Regulator upravljanja kvalitetom zraka u ventilacijskom sustavu

Spišić, Josip

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

REGULATOR UPRAVLJANJA KVALITETOM ZRAKA U VENTILACIJSKOM SUSTAVU

Diplomski rad

Josip Spišić

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 26.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Josip Spišić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1014R, 24.09.2019.
OIB studenta:	71644013012
Mentor:	Izv. prof. dr. sc. Ivan Aleksi
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Josip Balen
Član Povjerenstva 1:	Izv. prof. dr. sc. Ivan Aleksi
Član Povjerenstva 2:	Doc.dr.sc. Tomislav Matić
Naslov diplomskog rada:	Regulator upravljanja kvalitetom zraka u ventilacijskom sustavu
Znanstvena grana rada:	**Procesno računarstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Zadatak ovog diplomskog rada je dizajnirati mikrouoravljački sustav temeljen na ARM mikroprocesoru koji služi za upravljanje kvalitetom zraka u HVAC sustavu ventilacije. Uređaj se sastoji od mjerenja temperature, vlage i kvalitete zraka. U radu se koristi Industrijska komunikacija rs485.
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:	26.09.2020.

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, 30.09.2020.

Ime i prezime studenta:	Josip Spišić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1014R, 24.09.2019.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Regulator upravljanja kvalitetom zraka u ventilacijskom sustavu**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Ivan Aleksi

i sumentora

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:

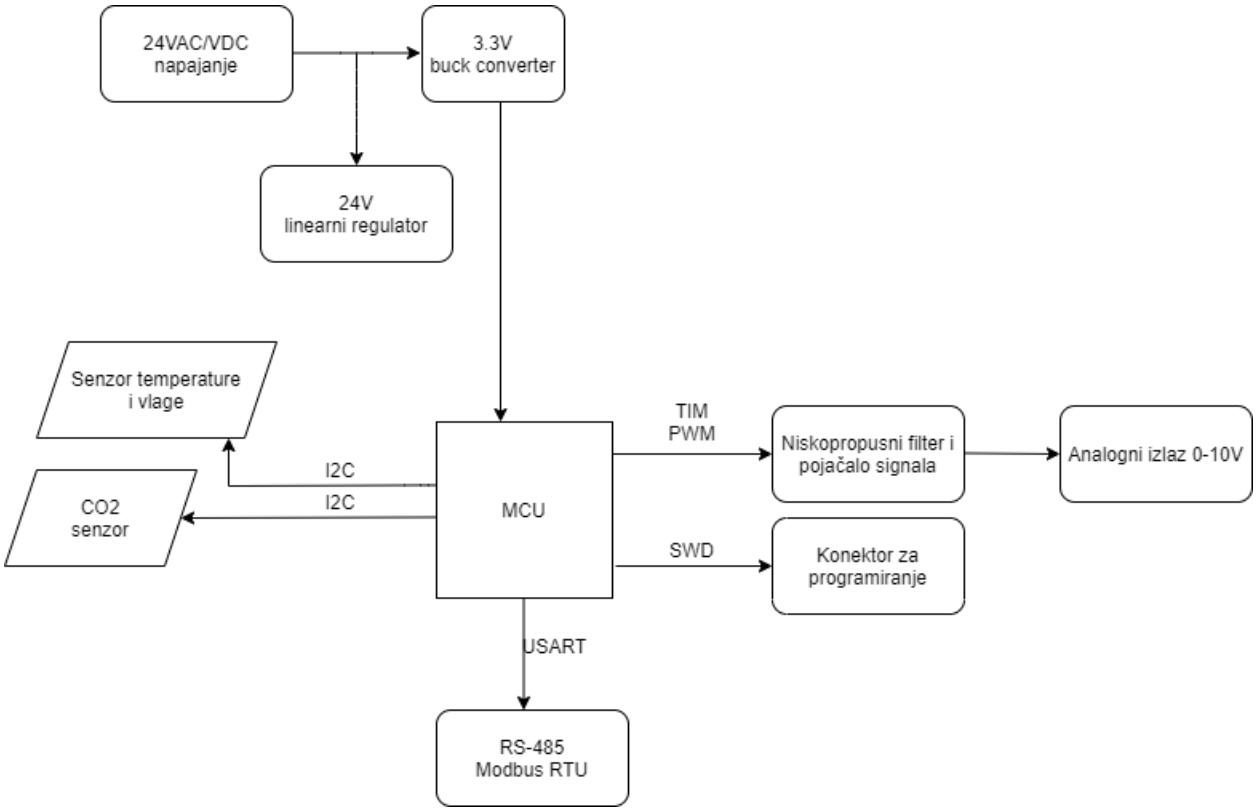
SADRŽAJ

1. UVOD

Cilj ovog diplomskog rada je dizajnirati mikroupravljački sustav temeljen na ARM mikroprocesoru koji služi za upravljanje kvalitetom zraka u HVAC sustavu ventilacije. Uređaj se sastoji od mjerenja temperature, vlage i kvalitete zraka tj. broja čestica CO2 na milijun čestica zraka(engl. *ppm- parts per million*). U radu se koristi industrijska komunikacija RS-485. Uređaj mora imati dva analogna izlaza za upravljanje aktuatorima (regulacijski ventil, žaluzina, ventilator) te mogućnost komunikacije sa upravljačkim PLC-om preko RS-485 standarda(Modbus, BacNet MS/TP,…). Potrebno je izraditi uređaj koji će obavljati sve zadane funkcije i imati mogućnost integracije u složeni sustav upravljanja i regulacije u zgradarstvu.

2. FUNKCIONALNI OPIS REGULATORA

Funkcijski dijagram regulatora prikazuje osnovne građevne blokove uređaja i njihovu međusobnu interakciju sa glavnim mikrokontrolerom. Svaki blok prikazuje na koji periferni dio mikrokontrolera se povezuje (I2C,SPI, TIM PWM, USART). U ovom poglavlju detaljnije će biti opisan svaki pojedini podsustav i realne komponente koje su korištene za realizaciju stvarnog hardvera. Slika 3.1. prikazuje osnovne građevne dijelove regulatora za upravljanje ventilacijom.



Slika 3.1. *Funkcionalni dijagram regulatora za upravljanje ventilacijom*

Regulator dizajniran u ovom diplomskom radu za zadatak ima povećati razinu udobnosti i energetsku učinkovitost novih i postojećih sustava ventilacije uz minimalnu cijenu.

2.1. Podsustavi regulatora

2.1.1. Napajanje

Napajanje je dio hardvera koji osigurava napon i struju elektronici uređaja. Ono osigurava da svaki dio uređaja dobije određenu količinu energije koja mu je potrebna, s obzirom na naponsku razinu koja je potrebna određenom dijelu (3,3V ili 24V).

Regulator izrađen u ovom diplomskom radu ima mogućnost napajanja sa širokim rasponom DC napona (15 VDC do 26 VDC) i AC napona (24VAC).

XL1509 je DC/DC pretvarač koji je odabran za napajanje mikrokontrolera i ostale niskonaponske periferije. Odabran je zbog fiksne frekvencije od 150 KHz. Pretvarač je sposoban isporučiti do 2A s visokom učinkovitošću, malim „*peak-to-peak*“ naponom i dobrom regulacijom opterećenja. Zahtijevajući minimalni broj vanjskih komponenata, regulator je jednostavan za upotrebu. Upravljački PWM krug može linearno regulirati izlaz od 0 do 100%. Pretvarač ima ugrađeno unutarnje ograničenje prenapona i unutarnje ograničenje struje[2].

