvalitete zraka, koji omogućuje praćenje koncentracije plinova i čestica u zraku koji utječu na kvalitetu zraka, na određenome području. Uređaj korištenjem ugrađenih senzora uz količinu plinova mjeri i razinu onečišćujućih čestica u zraku kao i temperaturu, vlagu i tlak zraka kao osnovne parametre kvalitete zraka. Izmjerene parametre uređaj šalje u oblak gdje se podaci za svaki parametar prikazuju u dijagramima što olakšava praćenje i čitanje podataka. Podaci se u oblaku mogu spremati određen vremenski period, a osim toga moguće ih je i poslati u bazu podataka gdje ostaju spremljeni dok se ručno ne obrišu što daje mogućnost za kasniju obradu podataka ili njihovo analiziranje.

Uređaj treba mjeriti koncentraciju dušikovog dioksida (NO2) i ugljikovog monoksida (CO) kao osnovne onečišćujuće plinove, a uz zamjenu elektrokemijskih senzora moguće je mjeriti koncentraciju nekih drugih plinova. Uz plinove kao važan parametar uređaj mjeri i količinu lebdećih PM (engl. *particulate matter*) čestica u zraku koje su najveći problem pogotovo u urbanim područjima. Uređaj koristeći bateriju treba raditi autonomno 5 godina bez održavanja te uz mogućnost promjene senzora za plinove ima široko područje primjene, od opisanog nadzora kvalitete zraka, praćenja količina PM čestica, sve do industrijskih primjena nadzora koncentracija određenih plinova. Kako uređaj radi na bateriji za slanje izmjerenih podataka u oblak koristi bežičnu NB-IoT komunikaciju velikog dometa koja osigurava da se podaci jednako dobro i brzo pošalju u urbanim i ruralnim područjima.

U poglavlju 2 opisana su postojeća rješenja sličnih uređaja dostupnih u literaturi. Poglavlje 3 daje informacije o parametrima kvalitete zraka, objašnjava kako plinovi i čestice u zraku utječu na ljudsko zdravlje te koje su najčešće štetne tvari. Četvrto poglavlje opisuje korištene senzore i module te način njihove implementacije u uređaj, a u 5. poglavlju nalaze se informacije o programskoj podršci za uređaj te o korištenoj the things.io platformi. Testiranje uređaja i prikaz podataka dani su u 6. poglavlju, a u 0. poglavlju je dan zaključak rada.

2. POSTOJEĆA RJEŠENJA ZA NADZOR KVALITETE ZRAKA

U ovome poglavlju dan je pregled postojećih rješenja koja imaju slične karakteristike kao uređaj izrađen u ovome diplomskom radu.

Za mjerenje i javno dijeljenje podataka o onečišćenju zraka u [1] je opisan „uSense“ uređaj koji mjeri koncentraciju NO2, CO i O3 (ozon) plinova u zraku, a za prijenos podataka koristi Wi-Fi te se napaja putem baterije. Također, korisnici mogu putem društvenih mreža dijeliti podatke kako bi se osigurao zajednički nadzor kvalitete zraka. Glavni cilj rada [1] bio je istražiti može li jeftin uređaj poput opisanoga „uSense“ uređaja osigurati kvalitetu podataka kako bi se mogao praktično koristiti. Mjerenja dobivena uređajem su uspoređivana s mjerenjima dostupnima od lokalnih nadzornih tijela, stoga se došlo do zaključka da „uSense“ uređaj ne može pružiti dovoljno točna mjerenja kao što su službena mjerenja. Unatoč toj činjenici, podaci dobiveni od „uSense“ uređaja mogu pružiti korisne informacije o kvaliteti zraka na određenom području.

U [2] opisana je eksperimentalna studija praćenja onečišćenja zraka u stvarnom vremenu korištenjem senzora na gradskim autobusima u sklopu projekta „GreenIoT“. U radu je opisano korištenje „Libelium Waspmote Plug & Sense!“ uređaja koji su postavljeni na gradski autobus kako bi se pratilo onečišćenje zraka u Švedskom gradu Uppsala. Uređaj za slanje podataka koristi 4G mrežu, a mjeri koncentraciju NO2 i CO plinova te temperaturu, vlagu i tlak zraka uz mogućnost određivanja GPS koordinata. Korištenjem naprednog Matlab algoritma odabrana je 21 ruta u gradu, a zatim su montirani uređaji na autobuse na tim rutama kako bi se prikupili podaci iz cijeloga grada. Slanje podataka odrađeno je HTTP zahtjevom na posebno izrađenu aplikaciju, a PHP kodom u pozadini aplikacije obavljeno je spremanje podataka u MongoDB bazu podataka. Osim prikupljanja i analize podataka te integracije uređaja na vozila, u radu je procijenjena kvaliteta odabrane komunikacije i kvaliteta dobivenih podataka.

Nadalje u [3] autori opisuju sustav za mjerenje onečišćenja zraka ispušnim plinovima vozila i sitnim česticama koji je namijenjen za postavljanje na javne bicikle. Kao rezultat, izgrađena je mobilna senzorska mreža za praćenje onečišćenja zraka na sustavu javnih bicikala. Razvijeni se uređaj sastoji od senzora ispušnih plinova, senzora PM čestica, Bluetooth modula, GPS modula te modula za SD karticu. U radu autori opisuju da uređaj tijekom vožnje bicikla na SD karticu zapisuje mjerenja senzora zajedno s podacima o lokaciji i vremenu. Nakon što se bicikl vrati u stanicu, podaci se prebacuju u podatkovni centar putem Bluetooth komunikacije. Autori opisuju i testiranje uređaja koje se sastoji od dva mjerenja, prvoga u zatvorenom prostoru i drugoga pored ispušne cijevi vozila. Usporedbom podataka vidljivo je da senzor mjeri puno više PM čestica pored

ispušne cijevi što je i očekivano. Nakon navedenoga testiranja uređaj je postavljen na bicikl kako bi mjerio onečišćenje tijekom vožnje, a zatim je s podacima napravljena mapa mjerenja.

Kako bi se omogućilo jeftinije praćenje podataka o onečišćenju zraka, u članku [4] opisan je razvoj i testiranje sustava pod nazivom „GasMobile“. Sustav se sastoji od uređaja za mjerenje koncentracije ozona u zraku te mobilne aplikacije koja prikuplja i obrađuje podatke sa senzora te ih šalje na server što omogućuje izradu mape zagađenja. Sustav je izrađen s mogućnošću prilagodbe za mjerenje drugih plinova uz određene izmjene. Također, opisana je i analiza utjecaja vjetra na senzor te kompenzacija toga utjecaja. U radu autori daju mapu mjerenja ozona korištenjem sustava te usporedbu dobivenih rezultata s rezultatima ovlaštenih mjernih stanica.

Kako bi dobili trodimenzionalnu prostornu raspodjelu koncentracije PM čestica, autori u [5] su razvili i testirali bespilotni sustav senzora. Na početku je opisan utjecaj PM čestica na zdravlje i okoliš, a zatim su opisana 4 načina mjerenja PM čestica. Nadalje opisan je dizajn i izrada sustava za mjerenje PM čestica. Autori naglašavaju važnost mogućnosti promjene frekvencije mjerenja i metode pohrane u stvarnom vremenu što je jedna od funkcionalnosti razvijenog sustava. Nakon izrade sustava opisan je proces kalibracije sustava korištenjem podataka nacionalne stanice za mjerenje podataka te utjecaj propelera na mjerenja. Također, autori daju rezultate provedenih mjerenja u polju žita te trodimenzionalni prikaz raspodjele PM čestica.

Autori u [6] opisuju razvoj senzorske mreže za praćenje kvalitete zraka duž maratonske rute u Hong Kongu te prikaz mjerenja u stvarnom vremenu. Prvo je dan opis zahtjeva na uređaj za mjerenje te razvoj samoga uređaja. Također, u prvom poglavlju autori daju rezultate testiranja uređaja u laboratoriju i na otvorenome te korekcijsku funkciju koja ispravlja pogrešku mjerenja zbog utjecaja temperature i vlage zraka. Nakon izrade i testiranja uređaja, opisane su lokacije uređaja s obzirom na utjecaj prometa na kvalitetu zraka te se opisuje izračun indeksa kvalitete zraka. U zadnjem poglavlju autori su dali rezultate testiranja uređaja te rezultate mjerenja tijekom održavanja maratona i usporedbu s mjerenjima stanica kojima upravlja ured za zaštitu okoliša.

U [7] autori opisuju sustav za praćenje kvalitete zraka u zatvorenom prostoru koji se sastoji od senzorskih čvorova, poveznika te IoT servera za prikaz podataka. Svaki senzorski čvor sastoji se od senzora za mjerenje koncentracije 6 plinova, temperature i vlage zraka, baterije i XBee PRO modula za komunikaciju. Za prikupljanje podataka, obradu i slanje na mrežni poslužitelj koriste poveznik koji je napravljen korištenjem Raspberry Pi 2 B mikroračunala i XBee PRO S2 modula za komunikaciju s čvorovima. U nastavku autori opisuju izrađenu programsku podršku za upravljanje senzorskim čvorovima, upravljanje poveznikom, obradu podataka i slanje na IoT

poslužitelj. U radu je dan prikaz podataka te analiza korištene komunikacije s obzirom na gubitke u prijenosu podatka.

Neka od opisanih rješenja mjere samo razinu PM čestica ili koncentraciju određenih plinova dok druga mjere oboje kao i uređaj koji je izrađen i opisan u ovome radu. Ovisno o potrebama uređaji koriste različite senzore za mjerenje određenih parametara, a neki koriste elektrokemijske senzore za plinove poput senzora korištenih u ovome uređaju. Svi uređaji za očitavanje senzora koriste određene mikrokontrolere ili mikroprocesore. Uređaji za slanje podataka koriste određene komunikacijske module koji koriste Bluetooth, ZigBee, Wi-Fi ili LTE komunikaciju. Niti jedan opisani uređaj ne koristi NB-IoT komunikaciju što je glavna prednost uređaja opisanoga u ovome radu.

3. PARAMETRI KVALITETE ZRAKA

Na kvalitetu zraka utječu plinovi i lebdeće čestice koje onečišćuju zrak, ali pri tome je važno napomenuti kako nisu svi plinovi onečišćivači i štetni, naprotiv neki su čak i potrebni za određene procese u prirodi. Prema tome onečišćenje zraka je postojanje onečišćujućih tvari u atmosferi u razinama koje štetno utječu na ljudsko zdravlje i okoliš. Kako bi se zrak održavao čistim i kvalitetnim za ljudsko zdravlje i okoliš, potrebno je mjeriti koncentraciju štetnih tvari te usporedbom s referentnim vrijednostima donositi odluke kojima bi se smanjila proizvodnja štetnih tvari. Za praćenje kvalitete zraka potrebno je prije svega odrediti koje tvari spadaju u skupinu štetnih. PM čestice su štetna tvar koja najviše šteti ljudskom zdravlju jer su lagane i veličine mikrometra te lako ulaze u dišni sustav i krvotok. Zbog svoje veličine brzo se prenose zrakom, a nastaju prirodnim promjenama poput vulkanskih erupcija, šumskih požara ili pješčanih oluja, ali i ljudskim djelovanjem. Najčešće se u sitnim četicama nalazi čađa koja nastaje nepotpunim izgaranjem fosilnih goriva i drva. Čađa sadrži ugljik koji štetno utječe na zdravlje ljudi i doprinosi klimatskim promjenama jer upija toplinu sunca i zagrijava atmosferu. Lebdeće čestice mogu sadržavati i neke teške metale poput arsena, žive, kadmija i nikla koji imaju teške posljedice na ljudski organizam ako je dugotrajno izložen njihovom utjecaju.

Osim lebdećih čestica, u zraku se nalaze i druge tvari koje onečišćuju zrak i štetno utječu na ljudsko zdravlje, a to su plinovi nastali kemijskim procesima poput izgaranja fosilnih goriva. Izgaranjem fosilnih goriva plinovi iz atmosfere međusobno reagiraju jedni s drugima te nastaju novi plinovi koji imaju štetan utjecaj. Neki od tih plinova su dušikov dioksid, ugljični monoksid, amonijak (NH_3) te drugi kojih ima u manjim koncentracijama. Također, jedan od štetnijih plinova u najnižem sloju atmosfere je i ozon koji nastaje složenim kemijskim procesima kisika, dušičnih oksida i ugljičnog monoksida. Praćenjem koncentracija ovih plinova moguće je lokalizirati njihove izvore te umanjiti štetni utjecaj na ljudsko zdravlje.

Prema opisanom utjecaju vidljivo je da su za ljudsko zdravlje najštetnije lebdeće čestice te plinovi dušikov dioksid, amonijak i ugljikov monoksid. Prema tome, kako bi se pratila kvaliteta zraka, ovim radom izrađen je uređaj koji mjeri koncentraciju PM 1 (čestice prašine manje od 1 mikrometra), PM 2.5 (čestice manje od 2.5 mikrometra) i PM 10 (čestice manje od 10 mikrometara) čestica kao i koncentraciju NO_2 i CO plinova.

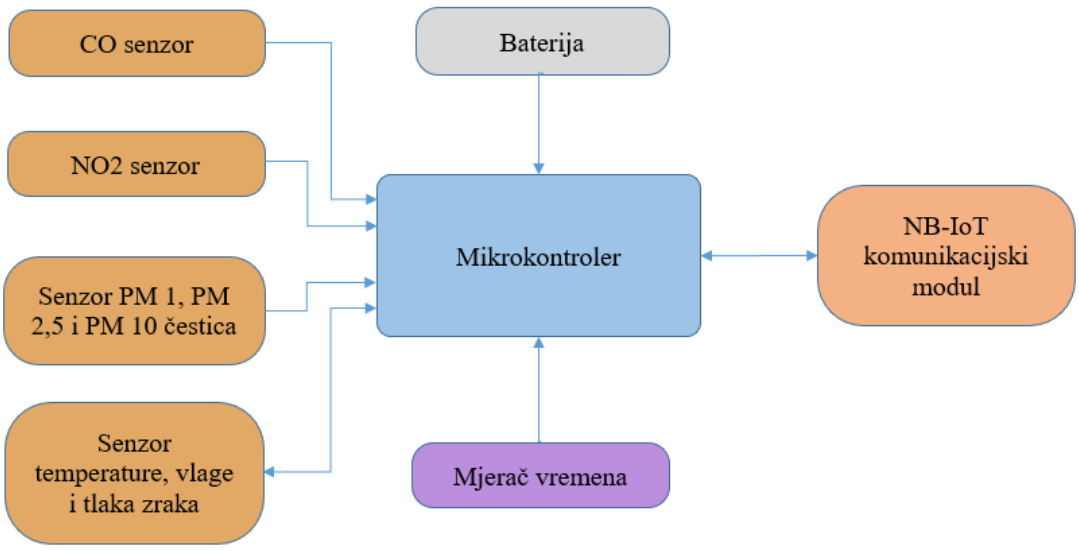
4. SKLOPOVLJE

U ovom poglavlju opisan je model uređaja male potrošnje za nadzor kvalitete zraka i njegovi podsustavi koji zajedno čine funkcionalnu cjelinu. Za svaki podsustav je opisana njegova funkcija u uređaju te potrebne komponente za rad podsustava i način na koji su međusobno povezane. Svaki podsustav odabran je da osigura nisku potrošnju energije kako bi uređaj mogao raditi barem 5 godina na bateriji bez ikakvog ljudskog djelovanja. Razlog za dugi radni vijek je što se uređaji za nadzor kvalitete zraka postavljaju na razna područja koja često nisu lako dostupna te se zbog toga treba smanjiti bilo kakva potreba za ljudskim djelovanjem za rad uređaja. Shema cijelog sklopovlja uređaja dana je u prilogu P.4.1.