Linearni regulator L78M24ABDT-TR korišten je samo kao stabilizator napona koji napaja operacijsko pojačalo signala za analogne izlaze 0 – 10V. L78M24ABDT-TR koristi unutarnje ograničenje struje i termičko gašenje u slučaju pregrijavanja te uz pravilno hlađenje može isporučiti do 500mA struje. U slučaju rada koji promatramo, ako su aktivna oba izlaza pojačala potrošnja ne prelazi 40mA, tj. 20mA po kanalu pojačala, što je maksimalna struja koju operacijsko pojačalo može isporučiti[3].

2.1.2. Mikrokontroler

Mikrokontroler je uređaj s integriranim krugom (engl. *IC*) koji se koristi za upravljanje ostalim dijelovima elektroničkog sustava, obično putem mikroprocesorske jedinice (engl. *MPU*), memorije i nekih perifernih uređaja. On je optimiziran za ugrađene aplikacije koje zahtijevaju funkcionalnost obrade, te agilnu i reaktivnu interakciju s digitalnim, analognim ili elektromehaničkim komponentama.

Mikrokontroler dobro je odabran naziv jer naglašava definiranje karakteristika ove kategorije proizvoda. Prefiks "mikro" podrazumijeva malenkost, a pojam "kontroler" podrazumijeva poboljšanu sposobnost izvođenja kontrolnih funkcija. Kao što je prethodno rečeno, ova je funkcija rezultat kombiniranja digitalnog procesora i digitalne memorije s dodatnim hardverom koji je posebno dizajniran kako bi pomogao mikrokontroleru u interakciji s drugim komponentama.

Za realizaciju ovog regulatora odabran je mikrokontroler ST-ove serije mikrokontrolera opće namjene STM32F103 koji pokriva potrebe velikog broja aplikacija na industrijskim, medicinskim i potrošačkim uređajima.

Visoke performanse s prvoklasnim perifernim uređajima i niskonaponski rad male potrošnje uparen je s visokom razinom integracije po pristupačnim cijenama uz jednostavnu arhitekturu i alate jednostavne za upotrebu[1].

- -40 to 105°C range
- USART, SPI, I²C
- 16- and 32-bit timers
- Temperature sensor
- Up to 3x12-bit ADC
- Dual 12-bit ADC
- Low voltage 2.0 to 3.6V (5V tolerant I/Os)

Product line	FCPU (MHz)	Flash (Kbytes)	RAM (Kbytes)	USB 2.0 FS	USB 2.0 FS	FSMC	CAN 2.0B	3-phase MC Timer	I²S	SDIO	Ethernet IEEE1588	HDMI CEC
STM32F100 Value line	24	16 to 512	4 to 32			•		•				•
STM32F101	36	16 to 1M	4 to 80			•						
STM32F102	48	16 to 128	4 to 16	•								
STM32F103	72	16 to 1M	4 to 96	•		•	•	•	•	•		
STM32F105 STM32F107	72	64 to 256	64		•	•	•	•	•		•	

Slika 3.2. *Karakteristike STM32F103 mikrokontrolera[1]*

2.1.3. Senzor temperature i vlage

Senzor temperature i vlažnosti Si7021 digitalni je senzor sa I2C komunikacijom što olakšava njegovu implementaciju s minimalnim brojem popratnih komponenti. Senzor uključuje integrirani jednostruki čip osjetnika temperature i vlage, koji se povezuje u 8-bitni mikrokontroler. Si7021 senzor ima prednosti poput izvrsne kvalitete mjerenja, vrlo brzog reagiranja i ekonomične cijene, a nudi precizno, tvornički kalibrirano digitalno rješenje s malom potrošnjom energije za mjerenje vlažnosti, temperature i točke rosišta [4].

Slika 3.3. *Izgled Si7021 senzora[20]*

2.1.4. ezPyro CO2 senzor

ezPyro CO2 senzor pripada skupini digitalnih piroelektričnih IR senzora za detekciju i mjerenje koncentracije plinova. Kombinira visokokvalitetne senzore s visokom razinom prilagodljivosti i elektroničke integracije u malom SMD paketu. Osigurava visoku osjetljivost u kombinaciji s brzim vremenskim odzivom i točnim otkrivanjem ciljanih plinova. Ovaj senzor integrira digitalni, trenutni način očitavanja koji omogućuje niže radne cikluse IR emitera, čime značajno štedi na razini potrošnje energije na razini sustava. Programabilno pojačanje i filtriranje nudi maksimalnu fleksibilnost u dizajnu sustava. Industrijski standardna I2C komunikacija omogućuje *plug-and-play* povezivost s mikrokontrolerima i omogućuje jednostavno podešavanje

i kalibriranje. EzPyro senzor je s vremenom vrlo stabilan, što osigurava dugi životni vijek bez potrebe za održavanjem. On također može biti lančano povezan kako bi se omogućilo sinkronizirano uzorkovanje između uređaja[6].

Slika 3.4. *ezPyro CO2 senzora[21]*

2.1.5. Operacijsko pojačalo

Operacijsko pojačalo je elektronički sklop koji omogućuje pojačavanje signala kada je to potrebno. Spajanjem operacijskog pojačala u negativnoj povratnoj vezi sa dva otpornika moguće je dobiti željeno pojačanje signala[7].

2.1.6. RS-485 odašiljač

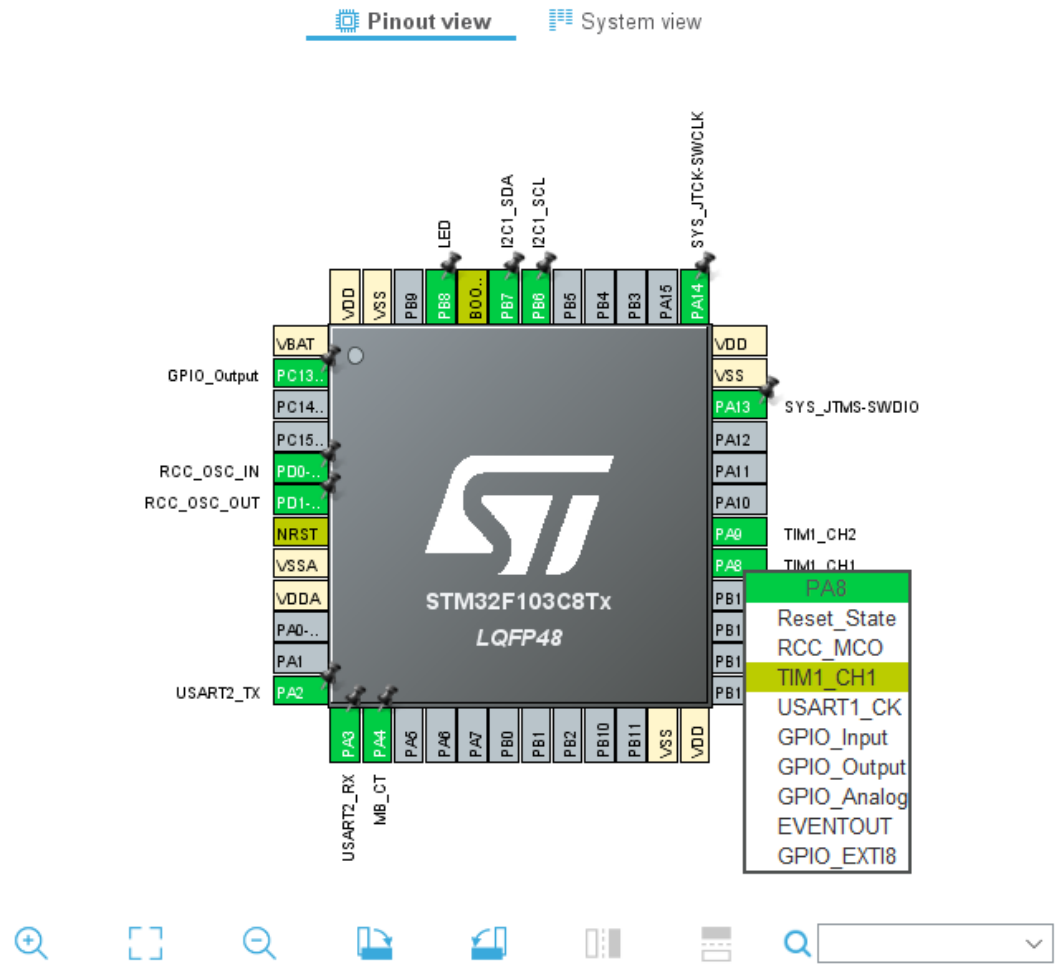
RS-485 i RS-422 standardi su žičane komunikacije na dvije žice. Oni koriste diferencijalni signal koja omogućuje prijenos podataka na velike udaljenosti u industriji i tvorničkim okruženjima za automatizaciju. Diferencijalni signal odbija šum uobičajenog načina rada, a preporučeni uvrnuti par kabela osigurava da većina primljenih smetnji ne utječe na uobičajeni način rada[8].

2.2. Konfiguracija pinova mikrokontrolera

Nakon odabira komponenti koje se planiraju koristiti, za izradu sheme i tiskane pločice potrebno je odrediti na koje pinove mikrokontrolera će pojedine komponente biti spojene, kako bi izradom korisničkog C koda povezali hardver i softver da obavljaju željenu funkciju.

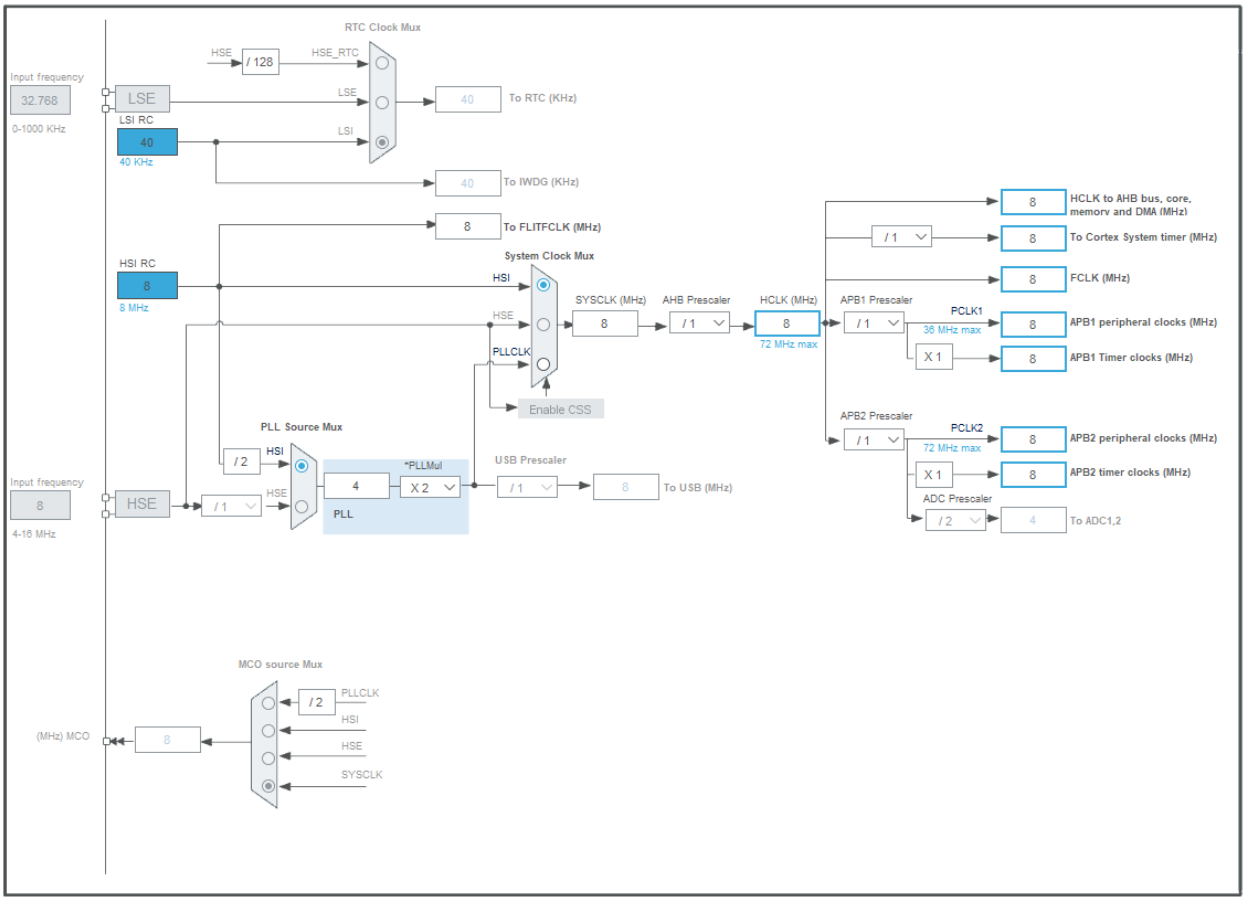
STM32CubeMX je grafički alat koji omogućuje vrlo jednostavnu konfiguraciju STM32 mikrokontrolera i mikroprocesora, kao i generiranje odgovarajućeg inicijalizacijskog C koda za jezgru Arm Cortex-M[5].

Slika 3.5. prikazuje pinout konfiguraciju mikrokontrolera korištenu u ovom diplomskom radu. Uz osnovnu dodjelu funkcija pinova moguće je definirati i dodatne mogućnosti kao što je vidljivo slici. Pin PA8 trenutno je definiran kao PWM pin povezan na TIM1, a u opcijama se vide i ostale mogućnosti koje pin PA8 može izvršavati ako se drugačije postave.



Slika 3.5. *Konfiguracijski prozor mikrokontrolera*

Prozor za konfiguraciju *Clock Configuration*. STM32CubeMX daje shemu pregleda puteva takta procesora, izvora takta, razdjelnika i množitelja. Padajući izbornici i gumbi se mogu koristiti za izmjenu stvarne konfiguracije stabla radi zadovoljavanja zahtjeva kao što je npr. podešavanje takta timera, kako bi se postigla odgovarajuća vremenska baza za generiranje PWM signala. Slika 3.6. prikazuje prozor za konfiguraciju takta.