4.1. Model uređaja

Uređaj se sastoji od senzora za mjerenje koncentracije NO2 i CO plinova u zraku, senzora za temperaturu, vlagu i tlak zraka te senzora za PM 1, PM 2.5 i PM 10 čestice. Osim senzora, uređaj ima i komunikacijski modul koji omogućuje slanje podataka na oblak odakle se podaci mogu prikazivati na raznim uređajima koji imaju pristup internetu. Kako bi uređaj mogao podatke slati u unaprijed definiranim vremenskim intervalima, koristi se mjerač vremena koji u određenom periodu budi mikrokontroler. Mikrokontroler upravlja cijelim uređajem, čita podatke sa senzora te ih uz pomoć navedenoga komunikacijskog modula šalje na oblak, a za napajanje svih komponenti koristi se Li-SOCl2 baterija.

Na slici 4.1. prikazana je blokovska shema uređaja na kojoj su vidljive osnovne komponente uređaja i veze između njih.

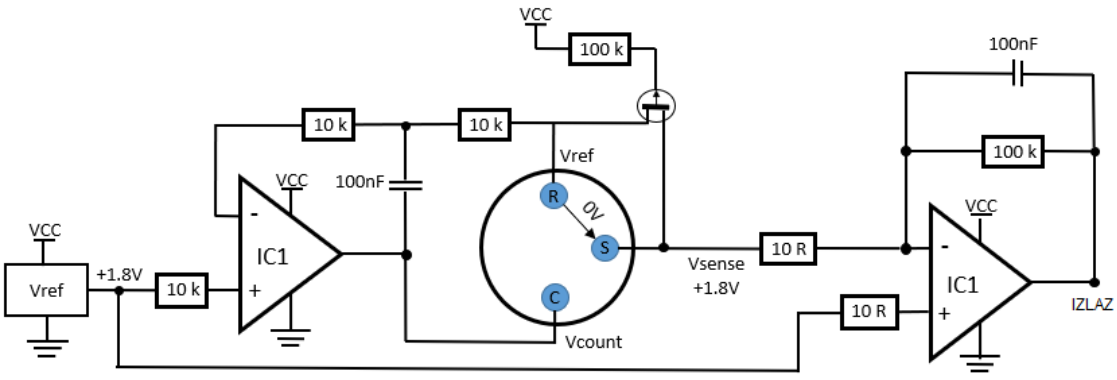


Slika 4.1. Blok shema uređaja.

4.2. Senzori NO2 i CO plinova

Za mjerenje koncentracije NO2 i CO plinova u zraku odabrani su elektrokemijski senzori SGX-4NO2 i SGX-4CO. Elektrokemijski senzori detektiraju plin stvarajući kemijsku reakciju između određenog plina i kisika koji se nalaze u senzoru te ta reakcija proizvodi malu struju koja je proporcionalna koncentraciji plina. Za rad senzora potrebno je dodati potenciostat između elektroda senzora i strujno naponski konverter na izlaz senzora kako je opisano u nastavku. Potenciostat precizno održava željenu razliku potencijala između radne (engl. *work*) i referentne (engl. *reference*) elektrode senzora mijenjajući napon brojačke (engl. *counter*) elektrode koristeći operacijsko pojačalo. Za odabrane senzore, prema dokumentaciji proizvođača [8], [9] i [10], razlika potencijala između radne i referentne elektrode treba biti 0 volta. Kako se uređaj napaja s baterije te postoji samo pozitivan potencijal u odnosu na masu (negativan pol) baterije te operacijsko pojačalo ne bi moglo ispravno regulirati napon na brojačkoj elektrodi senzora, postavljena je virtualna masa za senzor. Virtualna masa je ostvarena korištenjem naponske reference koja je postavljena na 1.8 volta što je polovina napona napajanja. Time je osigurana ispravna regulacija napona brojačke elektrode te održavanje konstantne razlike potencijala između radne i referentne elektrode. Operacijsko pojačalo je spojeno na način da se koristi negativna povratna veza s referentne elektrode, a pozitivni ulaz operacijskog pojačala se spaja na referentnu masu da se omogući ispravna regulacija kako je prethodno objašnjeno. Kako je razlika potencijala između radne i referentne elektrode 0 volti, dodan je tranzistor koji prilikom gašenja kruga potenciostata kratko spaja te dvije elektrode. To osigurava da se prilikom mirovanja senzora ne promijeni razlika potencijala između radne i referentne elektrode koja bi značila duže vrijeme mjerenja.

Na slici 4.2. vidljive su osnovne komponente korištene za ostvarivanje kruga potenciostata i strujno naponskog konvertera te način na koje su povezane.



Slika 4.2. Shema potenciostata i strujno naponskog konvertera.

Kako senzor na izlazu daje struju koja se ne može mjeriti s ADC-om (analogno digitalnim pretvarač) na mikrokontroleru dodan je strujno naponski konverter. Zbog toga što je izlazna struja senzora reda nanoampera, operacijskim pojačalom se osim pretvorbe te struje u napon pojačava izlazni signal kako bi se dobile vrijednosti napona reda milivolta koji je moguće mjeriti ADC-om mikrokontrolera.

Senzor za NO2 plin može mjeriti koncentraciju od 0 do 30 ppm-a i pri tome na izlazu daje 600 nanoampera po ppm-u. Drugi senzor za CO plin može mjeriti koncentraciju plina u većem rasponu od 0 do 1000 ppm-a te zbog toga po ppm-u daje struju od 70 nanoampera. Senzori mogu raditi na temperaturama od -30 do +50 stupnjeva Celzijevih i uz vlažnost zraka od 15 do 90%.

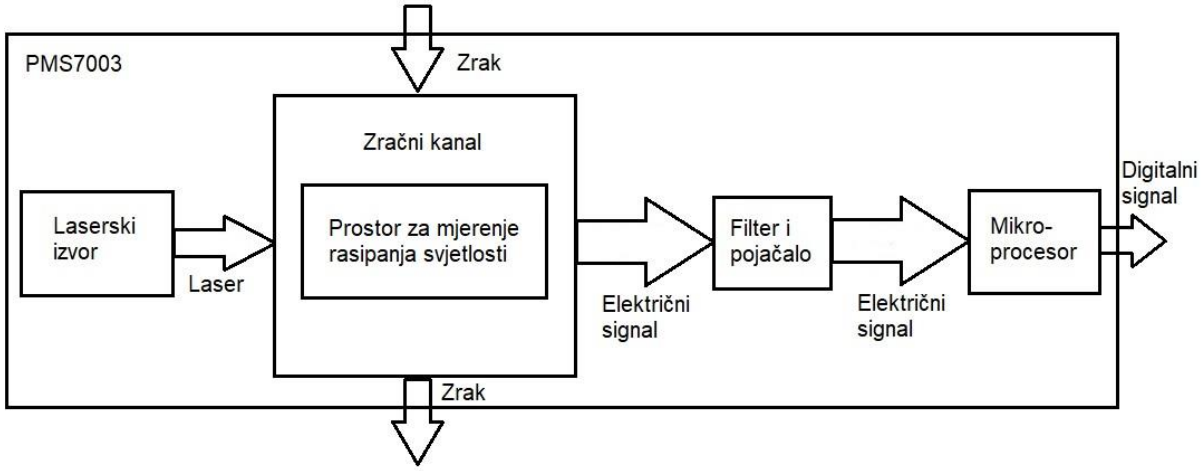
4.3. BME680 senzor

BME680 [11] je senzor koji omogućuje mjerenje četiri parametra od kojih se u uređaju koriste temperatura, vlaga i tlak zraka, dok se aproksimacija zagađenosti zraka ne koristi jer se mjerenjem tog parametra značajno produžuje vrijeme mjerenja i povećava potrošnja energije. Za te potrebe koriste se drugi senzori (Slika 4.1.) koji mjere koncentraciju pojedinih plinova u zraku iz čega se dobije podatak o zagađenosti zraka. Za mjerenje tlaka zraka senzor koristi piezoelektrični MEMS (mikro-elektro-mehanički-sustav) koji stvara električni naboj uslijed promjene tlaka. Vlažnost zraka mjeri koristeći polimer koji mijenja svoju otpornost s promjenom razine vlage, a temperaturu mjeri korištenjem promjene napona vođenja diode s promjenom temperature. Radni napon senzora je od 1.7 do 3.6 volta te je pri mjerenju temperature, vlage i tlaka zraka potreba struje od 2 miliampera. Radno područje senzora je od -40 do +85 stupnjeva Celzijevih uz vlagu od 0 do 100% te tlak zraka između 300 i 1100 hektopaskala.

4.4. PMS7003 senzor PM 1, PM 2.5 i PM 10 čestica

PMS7003 [12] je digitalni senzor koncentracije PM 1, PM 2.5 i PM 10 čestica u zraku. Senzor koristi serijsku komunikaciju s mikrokontrolerom te nakon uključenja mjeri i šalje podatke mikrokontroleru preko navedene komunikacije.

Na slici 4.3. vidljivo je da senzor čestice mjeri koristeći laserski izvor, senzor rasipanja laserske zrake, filtere i mikroprocesor za izračunavanje količine čestica.



Slika 4.3. Funkcijski blok dijagram PMS7003 senzora.

Na senzoru se nalazi i ventilator koji osigurava potreban protok zraka u zračnom kanalu za precizno i točno mjerenje količine navedenih čestica. Mikroprocesor unutar senzora određuje količinu pojedinih čestica koristeći MIE teoriju te podatke o difuziji i apsorpciji laserske svjetlosti. Senzor može mjeriti čestice promjera od 0.3 mikrometra sve do 10 mikrometara. Podatak koji se dobije mjerenjem je količina čestica u mikrogramima po metru kubnom zraka. Maksimalna količina čestica koje senzor može izmjeriti je 1000 mikrograma po metru kubnom zraka s rezolucijom od 1 mikrogram. Raspon temperature pri kojoj senzor može raditi je od -10 do +60 stupnjeva Celzijevih, a vlažnosti od 0 do 99%. Logički dio senzora radi na 3.3 volta, ali zbog ventilatora potrebno je dovesti 5 volta na pin za napajanje senzora te se zbog toga koristi uzlazni pretvarač napona za napajanje ovoga senzora koji je dizajniran kao posebna tiskana pločica.

4.5. Mikrokontroler STM32L071KB

Kako uređaj radi na bateriju te je potrebno ostvariti autonoman rad na dugi niz godina, odabrana je L serija mikrokontrolera koja u spavanju ima potrošnju od nekoliko mikroampera. Nadalje, potrebno je osigurati da mikrokontroler ima I2C komunikaciju koju koriste prethodno opisani senzori, serijsku komunikaciju za razmjenu podataka sa SARA N210 modulom i senzorom PM čestica te ADC-om za mjerenje stanja baterije i signala sa senzora plinova. Prema navedenim

potrebama uređaja, za upravljanje odabran je mikrokontroler STM32L071KB koji ima sve potrebno, a cijenom je bio najprihvatljiviji. Mikrokontroler upravlja radom svih senzora, određuje kada se koji parametar mjeri te po završetku mjerenja gasi senzore kako bi se ostvarila minimalna potrošnja baterije. Uz upravljanje senzorima, mikrokontroler kontrolira rad komunikacijskog modula te određuje kada se određeni podaci šalju i u kojem redoslijedu.

Prema dokumentaciji [13] odabrani mikrokontroler može raditi na naponu od 1.65 do 3.6 volti što odgovara naponu odabrane baterije te ima potrošnju od 290 nanoampera kada je u spavanju. Za komunikaciju ima 4 UART sučelja te 3 sučelja za I2C i 2 sučelja za SPI komunikaciju. Za mjerenje napona na raspolaganju je 12-bitni ADC sa 16 kanala od kojih se samo 3 koriste u uređaju. Kako bi se mogao spremiti programski kod, mikrokontroler ima 192 kilobajta flash memorije, 20 kilobajta radne memorije te 6 kilobajta EEPROM memorije. Uz navedeno, mikrokontroler ima i registar od 20 bajta koji služi za spremanje podataka kada je uređaj u spavanju te 2 pina koji se mogu postaviti za buđenje mikrokontrolera iz spavanja. Pored svega ima i nekoliko mjerača vremena i sučelje za LIN komunikaciju, ali se to ne koristi u sklopu ovoga diplomskog rada.

4.6. SARA N210 komunikacijski modul

Nakon što se podaci očitaju sa senzora, potrebno ih je poslati na oblak gdje će se pohraniti te određeno vrijeme biti dostupni autoriziranim korisnicima. Pošto je cijeli uređaj dizajniran na način da ima malu potrošnju, za komunikaciju je odabran SARA N210 [14] NB-IoT (engl. *Narrow Band Internet of Things*) modul koji je napravljen za uređaje male potrošnje. Sam uređaj koristi NB-IoT tehnologiju koja omogućuje slanje male količine podataka putem mobilne mreže, uz veliku pokrivenost signalom i malu potrošnju. Modul za rad zahtjeva SIM karticu kako bi se mogao spojiti na mrežu operatera, a konekcija između modula i nano SIM kartice je ostvarena pomoću konektora za nano SIM kartice koji je postavljen na stražnjoj strani tiskane pločice. Za ostvarivanje dobroga signala, na uređaj je dodan IPX konektor kako bi se omogućilo spajanje vanjske antene koja se može prilagoditi ovisno u kakvom području uređaj radi i kakva je pokrivenost signalom.

4.7. TPL5010 mjerač vremena

Da bi uređaj slao podatke u jednakim vremenskim intervalima, potrebno je probuditi mikrokontroler u točno određeno vrijeme što osigurava TPL5010 [15] mjerač vremena. TPL5010 je mjerač vremena koji se odlikuje malom potrošnjom od 40 nanoampera, a daje impulsni signal, u unaprijed definiranim vremenskim intervalima, koji se koristi za buđenje mikrokontrolera. Također, uređaj ima i sigurnosni mehanizam koji resetira mikrokontroler u slučaju ako se ne probudi i ne pošalje potvrdu da je probuđen. Uz navedeno sadrži i pin za *RESET* koji, ako se

aktivira, resetira mjerenje vremena te omogućuje i resetiranje mikrokontrolera ako se taj pin drži aktivnim duže od 20 milisekundi. Ova funkcionalnost se može koristiti prilikom postavljanja uređaja kako bi se resetirao i u točno određeno vrijeme započeo proces mjerenja vremena, odnosno mjerenje i slanje ostalih parametara. Mjerač vremena je najjednostavnije rješenje koje zadovoljava preciznošću mjerenja za odabranu aplikaciju, a ima i daleko najmanju potrošnju od sličnih uređaja za mjerenje vremena. Interval između buđenja se postavlja s vanjskim otpornikom koji treba imati jako malu toleranciju (ispod 1%) i veliku otpornost na promjenu temperature. Interval buđenja može se postaviti na rasponu od 100 milisekundi do 2 sata, a za uređaj izrađen u sklopu ovoga diplomskog rada odabran je interval buđenja od 5 minuta.