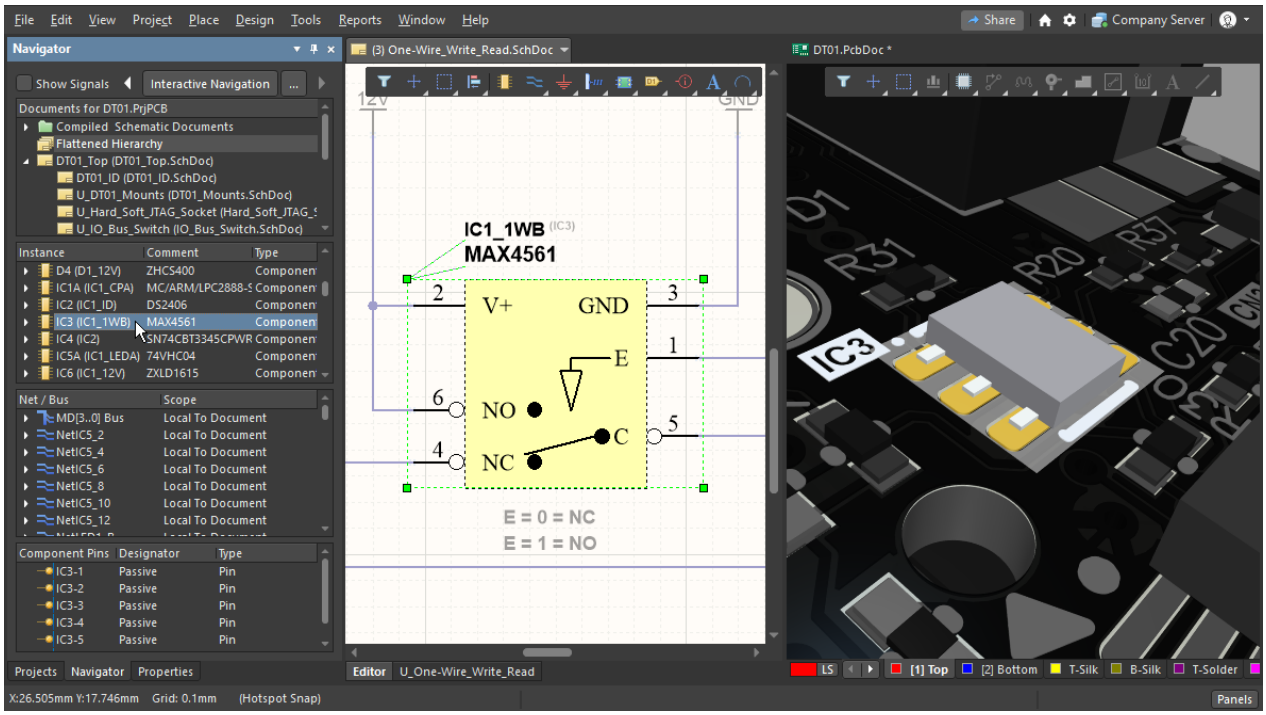
Slika 3.6. *Prozor za konfiguraciju takta mikrokontolera*

2.3. Realizacija regulatora

2.3.1. Dizajn sheme

Shema za ovaj diplomski rad izrađena je pomoću programskog alata Altium Designer od tvrtke Altium Limited. Altium Designer je CAD (engl. *CAD- Computer Aided Design*) softver za dizajn elektronike i tiskanih pločica. Ovaj napredan softverski paket nudi sve mogućnosti za razvoj od ideje do gotovog uređaja.

Korisničko sučelje Altium Designer-a prikazano je na slici 3.7.



Slika 3.7. *Izgled sučelja programskog alata Altium Designer[22]*

Shema napajanja nalazi se u prilogu P1.

Shema mikrokontrolera nalazi se u prilogu P2.

Shema pojačala signala i RS-485 odašiljača nalazi se u prilogu P3.

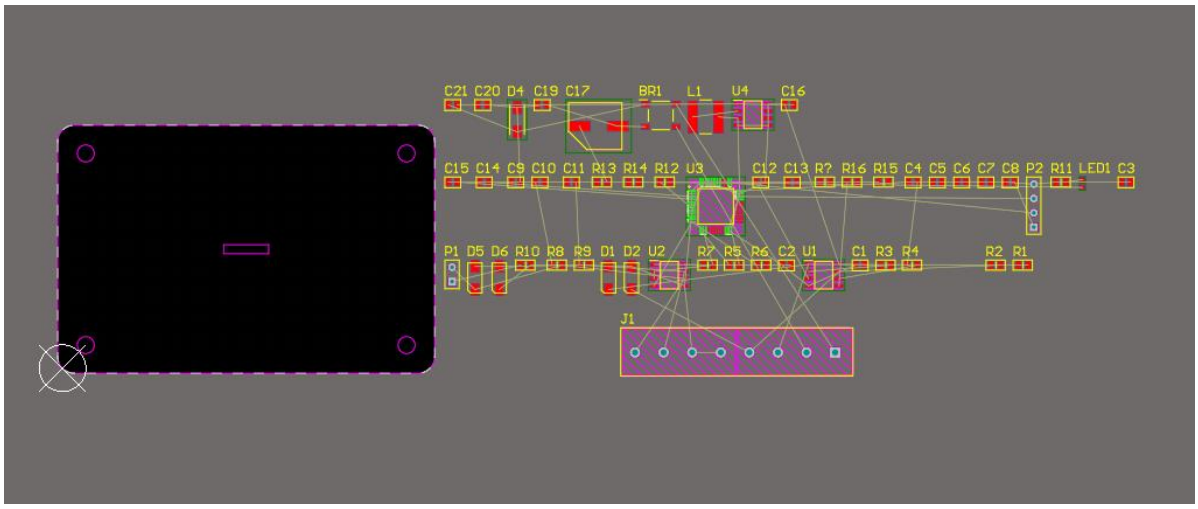
Shema senzora Si7021 i EPY nalazi se u prilogu P4.

Popis komponenti potrebnih za izradu nalazi se u prilogu P5.

2.4. Dizajn tiskane pločice

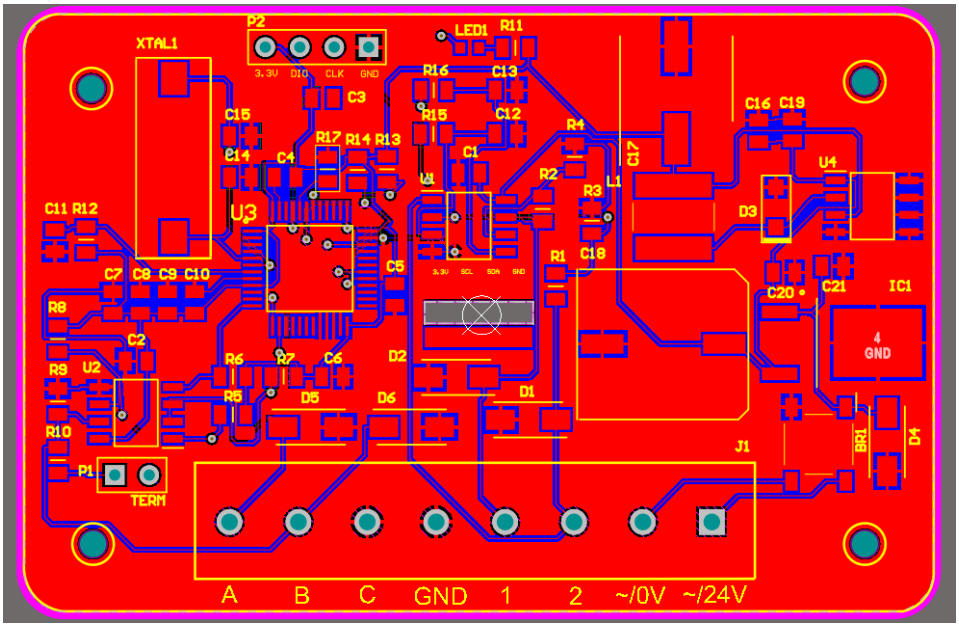
2.4.1. Slaganje komponenti na tiskanu pločicu

Prilikom izrade tiskane pločice nema točnog ili pogrešnog načina. No svejedno postoje određena pravila i dobra praksa koji se prilikom postavljanja komponenti trebaju poštivati kako bi dizajn bio što kvalitetniji. Kod postavljanja SMD komponenti važno je paziti na dobro pozicioniranje komponenti, postavljanje kondenzatora što bliže pinovima integriranog kruga za koji su predviđeni, postavljanje oscilatora blizu ulaza za oscilator, postavljanje zavojnica i kondenzatora blizu izlaza DC/DC pretvarača, itd. Slika 3.8. prikazuje komponente uvezene u prozor za dizajniranje tiskane pločice prije nego što su pravilno raspoređene na prostor tiskane pločice.

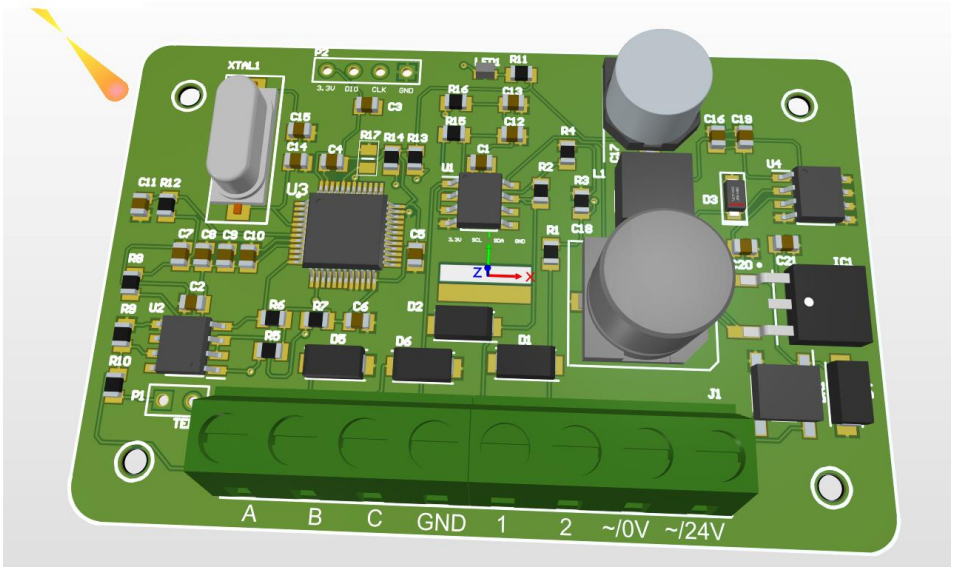


Slika 3.8. *Komponente uvezene u prozor za dizajniranje tiskane pločice*

Slike 3.9. i 3.10. prikazuju 2D i 3D model gotove pločice spremne za izradu.



Slika 3.9. *Pločica spremna za izradu*



Slika 3.10. *Izgled 3D modela pločice*

2.4.2. Izrada tiskane pločice

Nakon slaganja komponenti na tiskanu pločicu sljedeći korak je generiranje *Gerbera*. *Gerber* format je otvoreni ASCII vektorski format za dizajn tiskanih pločica. To je standard koji PCB industrije koriste za opisivanje slika tiskanih ploča: bakreni slojevi, maska za lemljenje, legenda, podaci o bušilici itd.[17].

Nakon izrade *Gerbera* slijedi izrada pločice, koja je izrađena kod profesionalne firme za izradu tiskanih pločica JLCPCB.

Slika 3.11. *Tvornica tvrtke JLCPCB[19]*

2.5. Sastavljanje regulatora

Sastavljanje regulatora sastoji se od nekoliko koraka:

- Lemljenje DC/DC pretvarača i linearnog regulatora
- Lemljenje pasivnih komponenti otpornika, kondenzatora i zavojnica
- Lemljenje mikrokontrolera i ostalih integriranih čipova
- Lemljenje senzora Si7021 i EPY
- Sklapanje uređaja i ugradnja u kućište

2.5.1. Lemljenje

Meko lemljenje je proces pri kojemu se spajaju dva metalna materijala pomoću rastaljenog dodatnog materijala ili lema. Za sastavljanje ove pločice korištena je lemilica marke Weller.

Slika 3.12. *Izrađena pločica spremna za lemljenje*

2.5.2. Izrada upravljačkog programa

Nakon što je pločica dizajnirana i izrađena te je na njoj sastavljen regulator, potrebno je napisati upravljački program koji će povezati hardver i od njega napraviti koristan uređaj. Programiranje je započeto u Cube MX programskom alatu u kojemu su deklarirani pinovi mikrokontrolera te su im dodijeljene funkcije koje će se izvršavati kada uređaj bude završen.

Nakon generiranja inicijalizacijskog C koda za periferiju, Slika 3.13. potrebno je isprogramirati uređaj kako bi se provjerilo radi li on ispravno. Najjednostavniji način za provjeru ispravnosti mikrokontrolera je provjera *blink* LED (engl. *Light-emitting diode*) i provjera SysTick baze timera od 1ms. SysTick timer je 24-bitni timer s automatskim ponovnim učitavanjem. Obično se koristi za pružanje prekida za RTOS (engl. *Real Time Operating System*) ili ostale periodičke zadatke. *Blink* LED je način provjere takav da na određenom pinu mikrokontrolera koji na sebi ima spojenu svjetleću diodu koja se periodički pali i gasi. Nakon navedenih provjera nastavlja se s testiranjem ostalih dijelova periferije i pisanjem logike uređaja.

```c
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */
```

Slika 3.13. *Funkcija za inicijalizaciju I2C-a*

2.5.3. I2C komunikacija

Prilikom dizajniranja elektroničkih uređaja u većini slučajeva oni su kompleksni i sastoje se od više mikroprocesora, integriranih krugova ili senzora, te pri takvom dizajnom sustava postoji potreba za prijenosom i razmjenom podataka između tih podsustava. U načelu postoje dvije osnovne vrste komunikacije između uređaja :

- Paralelna komunikacija
- Serijska komunikacija

Paralelna komunikacija omogućava prijenos više bitova u istom trenutku, najčešće osam tj. jedan bajt, ali taj broj može biti i puno veći. U većini slučajeva današnji mikrokontroleri nemaju dovoljan broj pinova da zadovolje uvjete paralelne komunikacije, iako ona pruža puno veće brzine prijenosa podataka praktično se koristi samo u posebnim slučajevima gdje su velike brzine neophodne za rad sustava.

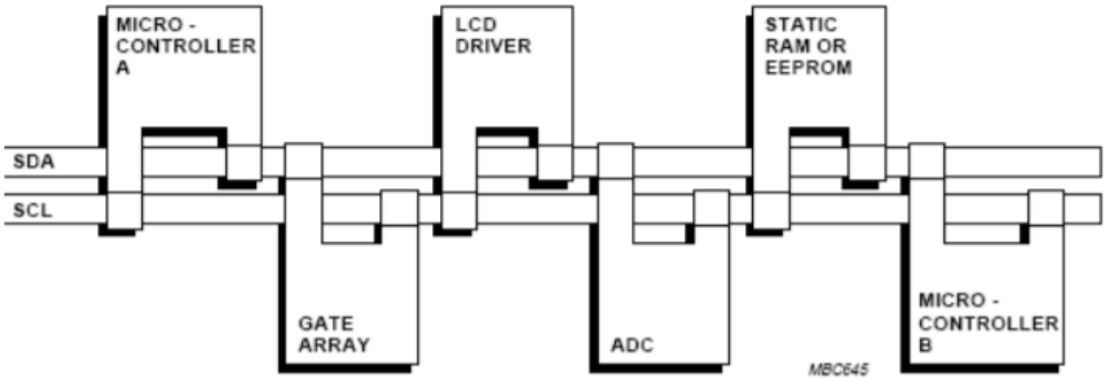
Serijska komunikacija u istom trenutku može prenositi samo jedan bit, tj. bitovi se šalju serijski jedan iza drugoga od izvora do odredišta. Ovakav način komunikacije značajno je

jednostavniji jer za asinkronu jednosmjernu komunikaciju potreban je samo jedan vod, dok paralelna komunikacija zahtjeva barem osam vodova. Upravo radi tog manjeg broja potrebnih vodova serijska komunikacija se najčešće koristi za komunikaciju između uređaja na tiskanoj pločici (engl. *on-bord-protocol*).