4.8. Li-SOCl2 baterija

Za napajanje uređaja koristi se litij-tionil klorid baterija koja nije punjiva, a odabrana je zato što uređaj treba raditi autonomno dugi niz godina te zbog malog samopražnjenja. Baterija ima maksimalni napon od 3.6 volti, a kako to ne odgovara svim modulima dodatno je korišten uzlazni pretvarač napona da se dobije 5 volti. Ovisno o tipu baterije, postoji razlika u kapacitetu i maksimalnoj struji koju baterija može dati. Za uređaj je odabrana baterija LSH14 koja ima kapacitet od 5.8 amper-sati i struju pražnjenja od 1.3 ampera što je puno više nego što je potrebno uređaju. Odabirom LSH14 baterije osiguran je stabilan napon svim komponentama uređaja kroz cijeli životni vijek baterije.

5. PROGRAMSKA PODRŠKA UREĐAJA

Programska podrška za uređaj izrađena je u Atollic TrueStudio te je sav programski kod pisan C programskim jezikom, a za mikrokontroler je korištena HAL biblioteka koja omogućava lakše upravljanje mikrokontrolerom i korištenje njegovih funkcionalnosti. Kako bi se olakšalo postavljanje svih potrebnih podsustava mikrokontrolera poput serijske komunikacije, prekidnih rutina, ADC-a i ostaloga, korišten je STM32CubeMX programski alat. Za senzor temperature napisana je biblioteka kako bi bila lakša integracija u glavni kod, a za senzore koncentracije plina, senzor PM čestica, modul za komunikaciju i mjerač vremena napisane su odgovarajuće funkcije koje poboljšavaju preglednost programskog koda.

5.1. Generiranje glavnog programskog koda

Korištenjem programskog alata STM32CubeMX, koji je namijenjen za generiranje kostura programskog koda te inicijalizaciju potrebnih podsustava, olakšan je postupak izrade programske podrške za uređaj. Prvo se krenulo s postavljanjem podsustava za serijsku i I2C komunikaciju, a na slici 5.1. prikazane su podešene postavke za serijsku komunikaciju.

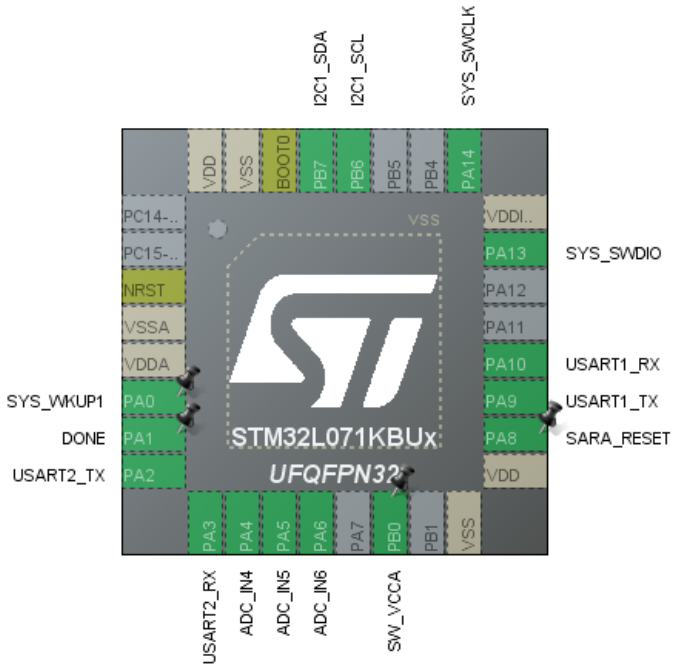
⌄ Basic Parameters	
Baud Rate	9600 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1
⌄ Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable
⌄ Advanced Features	
Auto Baudrate	Disable
TX Pin Active Level Inversion	Disable
RX Pin Active Level Inversion	Disable
Data Inversion	Disable
TX and RX Pins Swapping	Disable
Overrun	Enable
DMA on RX Error	Enable
MSB First	Disable

Slika 5.1. Prikaz postavki za serijsku komunikaciju.

Nakon toga postavljeni su pinovi na kojima se koristi ADC kako bi bilo moguće mjeriti napon na izlazu senzora za koncentraciju NO2 i CO plinova i napon baterije te su tu odmah postavljeni parametri za ADC kao i pin za buđenje mikrokontrolera. Nakon postavljanja svih podsustava potrebnih za očitavanje senzora i komunikaciju između senzora i mikrokontrolera, još su postavljena tri dodatna digitalna pina. Prvi pin služi za paljenje i gašenje napajanja svih senzora,

drugi za slanje signala mjeraču vremena, a treći za resetiranje komunikacijskog modula ako dođe do problema sa slanjem podataka.

STM32CubeMX nakon postavljanja svih podsustava daje pregled koji je prikazan na slici 5.2. te je na njemu vidljivo što je sve postavljeno i na kojim pinovima mikrokontrolera. Dva pina označena sa *SYS_CLK* i *SYS_DIO* služe za programiranje mikrokontrolera, odnosno prebacivanje programskog koda u memoriju mikrokontrolera.



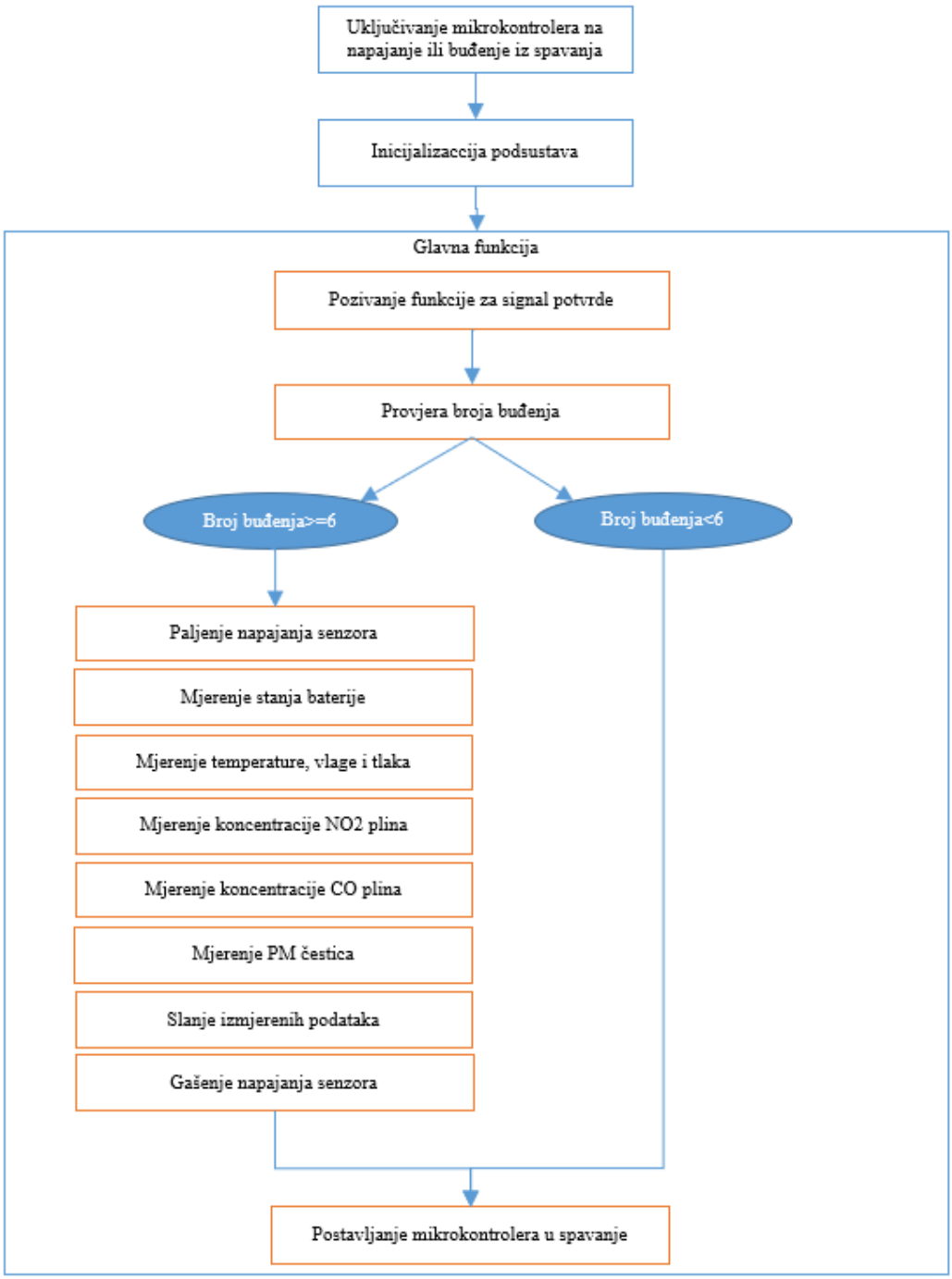
Slika 5.2. Prikaz postavljenih podsustava mikrokontrolera.

5.2. Opis glavnog programskog koda

Kako je ranije opisano, programski kod za svaki senzor ili modul pisan je u zasebnim funkcijama ili bibliotekama koje se pozivaju iz glavnog koda, a funkcije za korištene podsustave napravljene su STM32CubeMX programskim alatom te odmah uključene u glavni programski kod. Unutar njega su napisane i sve funkcije za senzore koje su opisane u sljedećim poglavljima, a pozivaju se u glavnoj funkciji koja se izvodi na mikrokontroleru. Glavna funkcija se pokreće nakon što je mikrokontroler priključen na napajanje te su inicijalizirani svi podsustavi definirani prije glavne funkcije. U glavnoj funkciji pozivaju se pojedine funkcije za svaki senzor kako bi se dobili podaci mjerenja senzora, a nakon toga slijedi slanje podataka na oblak. Za rad s komunikacijskim modulom nisu napravljene funkcije, nego se putem serijske komunikacije šalju AT naredbe kako je opisano u poglavlju za komunikacijski modul. Nakon slanja podataka gase se svi senzori i mikrokontroler se postavlja u spavanje. Mikrokontroler se budi svakih 5 minuta kako je postavljeno na mjeraču vremena, ali mjerenje i slanje podataka vrši svakih 30 minuta. To je

ostvareno na način da se prilikom svakog buđenja povećava varijabla te kada se mikrokontroler probudi šesti puta, onda se podaci mjere i šalju pa se i varijabla postavlja na 0. Ovo je napravljeno kako bi se mogla dodati mogućnost mijenjanja perioda slanja promjenom te varijable putem interneta, ali to nije bilo dio zadatka ovoga rada i nije implementirano u radu.

Na slici 5.3. prikazan je dijagram izvođenja programskog koda te je na njemu vidljivo u kojem trenutku se što radi, odnosno kada se pozivaju funkcije za pojedini senzor.



Slika 5.3. Dijagram izvođenja programskog koda.

5.3. Programski kod za SARA N210 komunikacijski modul

Komunikacijski modul SARA N210 služi za slanje podataka na internet putem NB-IoT komuniakcije koja predstavlja novitet u području interneta stvari. Komunikacijski modul s mikrokontrolerom je povezan putem asinkrone serijske komunikacije te je na mikrokontroleru postavljena serijska komunikacija na brzini od 9600 bauda s prekidnom rutinom na liniji za primanje podataka. Prekidna rutina omogućuje da se, neovisno o procesu koji obavlja mikrokontroler, prime podaci s komunikacijskog modula te spreme u memoriju.

U tablici 5.1. opisane su sve potrebne AT naredbe koje se koriste za spajanje modula na mrežu i slanje podataka te postavljanje osnovnih funkcionalnosti modula.

Tablica 5.1. AT naredbe za komunikacijski modul.

AT+CGATT?	Čita status registracije na mrežu
AT+CFUN=1	Postavlja uređaj u potpunu funkcionalnost
AT+CFUN=?	Provjera postavljene funkcionalnosti uređaja
AT+COPS=0	Postavlja automatsko registriranje uređaja na dostupnu mrežu
AT+COPS=?	Provjera mreže na koju je spojen uređaj
AT+NSOCR="DGRAM",protokol,lokalni_port,0	Kreiranje pristupne točke
AT+NSOST=pristupna_točka,"IP_adresa_hosta", pristupni_port,veličina_podataka,"podaci"	Slanje UDP datagrama
AT+NSOCL=pristupna_točka	Zatvaranje pristupne točke
AT+NPSMR?	Provjerava je li uređaj u normalnom načinu rada ili u načinu za uštedu energije
AT+CPSMS=1,,,"01000011","01000011"	Postavljanje uređaja u način za uštedu energije

5.4. Programski kod za senzore koncentracije NO2 i CO plina

Senzori koncentracije NO2 i CO plina uz korištenje modula, koji su opisani u poglavlju 4.2., na izlazu daju napon koji se mjeri korištenjem ADC-a mikrokontrolera. Senzori se razlikuju po tome što za jednaku koncentraciju plina ne daju istu struju odnosno napon na izlazu te su napravljene odvojene funkcije za računanje za svaki senzor. Iz razloga što strujno naponski konverter za referencu koristi napon od 1.8V od njegovog napona se oduzima taj referentni napon koji se mjeri, a u formulama je označen sa *refVolt* . Formula (5-1) opisuje ovisnost izlaznog napona o koncentracije NO2 plina, dok formula (5-2) opisuje ovisnost izlaznog napona o koncentraciji CO plina.

$$\boldsymbol{NO2_{ppm}} = \frac{(|\boldsymbol{U - refVolt}|)}{\mathbf{60.00}} \qquad \textit{(5-1)}$$

$$\boldsymbol{CO_{ppm}} = \frac{(|\boldsymbol{U - refVolt}|)}{\mathbf{1.800}} \qquad \textit{(5-2)}$$

Na slici 5.4. prikazana je funkcija za mjerenje i računanje ppm-a za senzor koncentracije NO2 plina. Vidljivo je da se u funkciji prvo mjeri i računa referentni napon, a nakon toga se mjeri i računa vrijednost ppm-a koristeći formulu (5-1).

```
1.  static void GET_NO2_gas() {
2.      int32_t NO2_adc;
3.      int32_t NO2_volt;
4.      int32_t ref_adc;
5.      int32_t ref_volt;
6.      ref_adc = analogRead(6);
7.      ref_volt = ref_adc * (float) (bat_voltage / 409.500);
8.      NO2_adc = analogRead(4);
9.      NO2_volt = NO2_adc * (float) (bat_voltage / 409.500);
10.     NO2_ppm = (abs(NO2_volt - (ref_volt)) / 60.000);
11. }
```

Slika 5.4. Prikaz funkcije za NO2 senzor.

Na slici 5.5. prikazana je funkcija za mjerenje i izračun ppm-a za senzor koncentracije CO plina.

```
1.  static void GET_CO_gas() {
2.      int32_t CO_adc;
3.      int32_t CO_volt;
4.      int32_t ref_adc;
5.      int32_t ref_volt;
6.      ref_adc = analogRead(6);
7.      ref_volt = ref_adc * (float) (bat_voltage / 409.500);
8.      CO_adc = analogRead(5);
9.      CO_volt = CO_adc * (float) (bat_voltage / 409.500);
10.     CO_ppm = (abs(CO_volt - (ref_volt)) / 1.800);
11. }
```

Slika 5.5. Prikaz funkcije za CO senzor.

5.5. Programski kod za PMS7003 senzor PM čestica

PMS7003 senzor PM čestica kada se uključi svake sekunde automatski napravi mjerenje i pošalje podatke putem serijske komunikacije. Podaci se šalju uvijek istim redoslijedom te se na taj način mogu lako parsirati u odvojene varijable za svaki podatak. Za primanje podataka se koristi prekidna rutina za serijsku komunikaciju koja čeka podatke te kada dođe prvi podatak sprema ih u spremnik koji je veličine 65 znakova tako da u njega stane dva podatka sa senzora. spremnik je kružni što znači da će se stari podaci prepisati novima. Ovaj način osigurava da ako je senzor uključen duže od 3 sekunde dobijemo novije podatke, a ne stare koji su prvi izmjereni. Na slici 5.6. prikazana je funkcija *PMS_Read_data* kojom se iz spremnika parsiraju podaci u strukturu *PMS_data* koja u sebi sadrži sve varijable koje očitava PMS senzor.

```
1.  uint8_t PMS_Read_data() {
2.      uint8_t count = 0;
3.      if (bufferHead == bufferTail) {
4.          return -1;
5.      } else {
6.          while (count < 35) {
7.              if (rx_buffPMS[bufferTail + 1] == 0x4d
8.                      && rx_buffPMS[bufferTail] == 0x42) {
9.                  PMS_data.start1 = rx_buffPMS[bufferTail];
10.                 PMS_data.start2 = rx_buffPMS[bufferTail + 1];
11.                 PMS_data.frame_length = ((rx_buffPMS[bufferTail + 2] << 8)
12.                         + (rx_buffPMS[bufferTail + 3]));
13.                 PMS_data.pm_1_0 = ((rx_buffPMS[bufferTail + 4] << 8)
14.                         + (rx_buffPMS[bufferTail + 5]));
15.                 PMS_data.pm_2_5 = ((rx_buffPMS[bufferTail + 6] << 8)
16.                         + (rx_buffPMS[bufferTail + 7]));
17.                 PMS_data.pm_10_0 = ((rx_buffPMS[bufferTail + 8] << 8)
18.                         + (rx_buffPMS[bufferTail + 9]));
19.                 bufferHead = 0;
20.                 bufferTail = 0;
21.                 return 0;
22.             }
23.             bufferTail = (bufferTail + 1) % RX_BUFFER_PMS_SIZE;
24.             count++;
25.         }
26.         return -1;
27.     }
28. }
```

Slika 5.6. Prikaz funkcije za PMS7003 senzor.

5.6. Programski kod za BME680 senzor

BME680 je senzor temperature, vlage i tlaka zraka te je za njega napisana biblioteka koja olakšava podešavanje senzora i dohvaćanje podataka iz registara senzora. Kako senzor koristi I2C komunikaciju za povezivanje sa mikrokontrolerom, postavljena je I2C komunikacija na brzini od 100 kHz što je standardna brzina. Biblioteka za ovaj senzor je razdvojena na dva dijela što je

standardna praksa te je napravljena zaglavna datoteka i C datoteka koje su dane u prilozima P.5.1. i P.5.2.

U zaglavnoj datoteci se nalaze adrese registara senzora, maske za čitanje i pisanje određenih bitova u registrima te prototipi korištenih funkcija i varijabli što je prikazano na slici 5.7.

```
#define BME680_RESET        0xE0
#define BME680_RESET_CODE   0xB6
#define BME680_ID           0xD0
#define BME680_CONFIG       0x75
#define BME680_CTRL_MEAS    0x74
#define BME680_CTRL_HUM     0x72
#define BME680_CTRL_GAS_1   0x71
#define BME680_CTRL_GAS_0   0x70

static uint8_t m_i2cAddress;    //i2c address for sensor
static uint8_t OS_temp; //oversampling value for temperature
static uint8_t OS_hum;  //oversampling value for humidity
static uint8_t OS_press;    //oversampling value for pressure
static uint8_t heater_state;    //state of heater ( on/off)
static I2C_HandleTypeDef m_i2c_handle;

static int32_t m_temperature;   //variable for temperature
static int32_t m_humidity;  //variable for humidity
static int32_t m_pressure;  //variable for pressure
static int32_t m_tfine;
static int8_t par_h3, par_h4, par_h5, par_h7, par_t3, par_p3, par_p6, par_p7;
static int16_t par_t2, par_p2, par_p4, par_p5, par_p8, par_p9;
```

Slika 5.7. Prikaz zaglavne datoteke biblioteke BME680 senzora.

U C datoteci se nalaze definicije funkcija za čitanje podataka i zapis podataka u registre senzora putem I2C komunikacije. Osim njih napisane su i funkcije za inicijalizaciju senzora te postavljanje načina rada, filtara te izračun kalibracijskih parametara. Također, tu su i funkcije s formulama za računanje temperature, vlage i tlaka zraka koje su dobivene iz dokumentacije senzora [11], kao i adrese pojedinih registara.

5.7. Programski kod za mjerenje stanja baterije

Kako je prethodno opisano, uređaj radi na bateriji koja nije punjiva te je potrebno mjeriti njeno stanje i taj podatak slati zajedno s ostalima kako bi korisnik znao kada treba zamijeniti bateriju na uređaju da ne bi došlo do prestanka rada uređaja. Stanje baterije predstavljeno je naponom na bateriji koji je jednostavno izmjeriti, a iz njega se može odrediti kada je baterija pri kraju svoga životnog vijeka. Koristeći ADC mjeri se napon napajanja uređaja što predstavlja napon na bateriji jer se uređaj napaja samo s baterije. Kako bi napon bio što točniji, koristi se kalibracijski parametar koji je spremljen u memoriji mikrokontrolera te se napon na bateriji dobije prema formuli (5-3).

$$\boldsymbol{VDA = 3000 * VREF_{CAL}/V_{COUNT}} \qquad \textit{(5-3)}$$

Prema formuli (5-3), $VREF_{CAL}$ predstavlja kalibracijski parametar koji je zapisan u memoriju prilikom proizvodnje mikrokontrolera. Kako je za kalibraciju korišten napon od 3 volta, koristi se konstanta 3000 koja predstavlja taj napon u milivoltima. Rezultat mjerenja napona s ADC-om spremljen je u varijablu V_{COUNT}, a može imati vrijednosti od 0 do 4096 za maksimalni napon jer mikrokontroler ima 12-bitni ADC. Zbog veće preciznosti, napon baterije računa se u milivoltima te se sprema u varijablu *VDA*.

Na slici 5.8. prikazana je funkcija koja, koristeći ADC, mjeri napon baterije i računa rezultat prema opisanoj formuli (5-3). Vidi se da se na početku iz memorije čita kalibracijski parametar te se onda izvodi mjerenje ADC-om pozivanjem funkcije *analogRead*, a zatim računa napon napajanja.

```
1.  static void GET_BATTERY_Voltage() {
2.     uint32_t adc_volt = 0;
3.     adc_volt = analogRead(11);
4.     bat_voltage = 3000 * (*VREFINT_CAL_ADDR) / adc_volt;
5.  }
```

Slika 5.8. Prikaz funkcije za računanje napona baterije.

5.8. Programski kod za mjerač vremena

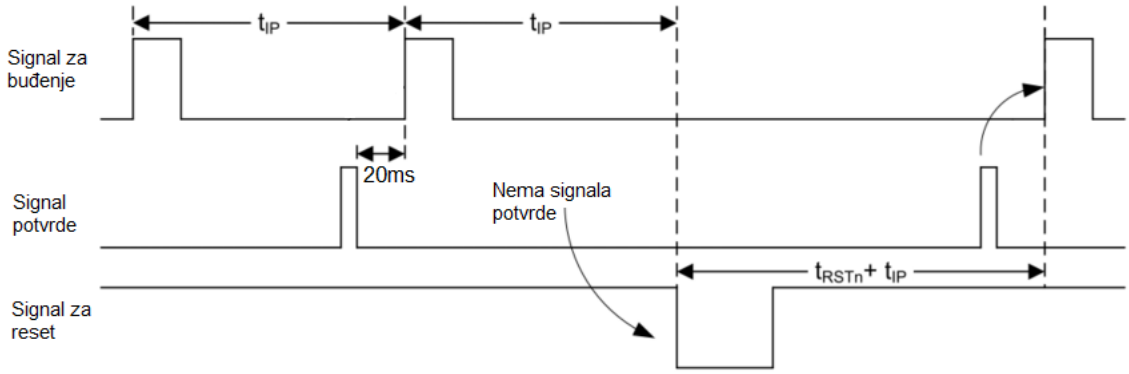
Mjerač vremena TPL5010 zbog svoje jednostavnosti i preciznosti je najbolji izbor za uređaj koji je izrađen u sklopu ovoga diplomskog rada. Kako bi mjerač vremena mogao probuditi mikrokontroler iz spavanja, korištena je mogućnost postavljanja pina za buđenje koju mikrokontroler ima. Nakon što se mikrokontroler probudi, potrebno je poslati signal kako bi mjerač vremena znao da je uspješno probudio mikrokontroler. Na slici 5.9. prikazana je funkcija koja služi za slanje toga signala te se iz nje vidi da je to zapravo impuls u trajanju od 100 milisekundi.

```
1.  static void TIMER_DONE() {
2.     HAL_GPIO_WritePin(GPIOA, DONE_Pin, GPIO_PIN_SET);
3.     HAL_Delay(100);
4.     HAL_GPIO_WritePin(GPIOA, DONE_Pin, GPIO_PIN_RESET);
5.     HAL_Delay(100);
6.  }
```

Slika 5.9. Prikaz funkcije za slanje signala o uspješnom buđenju mikrokontrolera.

U slučaju da mjerač vremena ne primi signal za uspješno buđenje mikrokontrolera, 20 milisekundi prije sljedećeg pulsa za buđenje aktivirat će se sigurnosni mehanizam mjerača vremena te će se umjesto aktivacije pina za buđenje aktivirati pin za resetiranje mikrokontrolera.

Na slici 5.10. dan je grafički prikaz svih signala mjerača vremena u normalnom radu kao i pri aktivaciji sigurnosnog mehanizma. Period između signala za buđenje označen je sa t_{IP}, a t_{RSTn} predstavlja širinu impulsa za resetiranje mikrokontrolera koji iznosi 320 milisekundi. Također, vidljivo je i djelovanje sigurnosnog mehanizma kada je propušten signal potvrde buđenja mikrokontrolera, a nakon toga i nastavak normalnog rada mjerača vremena.

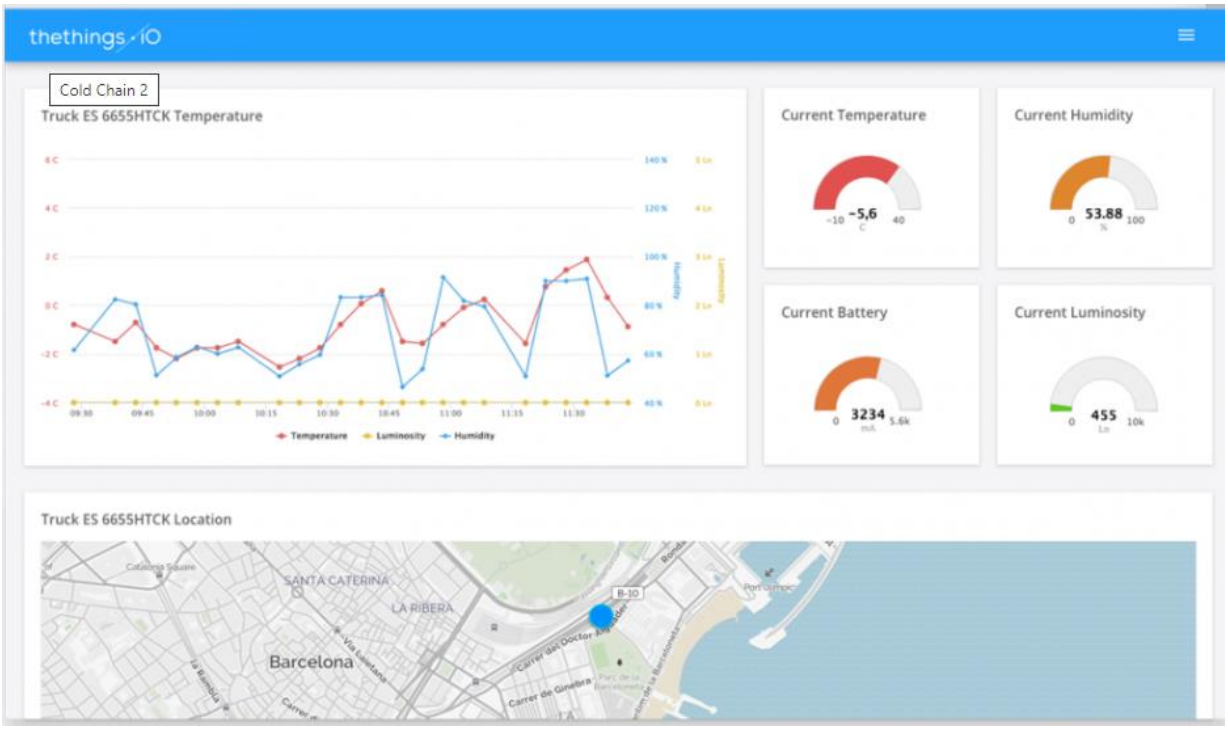


Slika 5.10. Prikaz vremenskog dijagrama signala za mjerač vremena.

5.9. The things.io platforma

Za spremanje i obradu podataka poslanih s uređaja korištena je the things.io platforma koja podržava razne mrežne protokole za primanje podataka, njihovu obradu i grafički prikaz na nadzornoj ploči. Za primanje podataka platforma nudi nekoliko protokola kao što su HTTP, MQTT, CoAP i UDP, a preko HTTP protokola pruža integraciju s drugim platformama. Također, postoji opcija dodavanja vlastitoga algoritma u vidu funkcija, okidača, poslova ili API-a pomoću kojih se može slati SMS poruke, e-mail poruke, kreirati pozive ili objavljivati podatke na društvenim mrežama. Za korištenje platforme potrebno se je registrirati, a nakon toga dodati novi proizvod u kojemu se mogu dodavati uređaji koji šalju ili primaju podatke. Besplatna verzija omogućuje dodavanje raznih grafova na nadzornu ploču za parametre koji se mjere, mijenjanje tipa grafa koji se prikazuje te prilagođavanje podacima, dodavanje naziva osi, naslova i ostalih grafičkih detalja. U postavkama za svaki graf moguće je postaviti limit podataka koji se prikazuju kao i postavljanje praga za podatke koji se prikazuju. Na nadzornoj ploči nije moguće preuzeti sve podatke odjednom, ali moguće je preuzeti podatke za svaki graf posebno što omogućuje daljnju analizu podataka u drugim alatima. Također, u opciji koja se plaća moguće je personalizirati korisničko sučelje prema želji korisnika, preuzeti sve podatke odjednom, koristiti nadogradnju programske podrške putem mreže (OTA) te razne druge mogućnosti koje nisu dostupne u besplatnoj verziji. Osim primanja podataka, platforma omogućava drugim uređajima dohvaćanje podataka kao i upravljanje uređajima putem interneta, ali u sklopu diplomskog rada se koristi samo

za primanje i prikaz mjerenih podataka. Primjer nadzorne ploče sa platforme prikazan je na slici 5.11. gdje je vidljivo na koji način se mogu prikazati podaci korištenjem raznih oblika grafova kao i primjer korištenja karte za prikaz lokacije.