I2C je serijska, sinkrona komunikacija kojim se povezuju dva ili više uređaja, te se ona izvršava se pomoću dva voda:

- SCL – (engl. *Serial Clock*), za prijenos takta
- SDA – (engl. *Serial Data*), za prijenos podataka

SCL i SDA su alternativna funkcija pinova mikroprocesora i konfigurirani su kao pinovi otvorenog kolektora (engl. *open drain*), zbog čega se linije SDA i SCL spajaju na napajanje preko otpornika (engl. *pull up resistor*). Iznos otpora ovog otpornika nije fiksno definiran, te se kreće od jednog kiloohma do nekoliko desetaka kiloohma. Kad nema prometa podataka na liniji, ona se nalazi u stanju logičke jedinice. Napon napajanja na liniji je 5V ili 3.3V, iako I2C standard dozvoljava napone širokog raspona. Slika 3.14. prikazuje primjer spajanja komponenata na I2C sabirnicu[10].



Slika 3.14. *Primjer I2C sabirnice[10]*

2.5.4. Biblioteka gotovih programskih funkcija za I2C komunikaciju

Nakon što je ukratko objašnjeno načelo rada I2C komunikacije potrebno je referirati se na dokumentaciju proizvođača[11,12] senzora korištenih u ovom diplomskom radu koji koriste ovaj protokol i prikazati na koji je način realizirana komunikacija s njima.

Prvi senzor je Si7021 senzor temperature i vlage zraka, a drugi senzor je ezPyro CO2 senzor. Cjelovite biblioteke nalaze se u prilogu P6 i P7.

Slika 3.15. i Slika 3.16. prikazuju izgled biblioteki programskih funkcija za senzor Si7021 i ezPyro CO2.

```
typedef enum Si7021_commands
{
  Humi_HM      = 0xE5, // Measure Relative Humidity, Hold Master Mode
  Humi_NHM     = 0xF5, // Measure Relative Humidity, No Hold Master Mode
  Temp_HM      = 0xE3, // Measure Temperature, Hold Master Mode
  Temp_NHM     = 0xF3, // Measure Temperature, No Hold Master Mode
  Temp_AH      = 0xE0, // Read Temperature Value from Previous RH Measurement
  Si7021_Reset = 0xFE, // Reset
  W_RHT_U_reg  = 0xE6, // Write RH/T User Register 1
  R_RHT_U_reg  = 0xE7, // Read RH/T User Register 1
  W_Heater_C_reg = 0x51, // Write Heater Control Register
  R_Heater_C_reg = 0x11, // Read Heater Control Register
  R_ID_Byte11  = 0xFA, // Read Electronic ID 1st Byte, first part
  R_ID_Byte12  = 0x0F, // Read Electronic ID 1st Byte, second part
  R_ID_Byte21  = 0xFC, // Read Electronic ID 2nd Byte, first part
  R_ID_Byte22  = 0xC9, // Read Electronic ID 2nd Byte, second part
  R_Firm_rev1  = 0x84, // Read Firmware Revision, first part
  R_Firm_rev2  = 0xB8  // Read Firmware Revision, second part
}Si7021_commands_t;
```

Slika 3.15. *Enumeracija komandi za Si7021 senzor*

```
/*
 * status
 * @arg
 *  TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY12231_Status(TypeDefHandleEPY12231 *Handle) {
    uint8_t ReturnBuffer[1];

    if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
        return HAL_ERROR;
    } else {
        HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, FIFO_STATUS, 1, ReturnBuffer, 1,
                I2C_TIMEOUT);
    }
    Handle->InversedStatus = ReturnBuffer[0] & 0x01;
    Handle->FIFO_Count = (ReturnBuffer[0] & 0x1E) >> 1;
    Handle->Error = (ReturnBuffer[0] & 0x30) >> 5;
    Handle->WakeDetected = ReturnBuffer[0] & 0x80 >> 7;

    return 1;
}
```

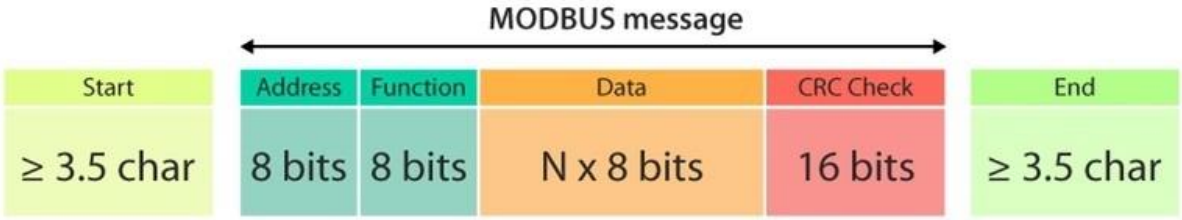
Slika 3.16. *Funkcija za dohvaćanje statusa ezPyro CO2 senzora*

2.5.5. Modbus RTU komunikacija

Modbus RTU (engl. *RTU-remote terminal unit*) je industrijski protokol dizajniran za komunikaciju u industrijskoj informatici, a danas je prihvaćen kao standardni protokol za razne elektroničke uređaje. Modbus RTU zasnovan je na *master – slave* principu gdje jedan *master* uređaj komunicira sa više *slave* uređaja. Poruke slane pomoću Modbus RTU protokola su točno

strukturirane i standardom definirane kako bi uređaji kojima je poruka poslana mrežom mogli primjereno odgovoriti. Struktura poruke je:

- Funkcijski kod
- Podatkovni dio
- CRC provjera



Slika 3.17. *Modbus RTU podatkovni paket[9]*

2.5.6. Biblioteka gotovih programskih funkcija za Modbus komunikaciju

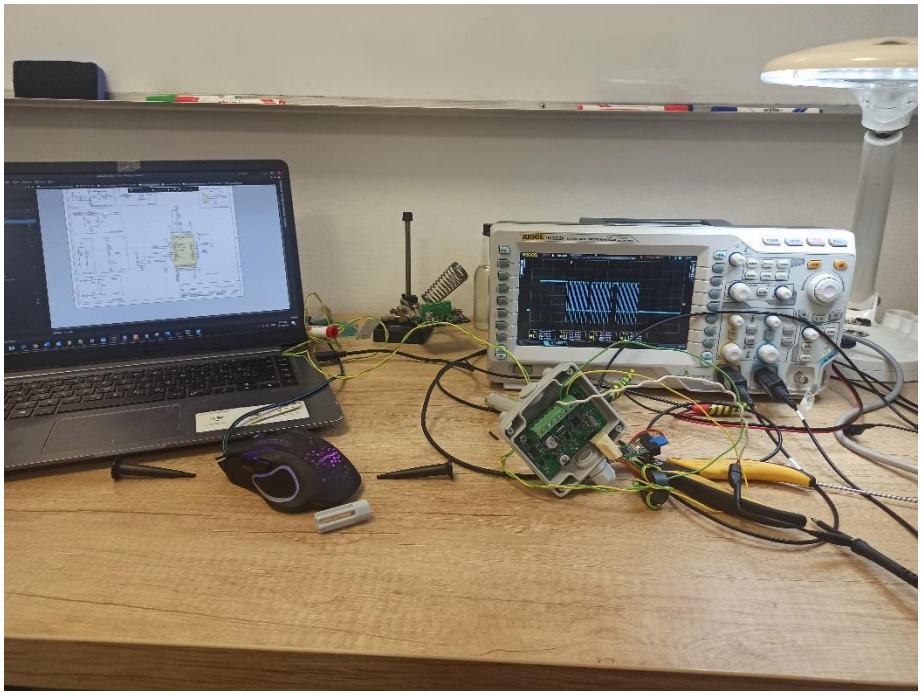
Implementacija Modbus RTU funkcije READ_ONE_HOLDING_REGISTER_RESPONSE može se vidjeti na slici 3.17., a cjeloviti kod dostupan je u privitku P8.

```
void
READ_ONE_HOLDING_REGISTER_RESPONSE(Modbus_HandleTypeDef* Modbus,uint16_t REGISTER_ADDRESS, uint16_t NUM_OF_REGS)
{
if((REGISTER_ADDRESS + NUM_OF_REGS) <= NUM_OF_HOLDING_REGISTERS){
        unsigned char MODBUS_TX_BUFFER[7];

        uint16_t CRC_;

        MODBUS_TX_BUFFER[0] = Modbus->mb_address;
        MODBUS_TX_BUFFER[1] = READ_HOLDING_REGISTER;
        MODBUS_TX_BUFFER[2] = 0x02;

        MODBUS_TX_BUFFER[3] = (Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]) >> 8;
        MODBUS_TX_BUFFER[4] = (Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]) & 0x00FF;

        CRC_ = CRC16(MODBUS_TX_BUFFER,5);

        MODBUS_TX_BUFFER[5] = LOW_BYTE(CRC_);
        MODBUS_TX_BUFFER[6] = HIGH_BYTE(CRC_);

        asm("NOP");
        HAL_UART_Transmit_DMA(Modbus->UART,MODBUS_TX_BUFFER,7);
        while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_TC) != 1);
        //CLEAR_RX_BUFFER();

}else{
    //CLEAR_RX_BUFFER();
}
}
```

Slika 3.17. *Funkcija iz Modbus biblioteke*

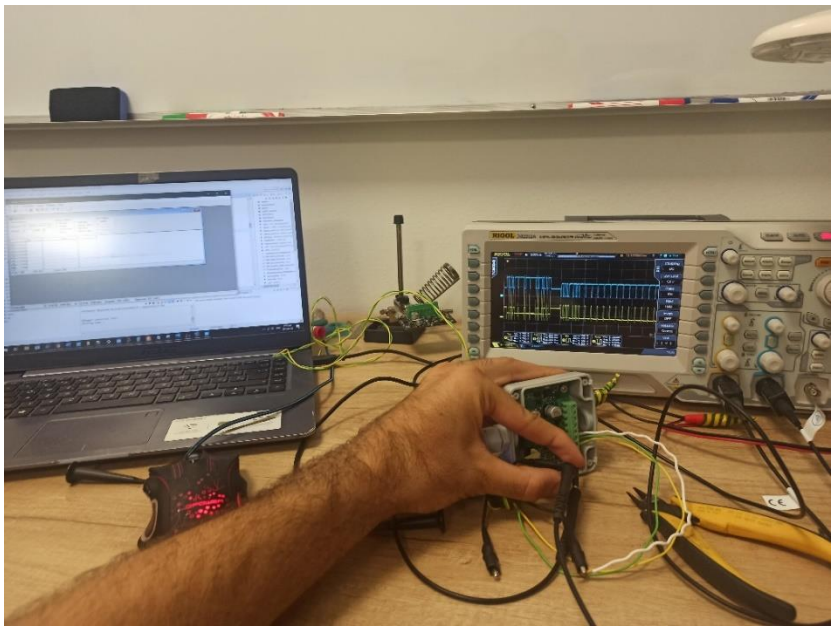
3. TESTIRANJE REGULATORA

Ispitivanje funkcionalnosti uređaja posljednji je korak u dizajnu uređaja, do sada su svi podsustavi ispitani pojedinačno i vrijeme je za testiranje cjelokupnog sustava. Slika 4.1. prikazuje gotov regulator. Za testiranje je korišten digitalni osciloskop, laboratorijsko napajanje, multimetar, RS-485 na UART konverter.



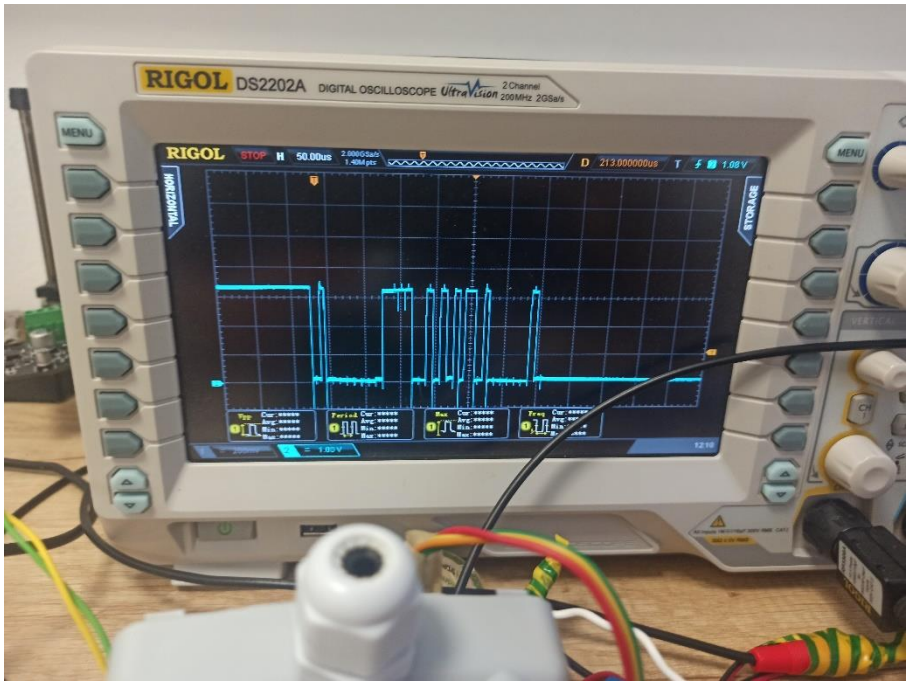
Slika 4.1. *Testiranje*

Testiranje Modbus komunikacije odrađeno je pomoću besplatnog softvera RMMS[18], koji simulira Modbus master uređaj. Slika 4.2. prikazuje izgled signala snimljen osciloskopom. Kvaliteta signala je zadovoljavajuća i komunikacija između računala i regulatora je potpuno funkcionalna. Kada je komunikacija prošla testiranje na red je došlo snimanje kvalitete I2C sabirnice.