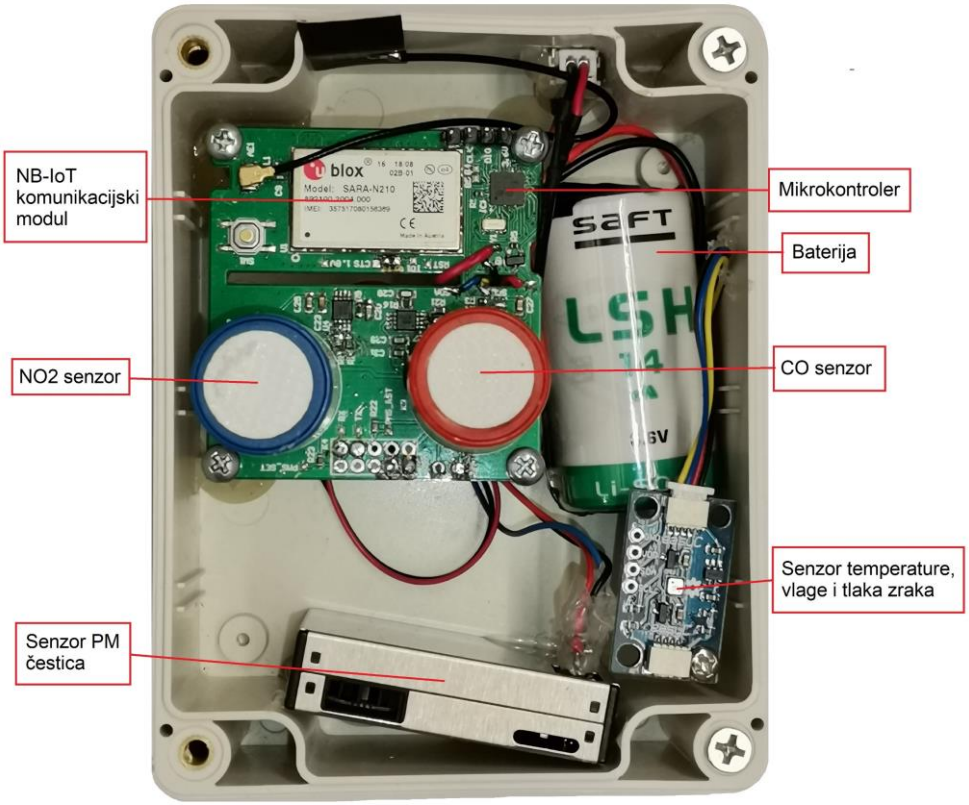
Slika 5.11. Izgled nadzorne ploče na the things.io platformi.

Uređaj izrađen u sklopu ovog diplomskog rada izmjerene podatke pretvara u heksadecimalni format te ih sprema u jednu varijablu i šalje putem UDP mrežnog protokola na the things.io platformu. Na platformi je nužno nakon primanja UDP paketa napraviti razdvajanje podataka i vraćanje u decimalni format kako bi se mogli jednostavno prikazati na dijagramima. Za pretvorbu i razdvajanje primljenih paketa, napravljena je funkcija u JavaScript programskom jeziku koja je dana u prilogu P.5.3. Nakon pretvorbe podataka napravljena je nadzorna ploča s grafovima za sve mjerne podatke te je postavljeno da se prikazuje zadnjih 50 podataka što omogućava prikaz podataka za jedan dan ako se šalju svakih 30 minuta.

6. TESTIRANJE RADA UREĐAJA

Nakon izrade uređaja pojedinačno su testirani svi moduli i senzori kako bi se dokazalo da je tiskana pločica ispravna i da je sve dobro dizajnirano i zalemljeno. Prvo je pločica vizualno pregledana te nakon toga ispitana multimetrom te je utvrđeno da nema kratkih spojeva na pločici. Drugi korak je testiranje mikrokontrolera na način da je prebačen testni programski kod za paljenje jednog od slobodnih pinova na koji je bio spojen multimetar. Kako prebacivanje programskog koda nije uspjelo, čip mikrokontrolera je skinut, očišćen i ponovno vraćen te je nakon toga uspjelo prebacivanje testnog programskog koda koji je dobro radio.

Nakon testiranja mikrokontrolera testirani su i ostali senzori i moduli, a s obzirom da je pojedinačno testiranje svakog modula prošlo uspješno, na mikrokontroler je prebačen programski kod opisan u ovome radu. Prilikom testnog rada utvrđeno je da se podaci sa senzora PM čestica ne prime uvijek te je nakon ponovnog testiranja toga senzora utvrđeno da treba dodati pauzu od 2.4 sekunde, do čega se došlo eksperimentalno, prije mjerenja kako bi osigurali da u spremniku uvijek ima barem 2 podatka sa senzora. Kada je utvrđeno da uređaj dobro mjeri i šalje sve podatke, postavljen je u kutiju koja osigurava lakše prenošenje uređaja tijekom testiranja. Izgled gotovog uređaja prikazan je na slici 6.1. te su označeni svi dijelovi koji su opisani u poglavlju 4.1.



Slika 6.1. Prikaz gotovog uređaja s označenim komponentama.

Za testiranje rada uređaja provedena su dva testa, jedan unutar prostorije na laboratorijskom napajanju kako bi se izmjerila struja te drugi kada je uređaj radio na bateriji u vanjskim uvjetima. Za vrijeme prvoga testa uređaj je radio 5 dana u dva intervala i slao podatke svakih 30 minuta. Prvi interval bio je 12 sati te je za to vrijeme izmjerena struja kako bi se odredila potrošnja uređaja, dok je za vrijeme drugog intervala praćeno slanje podataka kako bi se potvrdilo da uređaj šalje podatke u točnim vremenskim intervalima kako je određeno mjeračem vremena.

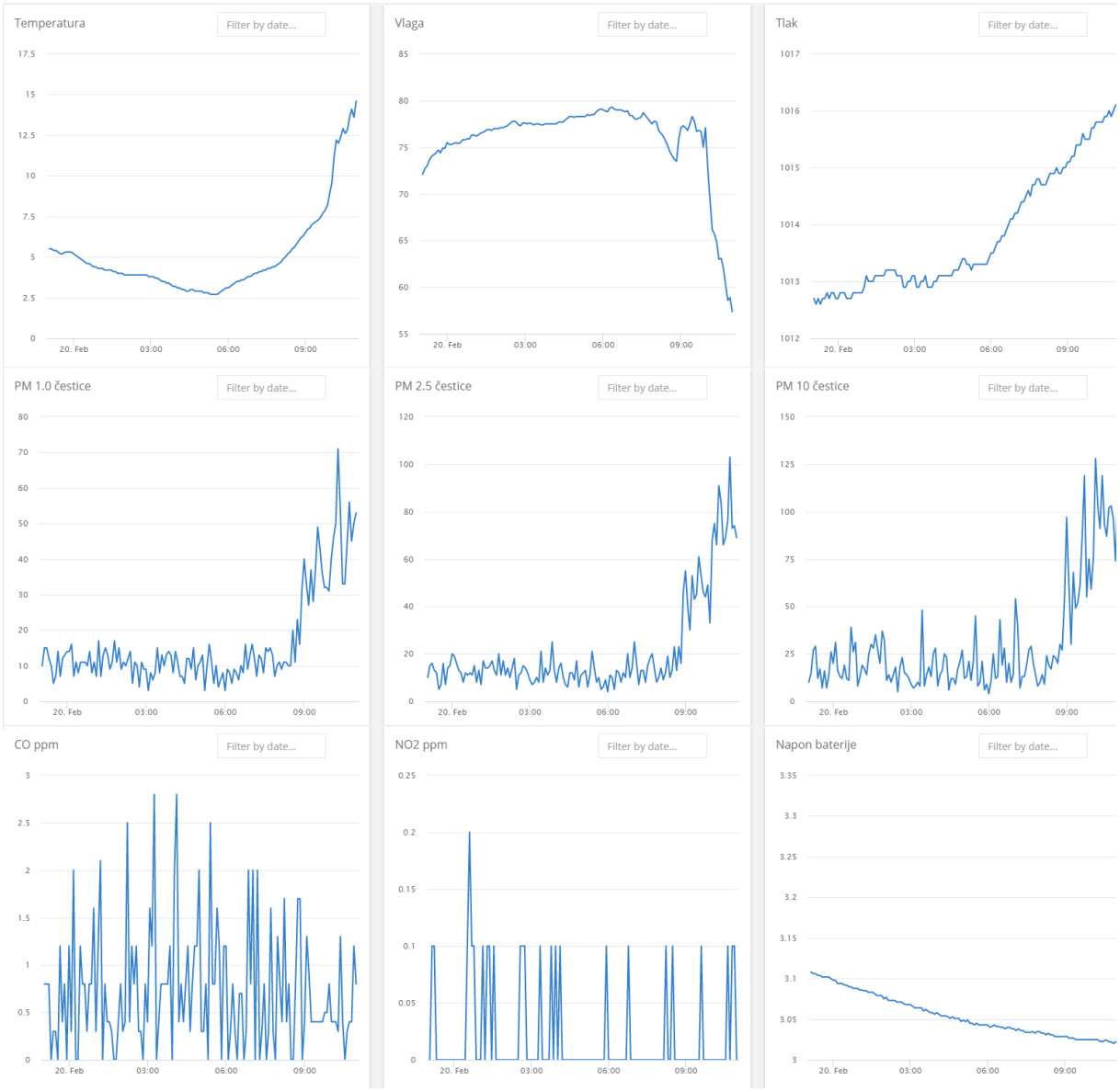
Za određivanje potrošnje, izmjerena je struja tijekom rada uređaja te je dobivena prosječna vrijednost od 50mA, a vremenski period koji je potreban uređaju da izmjeri i pošalje podatke te ode u spavanje je 20 sekundi. Duže aktivno vrijeme je iz razloga što komunikacijski modul ostaje aktivan određeni vremenski period koji je određen od strane operatera te je u tome periodu omogućeno primanje podataka. Iako uređaj ne prima nikakve podatke s mreže niti se kontrolira putem iste, nije moguće smanjiti vremenski period jer trenutni operater ne prihvaća izmjenu toga vremenskog perioda. Uređaj u spavanju ima struju od 5uA, a vremenski period ovisi o postavljenom intervalu mjerenja podataka koji je varijabilan u koracima od 5 minuta i postavlja ga korisnik. U tablici 6.1. prikazana je potrošnja uređaja ovisno o učestalosti mjerenja i slanja podataka kao i potreban kapacitet baterije s kojom bi uređaj mogao raditi određeni vremenski period.

Tablica 6.1. Potrošnja uređaja pri različitim intervalima mjerenja.

Period mjerenja	*Prosječna potrošnja (mA)*	*Kapacitet baterije (mAh) za rad u određenom vremenskom periodu*		
		1 godina	*2 godine*	*5 godina*
5min	*4.55*	*27440*	*54880*	*137200*
15min	*1.09*	*9600*	*19200*	*48000*
30min	*0.554*	*4860*	*9720*	*24300*
1h	*0.281*	*2465*	*4930*	*12325*
2h	*0.1435*	*1258*	*2515*	*6290*

Kako je struja uređaja uspješno izmjerena i uređaj je slao podatke u postavljenim vremenskim intervalima, završeno je prvo testiranje te je uređaj prebačen s laboratorijskog napajanja na bateriju i postavljen je izvan prostorije. Za vrijeme drugoga testa u vanjskim uvjetima uređaj je mjerio i slao podatke svakih 5 minuta kako bi se prikupilo više podataka. Podaci s uređaja su uspoređeni s podacima koje daje lokalna mjerna postaja državnog hidrometeorološkog zavoda. Uspoređivani su podaci za NO2 i CO plin te temperatura, vlaga i tlak zraka. Nakon analize podataka s uređaja i mjerne postaje, utvrđeno je da podaci za koncentraciju plinova odstupaju u iznosu od 2 posto od

podataka mjerne postaje, dok podaci za temperaturu, vlagu i tlak zraka imaju odstupanje manje od 1 posto što je zadovoljavajuće. Na slici 6.2. prikazani su grafovi za svaki mjereni podatak koji je dobiven tijekom testiranja u vanjskim uvjetima, a prikazani su svi podaci prikupljeni tijekom 12 sati mjerenja podataka, odnosno testiranja uređaja.



Slika 6.2. Prikaz podataka dobivenih tijekom testiranja.

7. ZAKLJUČAK

Cilj rada je izrada uređaja male potrošnje za nadzor kvalitete zraka koji izmjerene podatke šalje u oblak koristeći NB-IoT komunikaciju. Korištenjem NB-IoT komunikacije osigurana je mala potrošnja prilikom slanja podataka jer se NB-IoT modul ne spaja svaki puta na mrežu, nego ostaje spojen te se tako uvelike smanjuje aktivno vrijeme uređaja u odnosu na druge komunikacije poput 2G, 3G, Wi-Fi ili Bluetooth. Zbog smanjenja potrošnje u spavanju, odabrana je STM32L serija mikrokontrolera koja se odlikuje niskom potrošnjom od svega 0.3uA u spavanju.

Nakon što su odabrani potrebni senzori, napravljeno je sklopovlje uređaja, odnosno tiskana pločica na kojoj se nalaze mikrokontroler, komunikacijski modul i svi senzori. Programska podrška je napravljena za svaki senzor posebno te je na kraju sve spojeno u jednu cjelinu. Testiranje uređaja dalo je zadovoljavajuće rezultate te su, uz manje prilagodbe programske podrške, dobiveni očekivani rezultati za sve mjerene parametre.

U svrhu poboljšanja izrađenog uređaja bilo bi potrebno, korištenjem točnijeg uređaja za mjerenje NO2 i CO plina, umjeriti uređaj te točnije odrediti konstante u formulama za izračun ppm koncentracije plinova. Također, tijekom rada je primijećeno da mjerenja imaju određeni šum jer se radi o analognim senzorima te se koriste analogni signali koje bi bilo potrebno boljim dizajnom sklopovlja zaštititi od utjecaja šumova i smetnji kako bi dobili točnije rezultate.

Za sve senzore, koristeći bolju mjernu opremu, mogla bi se odrediti ovisnost mjerenja o temperaturi te u programsku podršku dodati korekcijske funkcije za mjerenja koja su ovisna o temperaturi. Za računanje ovisnosti o temperaturi bilo bi potrebno odabrati i bolji senzor za temperaturu jer trenutno odabrani ima spor odziv te ne može izmjeriti nagle promjene temperature. Uz optimizaciju sklopovlja dodavanjem senzora za druge plinove i izmjenom senzora za temperaturu, vlagu i tlak zraka, uređaj bi imao puno šire područje primjene. Također, optimizacijom programske podrške moglo bi se skratiti vrijeme rada uređaja, a ujedno i njegova potrošnja. Kako je ranije opisano, komunikacijski modul ostaje najdulje budan stoga bi trebalo odabrati drugog operatera koji omogućuje izmjenu vremenskog perioda aktivnosti komunikacijskog modula nakon slanja podataka. Ako bi se navedeni vremenski brojač potpuno ugasio, komunikacijski modul bi odmah nakon slanja podataka otišao u spavanje što bi uvelike smanjilo potrošnju te bi aktivno vrijeme bilo 5 sekundi što bi omogućilo rad uređaja na bateriji istog kapaciteta duži vremenski period za više od 50 posto.

Kako u sklopu rada nije predviđena izrada kućišta uređaj je stavljen u kutiju kako bi se mogao testirati što je opisano u poglavlju 6. Za korištenje uređaja na otvorenome prostoru trebalo bi

izraditi kućište koje mora osigurati zaštitu od vode zbog zaštite uređaja od padalina i drugih vremenskih utjecaja. Uz to treba osigurati i dovoljan protok zraka odnosno njegovu izmjenu kako bi se mogli mjeriti opisani parametri kvalitete zraka. Protok zraka potrebno je osigurati prirodnim strujanjem koje bi se postiglo određenim otvorima na kućištu čiju poziciju je potrebno računski odrediti, a nakon izrade i testirati. U slučaju da dovoljan protok zraka nije moguće osigurati prirodnim strujanjem, mogao bi se dodati ventilator koji bi se palio kratko prije senzora, ali time bi se znatno povećala potrošnja uređaja i smanjilo vrijeme rada na bateriji.

Daljnjim razvojem uređaja potrebno je ispraviti uočene nedostatke te dizajnirati kućište kako je opisano. Za povećanje autonomnosti uređaja moguće je trenutno korištenu nepunjivu bateriju zamijeniti s punjivom baterijom i dodati fotonaponski modul za punjenje baterije. Dodavanjem fotonaponskog modula bio bi potrebno dizajnirati MPPT punjač kako bi se iskoristila sva dostupna energija dobivena fotonaponskim modulom te naponski regulator ako je napon punjive baterije veći od 3.6 volti. Korištenjem punjive baterije otvorila bi se mogućnost postavljanja ventilatora na kućište kako je ranije opisano što bi pojednostavilo dizajn kućišta jer bi se izbjegla potreba za prirodnim strujanjem zraka.

LITERATURA

[1] S. Brienza, A. Galli, G. Anastasi, P. Bruschi, „A Low-Cost Sensing System for Cooperative Air Quality Monitoring in Urban Areas“, Sensors, 15(6), str. 12242-12259, 5.2015., dostupno na: https://doi.org/10.3390/s150612242 [12.9.2020.]

[2] S. Kaivonen, E. Ngai, „Real-time air pollution monitoring with sensors on city bus“, Digital Communications and Networks, 6(1), str. 23-30, 1.2020., dostupno na: https://doi.org/10.1016/j.dcan.2019.03.003 [11.9.2020.]

[3] X. Liu, B. Li, A. Jiang, S. Qi, C. Xiang, N. Xu, „A bicycle-borne sensor for monitoring air pollution near roadways“, Consumer electronics - Taiwan, 7216835, str. 166-167, Taiwan, 2015.

[4] D. Hasenfratz, O. Saukh, S. Sturzenegger, L. Thiele, „Participatory air pollution monitoring using smartphones“, Mobile Sensing: From Smartphones and Wearables to Big Data (2012), dostupno na: https://www.researchgate.net/publication/267963506_Participatory_Air_Pollution_Monitoring_Using_Smartphones [12.9.2020.]

[5] T. Wang, W. Han, M. Zhang, X. Yao, L. Zhang, X. Peng, C. Li, X. Dan, „Unmanned Aerial Vehicle-Borne Sensor for Atmosphere-Particulate-Matter Measurements: Design and Experiments“, Sensors, 20(1), 57, 12.2019., dostupno na: https://doi.org/10.3390/s20010057 [16.1.2021.]

[6] L. Sun, K. C. Wong, P. Wei, S. Ye, H. Huang, F. Yang, D. Westerdahl, P. K. K. Louie, C W. Y. Luk, Z. Ning, „Development and Application of a Next Generation Air Sensor Neetwork for the Hong Kong Marathon 2015 Air Quality Monitoring“, Sensors, 16(2), 211, 2.2016., dostupno na: https://doi.org/10.3390/s16020211 [16.1.2021.]

[7] M. Benammar, A. Abdaoui, S. H. M. Ahmad, F. Touati, A. Kadri, „A Modular IoT Platform for Real-Time Indoor Air Quality Monitoring“, Sensors 18(2), 581, 2.2018., dostupno na: https://doi.org/10.3390/s18020581 [16.1.2021.]

[8] https://sgx.cdistore.com/datasheets/sgx/ds-0228%20(sgx-4no2)%20v1.pdf [10.1.2021.]

[9] https://www.sgxsensortech.com/content/uploads/2014/07/DS-0138-SGX-4CO-V2.pdf [10.1.2021.]

[10] https://www.sgxsensortech.com/content/uploads/2014/08/A1A-EC_SENSORS_AN2-Design-of-Electronics-for-EC-Sensors-V4.pdf [10.1.2021.]

[11] https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme680-ds001.pdf [15.1.2021.]

[12] https://download.kamami.com/p564008-p564008-PMS7003%20series%20data%20manua_English_V2.5.pdf [15.1.2021.]

[13] https://www.st.com/resource/en/datasheet/stm32l071cb.pdf [15.1.2021.]

[14] https://www.u-blox.com/sites/default/files/SARA-N2_DataSheet_%28UBX-15025564%29.pdf [15.1.2021.]

[15] https://www.ti.com/lit/ds/symlink/tpl5010.pdf?ts=1610894720511&ref_url=https%253A%252F%252Fwww.google.com%252F [15.1.2021.]

SAŽETAK

U diplomskom radu je opisano i izrađeno sklopovlje i programska podrška uređaja male potrošnje za nadzor kvalitete zraka. Kako bi se napravio uređaj za nadzor kvalitete zraka, odabrani su onečišćivači koji najviše utječu na kvalitetu zraka. Za mjerenje NO2 i CO plinova kao najznačajnijih onečišćivača odabrani su elektrokemijski senzori, a uz plinove mjeri se i količina lebdećih čestica. Za senzore, komunikacijski modul i mikrokontroler napravljena je tiskana pločica kako bi se dobio kvalitetniji i bolji uređaj koji je otporniji na vanjske smetnje. Kako bi se podaci sa senzora mogli pročitati, napravljena je programska podrška za uređaj koja upravlja sa svim senzorima i komunikacijskim modulom. Za slanje podataka u oblak odabrana je NB-IoT komunikacija koja je namijenjena za slanje male količine podataka uz malu potrošnju energije. Također su napravljeni i dijagrami u oblaku za svaki podatak te je time olakšano čitanje podataka, a osim čitanja podataka u oblaku je moguće implementirati i slanje podataka u druge baze gdje se podaci dalje mogu obrađivati i analizirati. Nakon izrade uređaj je testiran, a mjereni podaci su uspoređeni s lokalnom mjernom postajom te je potvrđeno da uređaj zadovoljavajuće točno mjeri podatke uz odstupanje od 2 posto za mjerenje koncentracije plinova i manje od 1 posto za ostale parametre.

Ključne riječi: kvaliteta zraka, NB-IoT, elektrokemijski senzor, lebdeće čestice, mala potrošnja

ABSTRACT

Title: Low-power air quality monitoring device

Circuitry and software developed for a low-power air quality monitor is described in this graduate thesis. In order to achieve the best possible readings, pollutants that affect the air quality the most have been selected. For NO2 and CO measurement, electrochemical sensors are used, alongside particulate matter sensor. To accommodate sensors, communication module and microcontroller, custom made PCB is designed and manufactured. That way, the device can endure harsher outside environmental conditions with increased noise immunity. To read data from the sensors, special firmware runs on a microcontroller, which takes care of sensor readings and communications. NB-IoT is used to transmit readings to the cloud, since it's more energy efficient for such small amounts of data. Furthermore, the cloud system includes convenient graphs and the same data is ready to be transferred to another database for further analysis. After prototyping, the device was taken through the testing point where its readings were compared to the local metro

station. Readings were appropriate, with 2% deviation for gas pollutants and 1% deviation for other parameters.

Keywords: air quality, NB-IoT, electrochemical sensor, pollutants, low-power

ŽIVOTOPIS

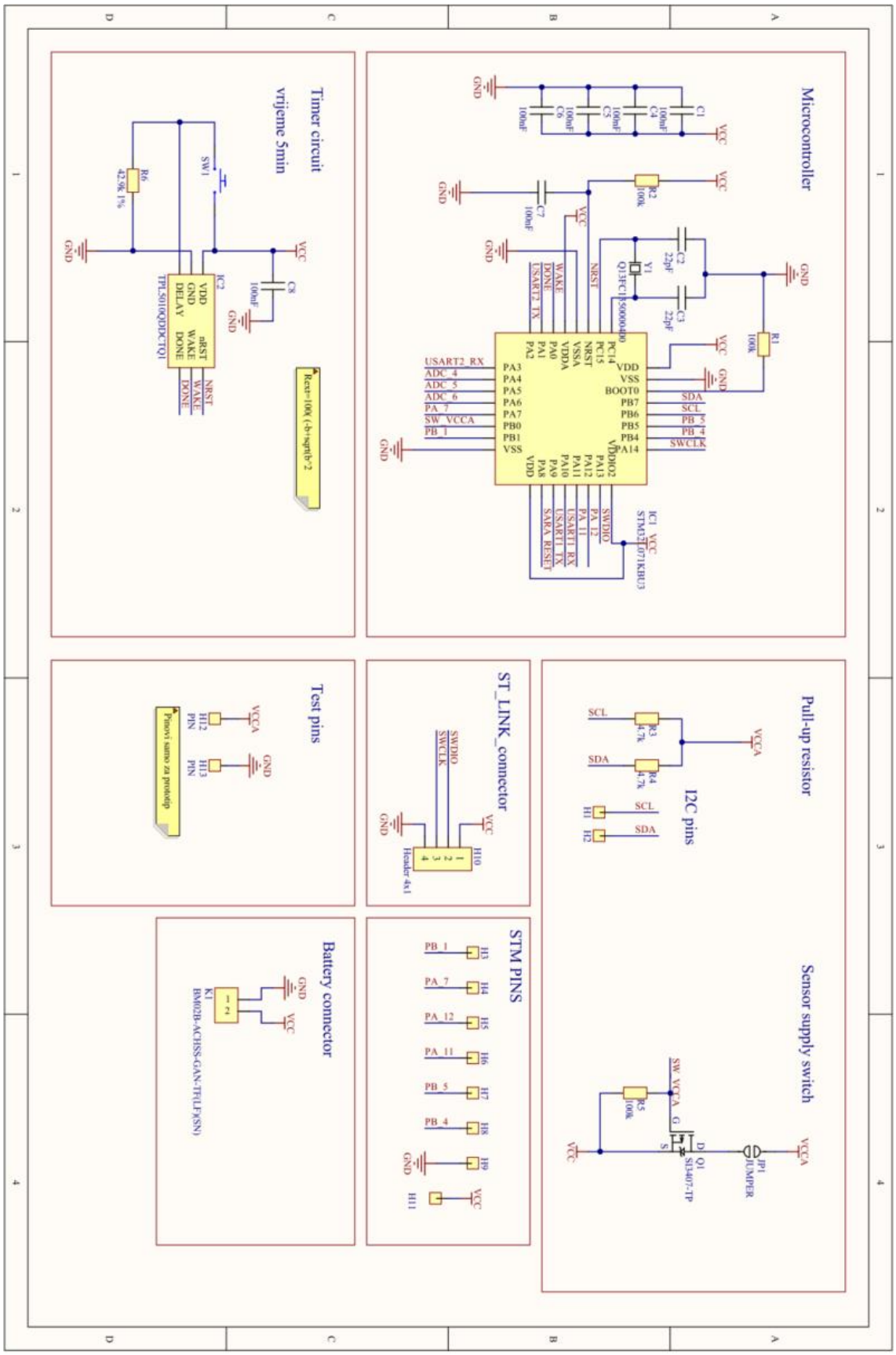
Ivan Fadiga rođen je 7.6.1996. u Osijeku, Hrvatska. Osnovnu školu završio je u Ivanovcima, a nakon toga upisuje elektrotehničku školu u Srednjoj školi Valpovo. Nakon završetka srednjoškolskog obrazovanja upisuje Preddiplomski sveučilišni studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Na drugoj godini opredjeljuje se za smjer Komunikacije i informatika te 2018. završava Preddiplomski sveučilišni studij nakon kojega upisuje Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije.

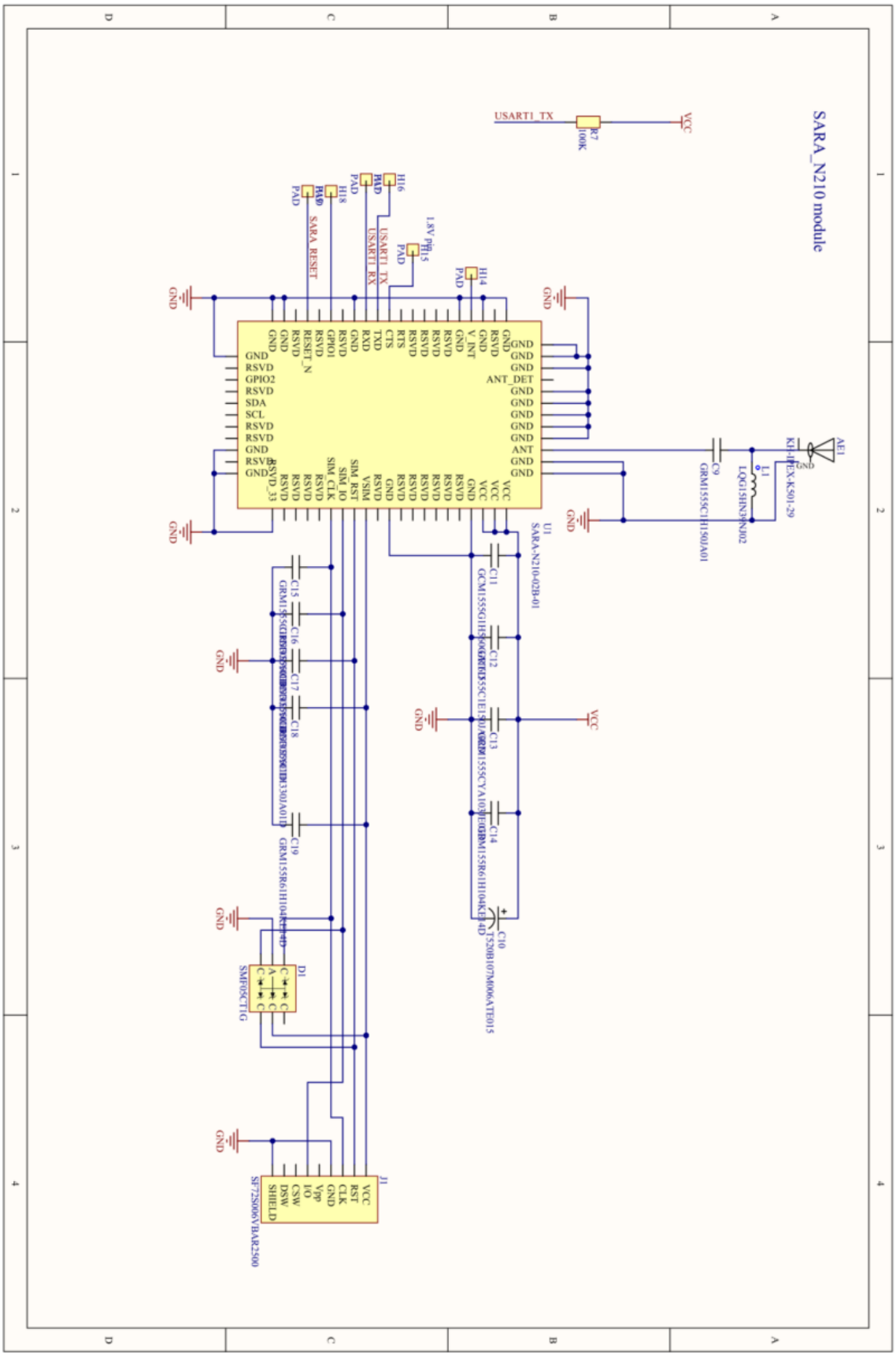
Za vrijeme srednjoškolskog obrazovanja sudjeluje na natjecanju iz matematike „Klokan bez granica“ te na državnom natjecanju iz osnova elektrotehnike.

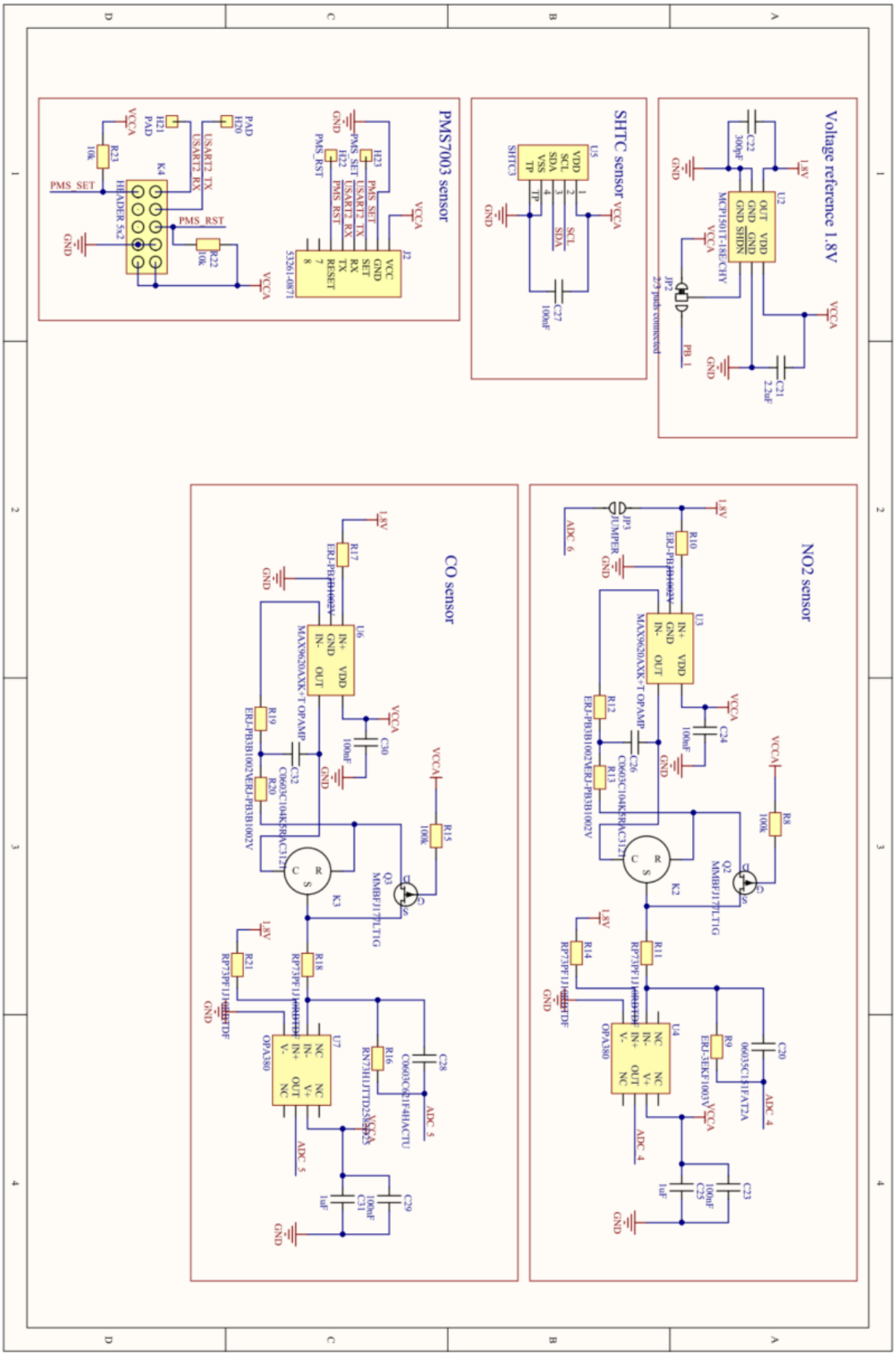
Tijekom studiranja povremeno je radio kao demonstrator na laboratorijskim vježbama iz Osnova elektrotehnike i mjerenja u elektrotehnici. Od 2018. započinje raditi kao student u TAVU d.o.o. gdje obavlja i stručnu praksu te ostaje kao stalno zaposlen.

PRILOZI

P.4.1. Shema uređaja




SARA_N210 module
USART1_TX
R7
100K
VCC
H16
H18
H15
H14
PAD
SARA_RESET
USART1_TX
USART1_RX
1.8V pin
GND
RSVD
GPIO2
SDA
SCL
ANT_DET
ANT
RESET_N
GPIO1
RXD
TXD
CTS
RTS
V_INT
SIM_RST
SIM_IO
SIM_CLK
VSIM
RSVD_33
U1
SARA-N210-02B-01
C9
GRM1555C1H150JA01
L1
LQG15HN39NJ02
AE1
KH-IPEX-K501-29
C10
T520B107M006ATE015
C11
C12
C13
C14
C15
C16
C17
C18
C19
D1
SMF05CT1G
J1
VCC
RST
CLK
GND
Vpp
I/O
CSW
DSW
SHIELD
SF72S006VBAR2500


Voltage reference 1.8V
SHTC sensor
PMS7003 sensor
NO2 sensor
CO sensor
VCCA
GND
1.8V
PMS_SET
PMS_RST
USART2_TX
USART2_RX
ADC_4
ADC_5
ADC_6

P.5.1. Zaglavna datoteka biblioteke za BME680 senzor

```
#ifndef BME_680_H_
#define BME_680_H_

#ifdef __cplusplus
 extern "C" {
#endif

#include "stdint.h"
#include "stm32l0xx_hal.h"
//sensor I2C address
#define BME680_I2C_ADDRESS (0x76<<1)
#define BME680_CONCAT_BYTES(msb, lsb)   (((uint16_t)msb << 8) | (uint16_t)lsb)

#define BME680_DATA_START_ADDRESS 0x1F
#define BME680_DATA_SIZE    8
#define BME680_COEFF_SIZE1  25
#define BME680_COEFF_SIZE2  16
#define BME680_COEFF_START_ADDRESS1 0x89
#define BME680_COEFF_START_ADDRESS2 0xE1
#define BME680_HUM_REG_SHIFT_VAL    4
#define BME680_BIT_H1_DATA_MASK     0x0F
#define BME680_STATUS_REG   0x1D

 //Register for temperature measurement
#define PAR_T1_LSB      8   //coeff_array2
#define PAR_T1_MSB      9   //coeff_array2
#define PAR_T2_LSB      1   //coeff_array1
#define PAR_T2_MSB      2   //coeff_array1
#define PAR_T3          3   //coeff_array1
#define TEMP_ADC_XLSB   5   //data_array
#define TEMP_ADC_LSB    4   //data_array
#define TEMP_ADC_MSB    3   //data_array

//Register for pressure measurement
#define PAR_P1_LSB      5   //coeff_array1
#define PAR_P1_MSB      6   //coeff_array1
#define PAR_P2_LSB      7   //coeff_array1
#define PAR_P2_MSB      8   //coeff_array1
#define PAR_P3          9   //coeff_array1
#define PAR_P4_LSB      11  //coeff_array1
#define PAR_P4_MSB      12  //coeff_array1
#define PAR_P5_LSB      13  //coeff_array1
#define PAR_P5_MSB      14  //coeff_array1
#define PAR_P6          16  //coeff_array1
#define PAR_P7          15  //coeff_array1
#define PAR_P8_LSB      19  //coeff_array1
#define PAR_P8_MSB      20  //coeff_array1
#define PAR_P9_LSB      21  //coeff_array1
#define PAR_P9_MSB      22  //coeff_array1
#define PAR_P10         23  //coeff_array1
#define PRESS_ADC_XLSB  2
#define PRESS_ADC_LSB   1   //data_array
#define PRESS_ADC_MSB   0   //data_array

//Register for humidity measurement
#define PAR_H1_LSB      1   //coeff_array2
#define PAR_H1_MSB      2   //coeff_array2
#define PAR_H2_LSB      1   //coeff_array2
#define PAR_H2_MSB      0   //coeff_array2
#define PAR_H3          3   //coeff_array2
#define PAR_H4          4   //coeff_array2
#define PAR_H5          5   //coeff_array2
#define PAR_H6          6   //coeff_array2
```

```
64. #define PAR_H7          7   //coeff_array2
65. #define HUM_ADC_LSB     7   //data_array
66. #define HUM_ADC_MSB     6   //data_array
67.
68.  /** Sensor configuration and control registers */
69. #define BME680_RESET        0xE0
70. #define BME680_RESET_CODE   0xB6
71. #define BME680_ID           0xD0
72. #define BME680_CONFIG       0x75
73. #define BME680_CTRL_MEAS    0x74
74. #define BME680_CTRL_HUM     0x72
75. #define BME680_CTRL_GAS_1   0x71
76. #define BME680_CTRL_GAS_0   0x70
77.
78.  //Sensor mode control
79. #define BME680_MODE_MASK    0x03
80. #define BME680_SLEEP        0x00
81. #define BME680_FORCED       0x01
82.
83.  //Over sampling control
84. #define BME680_CTRL_HUM_MASK    0x07    //mask for humidity sensor control register
85. #define BME680_CTRL_TEMP_MASK   0xE0    //mask for temperature sensor control register

86. #define BME680_CTRL_PRESS_MASK  0x1C    //mask for pressure sensor control register
87. #define BME680_SENSOR_DISABLE   0x00    //skipped humidity (output set to 0x8000)
88. #define BME680_OS_1         0x01    //over sampling x1
89. #define BME680_OS_2         0x02    //over sampling x2
90. #define BME680_OS_4         0x03    //over sampling x4
91. #define BME680_OS_8         0x04    //over sampling x8
92. #define BME680_OS_16        0x05    //over sampling x16
93.
94.  //Filter control
95. #define BME680_FILTER_MASK  0x1C    //mask for IIR filter
96. #define BME680_FILTER_0     0x00    //IIR filter coefficient 0
97. #define BME680_FILTER_1     0x01    //IIR filter coefficient 1
98. #define BME680_FILTER_3     0x02    //IIR filter coefficient 3
99. #define BME680_FILTER_7     0x03    //IIR filter coefficient 7
100.       #define BME680_FILTER_15    0x04    //IIR filter coefficient 15
101.       #define BME680_FILTER_31    0x05    //IIR filter coefficient 31
102.       #define BME680_FILTER_63    0x06    //IIR filter coefficient 63
103.       #define BME680_FILTER_127   0x07    //IIR filter coefficient 127
104.
105.       //Gas sensor control
106.       #define BME680_CTRL_HEAT_MASK       0x08    //mask for ctrl_gas_0 register for c
    ontrol heater
107.       #define BME680_HEAT_OFF             0x01    //heater off
108.       #define BME680_CTRL_GAS_RUN_MASK    0x10    //mask for ctrl_gas_1 register,contr
    ol gas conversion
109.       #define BME680_RUN_GAS              0x00
110.
111.       // initialization variable
112.       static uint8_t m_i2cAddress;    //i2c address for sensor
113.       static uint8_t OS_temp; //oversampling value for temperature
114.       static uint8_t OS_hum;  //oversampling value for humidity
115.       static uint8_t OS_press;    //oversampling value for pressure
116.       static uint8_t heater_state;    //state of heater ( on/off)
117.       static I2C_HandleTypeDef m_i2c_handle;
118.
119.       static int32_t m_temperature;   //variable for temperature
120.       static int32_t m_humidity;  //variable for humidity
121.       static int32_t m_pressure;  //variable for pressure
122.       static int32_t m_tfine;
123.
124.       //calibration parameters for calculating temp,hum and pressure
125.       static int8_t par_h3, par_h4, par_h5, par_h7, par_t3, par_p3, par_p6, par_p7;
126.       static int16_t par_t2, par_p2, par_p4, par_p5, par_p8, par_p9;
```

```
127.        static uint8_t par_h6, par_p10;
128.        static uint16_t par_h1, par_h2, par_t1,par_p1;
129.
130.        enum FILTER_COEFFICIENT{
131.            COEF0,COEF1,COEF3,COEF7,COEF15,COEF31,COEF63,COEF127
132.        };
133.        enum BME680_MODE{SLEEP,FORCED};
134.        void BME680_INIT (I2C_HandleTypeDef i2c, uint8_t i2cAddress, uint8_t gas);
135.        void BME680_HEATER_DIS ();
136.        void BME680_OS_SET (uint8_t temp, uint8_t hum, uint8_t press);
137.        void BME680_FILTER_SET (enum FILTER_COEFFICIENT val);
138.        static void BME680_MODE_SET (enum BME680_MODE mode);
139.        static void BME680_GET_CALIBRATION();
140.        static void BME680_READ_SENSOR();
141.        void BME680_GET_DATA(int32_t *temp, int32_t *hum, int32_t *press );
142.        void readRegister(I2C_HandleTypeDef I2C, uint8_t ADDRESS, uint8_t reg,uint8_t *d
    ata,uint8_t count);
143.        void writeRegister(I2C_HandleTypeDef I2C, uint8_t ADDRESS, uint8_t reg, uint8_t
    value);
144.
145.        #ifdef __cplusplus
146.        }
147.        #endif
148.        #endif /* BME_680_H_ */
```

P.5.2. C datoteka biblioteke za BME680 senzor

```
#include "BME_680.h"

void writeRegister(I2C_HandleTypeDef I2C, uint8_t ADDRESS, uint8_t reg,
        uint8_t value) {
    uint8_t data[2];
    data[0] = reg;
    data[1] = value;
    HAL_I2C_Master_Transmit(&I2C, ADDRESS, data, 2, 200);
}
void readRegister(I2C_HandleTypeDef I2C, uint8_t ADDRESS, uint8_t reg,
        uint8_t *data, uint8_t count) {

    HAL_I2C_Master_Transmit(&I2C, ADDRESS, ®, 1, 200);
    HAL_I2C_Master_Receive(&I2C, ADDRESS, data, count, 200);
}

void BME680_INIT(I2C_HandleTypeDef i2c, uint8_t i2cAddress, uint8_t gas) {
    m_i2c_handle = i2c;
    m_i2cAddress = i2cAddress;
    BME680_GET_CALIBRATION();
}

void BME680_HEATER_DIS() {
    writeRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_GAS_0,
            (BME680_HEAT_OFF << 3));
}

void BME680_OS_SET(uint8_t temp, uint8_t hum, uint8_t press) {
    switch (temp) {
    case 0:
        OS_temp = BME680_SENSOR_DISABLE;
        break;
    case 1:
        OS_temp = BME680_OS_1;
        break;
    case 2:
        OS_temp = BME680_OS_2;
        break;
    case 4:
        OS_temp = BME680_OS_4;
        break;
    case 8:
        OS_temp = BME680_OS_8;
        break;
    case 16:
        OS_temp = BME680_OS_16;
        break;
    default:
        OS_temp = BME680_SENSOR_DISABLE;
    }
    switch (hum) {
    case 0:
        OS_hum = BME680_SENSOR_DISABLE;
        break;
    case 1:
        OS_hum = BME680_OS_1;
        break;
    case 2:
        OS_hum = BME680_OS_2;
        break;
    case 4:
        OS_hum = BME680_OS_4;
        break;
```

```
64.     case 8:
65.         OS_hum = BME680_OS_8;
66.         break;
67.     case 16:
68.         OS_hum = BME680_OS_16;
69.         break;
70.     default:
71.         OS_hum = BME680_SENSOR_DISABLE;
72.     }
73.     switch (press) {
74.     case 0:
75.         OS_press = BME680_SENSOR_DISABLE;
76.         break;
77.     case 1:
78.         OS_press = BME680_OS_1;
79.         break;
80.     case 2:
81.         OS_press = BME680_OS_2;
82.         break;
83.     case 4:
84.         OS_press = BME680_OS_4;
85.         break;
86.     case 8:
87.         OS_press = BME680_OS_8;
88.         break;
89.     case 16:
90.         OS_press = BME680_OS_16;
91.         break;
92.     default:
93.         OS_press = BME680_SENSOR_DISABLE;
94.     }
95.     uint8_t curReg;
96.     readRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_MEAS, &curReg, 1);
97.     curReg &= (~(BME680_CTRL_TEMP_MASK | BME680_CTRL_PRESS_MASK));
98.     curReg = curReg | (OS_temp << 5) | (OS_press << 2);
99.     writeRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_MEAS, curReg);
100.        readRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_HUM, &curReg, 1);
101.        curReg &= (~BME680_CTRL_HUM_MASK);
102.        curReg |= OS_hum;
103.        writeRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_HUM, curReg);
104.    }
105.
106.    void BME680_FILTER_SET(enum FILTER_COEFFICIENT val) {
107.        uint8_t filter = 0;
108.        switch (val) {
109.        case 0:
110.            filter = BME680_FILTER_0;
111.            break;
112.        case 1:
113.            filter = BME680_FILTER_1;
114.            break;
115.        case 2:
116.            filter = BME680_FILTER_3;
117.            break;
118.        case 3:
119.            filter = BME680_FILTER_7;
120.            break;
121.        case 4:
122.            filter = BME680_FILTER_15;
123.            break;
124.        case 5:
125.            filter = BME680_FILTER_31;
126.            break;
127.        case 6:
128.            filter = BME680_FILTER_63;
129.            break;
```

```
130.            case 7:
131.                filter = BME680_FILTER_127;
132.                break;
133.            }
134.            uint8_t curReg;
135.            readRegister(m_i2c_handle, m_i2cAddress, BME680_CONFIG, &curReg, 1);
136.            curReg &= (~BME680_FILTER_MASK);
137.            curReg |= (filter << 2);
138.            writeRegister(m_i2c_handle, m_i2cAddress, BME680_CONFIG, curReg);
139.        }
140.
141.        static void BME680_MODE_SET(enum BME680_MODE mode) {
142.            uint8_t m_mod = 0;
143.            switch (mode) {
144.            case 0:
145.                m_mod = BME680_SLEEP;
146.                break;
147.            case 1:
148.                m_mod = BME680_FORCED;
149.                break;
150.            }
151.            uint8_t curReg;
152.            readRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_MEAS, &curReg, 1);
153.            curReg &= ~(BME680_MODE_MASK);
154.            curReg = curReg | (m_mod);
155.            writeRegister(m_i2c_handle, m_i2cAddress, BME680_CTRL_MEAS, curReg);
156.        }
157.
158.        static void BME680_GET_CALIBRATION() {
159.            /*  Humidity calibration parameter   */
160.            uint8_t coeff_array1[BME680_COEFF_SIZE1];
161.            uint8_t coeff_array2[BME680_COEFF_SIZE2];
162.            readRegister(m_i2c_handle, m_i2cAddress, BME680_COEFF_START_ADDRESS1,
163.                    coeff_array1, BME680_COEFF_SIZE1);
164.            readRegister(m_i2c_handle, m_i2cAddress, BME680_COEFF_START_ADDRESS2,
165.                    coeff_array2, BME680_COEFF_SIZE2);
166.            par_t1 = (uint16_t) (BME680_CONCAT_BYTES(coeff_array2[PAR_T1_MSB],
167.                    coeff_array2[PAR_T1_LSB]));
168.            par_t2 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_T2_MSB],
169.                    coeff_array1[PAR_T2_LSB]));
170.            par_t3 = (int8_t) (coeff_array1[PAR_T3]);
171.            par_h1 = (uint16_t) (((uint16_t) coeff_array2[PAR_H1_MSB]
172.                    << BME680_HUM_REG_SHIFT_VAL)
173.                    | (coeff_array2[PAR_H1_LSB] & BME680_BIT_H1_DATA_MASK));
174.            par_h2 = (uint16_t) (((uint16_t) coeff_array2[PAR_H2_MSB]
175.                    << BME680_HUM_REG_SHIFT_VAL)
176.                    | ((coeff_array2[PAR_H2_LSB]) >> BME680_HUM_REG_SHIFT_VAL));
177.            par_h3 = (int8_t) (coeff_array2[PAR_H3]);
178.            par_h4 = (int8_t) (coeff_array2[PAR_H4]);
179.            par_h5 = (int8_t) (coeff_array2[PAR_H5]);
180.            par_h6 = (uint8_t) (coeff_array2[PAR_H6]);
181.            par_h7 = (int8_t) (coeff_array2[PAR_H7]);
182.            par_p1 = (uint16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P1_MSB],
183.                    coeff_array1[PAR_P1_LSB]));
184.            par_p2 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P2_MSB],
185.                    coeff_array1[PAR_P2_LSB]));
186.            par_p3 = (int8_t) (coeff_array1[PAR_P3]);
187.            par_p4 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P4_MSB],
188.                    coeff_array1[PAR_P4_LSB]));
189.            par_p5 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P5_MSB],
190.                    coeff_array1[PAR_P5_LSB]));
191.            par_p6 = (int8_t) (coeff_array1[PAR_P6]);
192.            par_p7 = (int8_t) (coeff_array1[PAR_P7]);
193.            par_p8 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P8_MSB],
194.                    coeff_array1[PAR_P8_LSB]));
195.            par_p9 = (int16_t) (BME680_CONCAT_BYTES(coeff_array1[PAR_P9_MSB],
```

```
196.                    coeff_array1[PAR_P9_LSB]));
197.            par_p10 = (uint8_t) (coeff_array1[PAR_P10]);
198.        }
199.
200.        void BME680_GET_DATA(int32_t *temp, int32_t *hum, int32_t *press) {
201.            BME680_READ_SENSOR();
202.            *temp = m_temperature;
203.            *hum = m_humidity;
204.            *press = m_pressure;
205.        }
206.
207.        /* Function for reading an calculating data from sensor */
208.        static void BME680_READ_SENSOR() {
209.            uint8_t data_array[BME680_DATA_SIZE];
210.            uint8_t state = 0xff;
211.            int64_t var1, var2, var3, var4, var5, var6, temp_scaled; // Work variables
212.            uint32_t adc_temp, adc_press;            // Raw ADC temperature and pressure
213.            uint16_t adc_hum;                           // Raw ADC humidity
214.            BME680_MODE_SET(FORCED);
215.            while ((state & 0b00100000) != 0) {
216.                readRegister(m_i2c_handle, m_i2cAddress, BME680_STATUS_REG, &state, 1);
217.            }
218.            readRegister(m_i2c_handle, m_i2cAddress, BME680_DATA_START_ADDRESS,
219.                    data_array, BME680_DATA_SIZE);
220.            adc_press = (uint32_t) (((uint32_t) data_array[PRESS_ADC_MSB] * 4096)
221.                    | ((uint32_t) data_array[PRESS_ADC_LSB] * 16)
222.                    | ((uint32_t) data_array[PRESS_ADC_XLSB] / 16));
223.            adc_temp = (uint32_t) (((uint32_t) data_array[TEMP_ADC_MSB] * 4096)
224.                    | ((uint32_t) data_array[TEMP_ADC_LSB] * 16)
225.                    | ((uint32_t) data_array[TEMP_ADC_XLSB] / 16));
226.            adc_hum = (uint16_t) (((uint32_t) data_array[HUM_ADC_MSB] * 256)
227.                    | (uint32_t) data_array[HUM_ADC_LSB]);
228.        //calculate temperature
229.            var1 = ((int32_t) adc_temp >> 3) - ((int32_t) par_t1 << 1);
230.            var2 = (var1 * (int32_t) par_t2) >> 11;
231.            var3 = ((((var1 >> 1) * (var1 >> 1)) >> 12) * ((int32_t) par_t3 << 4))
232.                    >> 14;
233.            m_tfine = (int32_t) (var2 + var3);
234.            m_temperature = (int16_t) (((m_tfine * 5) + 128) >> 8);
235.            //calculate pressure
236.            var1 = (((int32_t) m_tfine) >> 1) - 64000;
237.            var2 = ((((var1 >> 2) * (var1 >> 2)) >> 11) * (int32_t) par_p6) >> 2;
238.            var2 = var2 + ((var1 * (int32_t) par_p5) << 1);
239.            var2 = (var2 >> 2) + ((int32_t) par_p4 << 16);
240.            var1 =
241.                    (((((var1 >> 2) * (var1 >> 2)) >> 13) * ((int32_t) par_p3 << 5))
242.                            >> 3) + (((int32_t) par_p2 * var1) >> 1);
243.            var1 = var1 >> 18;
244.            var1 = ((32768 + var1) * (int32_t) par_p1) >> 15;
245.            m_pressure = 1048576 - adc_press;
246.            m_pressure = (uint32_t) ((m_pressure - (var2 >> 12)) * ((uint32_t) 3125));
247.            if (m_pressure >= (1 << 30))
248.                m_pressure = ((m_pressure / (uint32_t) var1) << 1);
249.            else
250.                m_pressure = ((m_pressure << 1) / (uint32_t) var1);
251.            var1 = ((int32_t) par_p9
252.                    * (int32_t) (((m_pressure >> 3) * (m_pressure >> 3)) >> 13)) >> 12;
253.            var2 = ((int32_t) (m_pressure >> 2) * (int32_t) par_p8) >> 13;
254.            var3 = ((int32_t) (m_pressure >> 8) * (int32_t) (m_pressure >> 8)
255.                    * (int32_t) (m_pressure >> 8) * (int32_t) par_p10) >> 17;
256.            m_pressure = (int32_t) (m_pressure)
257.                    + ((var1 + var2 + var3 + ((int32_t) par_p7 << 7)) >> 4);
258.            //calculate humidity
```

```
259.           temp_scaled = (((int32_t) m_tfine * 5) + 128) >> 8;
260.           var1 = (int32_t) adc_hum - (int32_t) ((int32_t) par_h1 << 4)
261.                   - (((temp_scaled * (int32_t) par_h3) / ((int32_t) 100)) >> 1);
262.           var2 = ((int32_t) par_h2
263.                   * (((temp_scaled * (int32_t) par_h4) / ((int32_t) 100))
264.                           + (((temp_scaled
265.                                   * ((temp_scaled * (int32_t) par_h5)
266.                                           / ((int32_t) 100))) >> 6) / ((int32_t) 100))
267.                           + (int32_t) (1 << 14))) >> 10;
268.           var3 = var1 * var2;
269.           var4 = (int32_t) par_h6 << 7;
270.           var4 = ((var4) + ((temp_scaled * (int32_t) par_h7) / ((int32_t) 100))) >> 4;
271.           var5 = ((var3 >> 14) * (var3 >> 14)) >> 10;
272.           var6 = (var4 * var5) >> 1;
273.           m_humidity = (((var3 + var6) >> 10) * ((int32_t) 1000)) >> 12;
274.           if (m_humidity > 100000) /* Cap at 100%rH */
275.               m_humidity = 100000;
276.           else if (m_humidity < 0)
277.               m_humidity = 0;
278.       }
```

P.5.3. JavaScript programski kod za aplikaciju u oblaku

```
// UDP_PARSER

function hexToInt(val){
if ((val & 0x80000000) < 0) {
    val = val - 0x100000000;
}
  return val;
}

function main(params, callback){
    var result = [
      {
          // temperatura
           "key": "Temperatura",
           "value": (hexToInt(parseInt('0x'+params.data.substring(0,8)))/10)
      },
      {
          // vlaga
           "key": "Vlaga",
           "value": (parseInt('0x'+params.data.substring(8,16))/10)
      },
      {
          // tlak
           "key": "Tlak",
           "value": (parseInt('0x'+params.data.substring(16,24))/10)
      },
      {
          // Napon baterije
           "key": "Napon_baterije",
           "value": (parseInt('0x'+params.data.substring(24,28))/1000)
      },
      {
          // PM_1
           "key": "PM_1",
           "value": parseInt('0x'+params.data.substring(28,32))
      },
      {
          // PM_2.5
           "key": "PM_2.5",
           "value": parseInt('0x'+params.data.substring(32,36))
      },
      {
          // PM_10
           "key": "PM_10",
           "value": parseInt('0x'+params.data.substring(36,40))
      },
      {
          // CO_ppm
           "key": "CO_ppm",
           "value": (parseInt('0x'+params.data.substring(40,44))/10)
      },
      {
          // NO2_ppm
           "key": "NO2_ppm",
           "value": (parseInt('0x'+params.data.substring(44,48))/10)
      }
    ]
    callback(null, result);
}
```