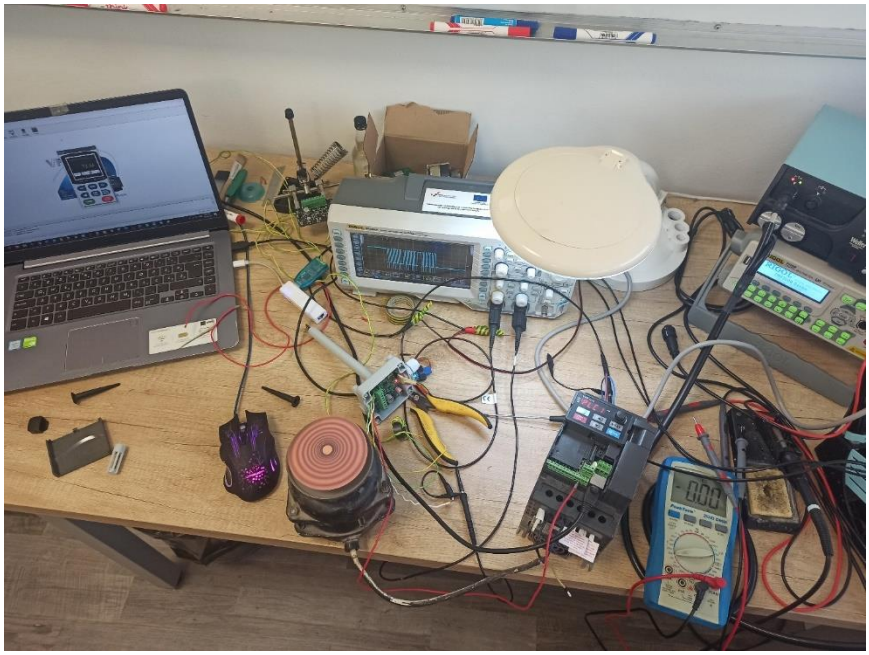
Slika 4.2. *Izgled Modbus signala*

Slika 4.3. prikazuje I2C signal na podatkovnoj liniji. Može se vidjeti kako je signal kvalitetan uz male distorzije signala prisutne zbog parazitnog kapaciteta na dugačkim linijama vodova, no testiranje je pokazalo kako oni ne utječu na mjerenja.



Slika 4.3. Izgled I2C signala

Završni korak u testiranju je povezivanje regulatora sa nekim izvršnim članom i provjeru funkcije analognih izlaza. Slika 4.4. prikazuje regulator spojen na frekventni pretvarač koji je podešen da za referentni signal koristi signal 0-10V koji će primati od regulatora kojeg testiramo.



Slika 4.4. *Izgled testiranja regulatora u povratnoj vezi*

Preko Modbus komunikacije(laptopa) regulator prima referentnu vrijednost koju postavlja korisnik, te izračunava upravljački signal koji je potom proslijeđen frekventnom pretvaraču i na taj način je zatvorena povratna veza i regulator je testiran i spreman za ugradnju. Implementacija digitalnog PI regulatora dostupna je u prilogu P9.

4. ZAKLJUČAK

U ovom diplomskom radu, koristeći mikrokontroler STM32F103 i senzore Si7021 i ezPyro CO2 predstavljen je postupak izrade regulatora za upravljanje kvalitetom zraka korak po korak, od ideje do prototipa. Za izradu sheme i tiskane pločice korišten je Altium Designer, a za programiranje Atollic TrueStudio te C programski jezik. Testiranja provedena na prototipu uređaja pokazuju da regulator sadrži sve očekivane funkcionalnosti za korištenje u praktičnoj primjeni. Regulator je dizajniran tako da ga je vrlo lako implementirati na nove ili već postojeće sustave. Mogućnost njegove integracije na centralni nadzor zgrade stavlja ga korak ispred sličnih uređaja zbog pristupačne cijene, ali i uključi/koristi dizajna.

LITERATURA

[1] STM32F1 serija mikrokontrolera, 20.6.2020.
https://www.st.com/en/microcontrollers-microprocessors/stm32f1-series.html
[2] XL1509 DC/DC pretvarač
https://pdf1.alldatasheet.com/datasheet-pdf/view/1134349/XLSEMI/XL15093.32500E1.html
[3] Linearni regulator L78M24ABDT-TR
https://hr.mouser.com/datasheet/2/389/cd00000447-1795461.pdf
[4] Si7021
https://hr.mouser.com/datasheet/2/368/Si7021-A20-1397917.pdf
[5] STM32CubeMX
https://www.st.com/en/development-tools/stm32cubemx.html
[6] CO2 senzor
https://eu.mouser.com/datasheet/2/948/ezpyro_ir_sensor_for_gas_sensing_short_form_datash-1483387.pdf
[7] Operacijsko pojačalo
https://hr.wikipedia.org/wiki/Operacijsko_poja%C4%8Dalo
[8] RS-485 odašiljač
https://www.renesas.com/eu/en/doc/whitepapers/interface/rs-485-transceiver-tutorial.pdf
[9]Modbus paket
https://www.com-port-monitoring.com/how-to-read-modbus-data/
[10] I2C
http://spvp.zesoi.fer.hr/seminari/2007/seminari/DavorBonaci_I2Csabirnica.pdf
[11] Si7021 dokumentacija
https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf
[12] ezPyro CO2 senzor dokumentacija(potrebna registracija)
https://pyreos.com/
[13] Modbus RTU
http://spvp.zesoi.fer.hr/seminari/2007/seminari/MaraZivcic_Modbus.pdf
[14] Ventilacija učinak na ljudsko zdravlje
http://zdravljezasve.hr/html/zdravlje06_ekologija2.html
[15] Prijenos respiratornih bolesti

Knibbs LD, Morawska L, Bell SC, Grzybowski P. Room ventilation and the risk of airborne infection transmission in 3 health care settings within a large teaching hospital. Am J Infect Control. 2011 Dec;39(10):866-72.

[16] Izmjenjiva topline

https://mcsolar.hr/rekuperacija-zraka/

[17] Gerberi

https://en.wikipedia.org/wiki/Gerber_format

[18] Modbus Master

http://en.radzio.dxp.pl/modbus-master-simulator/

[19] JLCPCB tvornica

http://www.electronoobs.com/eng_blog_13.php

[20] Si7021 senzor slika

https://eu.mouser.com/ProductDetail/Silicon-Labs/SI7021-A20-GM1R?qs=p9T7GgSe1IE3uXqkuVzM1Q%3D%3D

[21] ezPyro CO2 senzor slika

https://eu.mouser.com/ProductDetail/Pyreos/EPY12241?qs=qSfuJ%252Bfl%2Fd6DtzBDH5RbVw==

[22] Altium Designer slika

https://www.altium.com/documentation/altium-designer/working-between-the-schematic-and-the-board-ad?version=18.1

SAŽETAK

Pozitivne promjene u graditeljstvu i poticaji za izgradnju energetski učinkovitih objekata potakli su potrebu za većim korištenjem uređaja za mehaničku ventilaciju, a time se proporcionalno povećala potreba za uređajima za regulacijom i upravljanjem. Komponente korištene u realizaciji regulatora su mikrokontroler STM32F103 i senzori Si7021 i ezPyro CO2. Za izradu sheme i tiskane pločice regulatora korišten je Altium Designer, a za programiranje Atollic TrueStudio te C programski jezik. Kao rezultat rada izrađen je funkcionalan prototip koji je testiranjem zadovoljio sve postavljene funkcionalnosti.

Ključne riječi: ventilacija, regulator, CO2, temperatura, Altium Designer, C, STM32F103

ABSTRACT

Positive changes in construction and incentives for the construction of energy-efficient facilities have stimulated the need for greater use of mechanical ventilation devices, thus increasing the need for regulation and control devices proportionally. The components used in the realization of the controller are the STM32F103 microcontroller and the Si7021 and ezPyro CO2 sensors. Altium Designer was used to creating the diagram and the printed circuit board of the controller, and Atollic TrueStudio and C programming language were used for programming. As a result of the work, a functional prototype was made, which met all the set functionalities by testing.

Keywords: ventilation, regulator, CO2, temperature, Altium Designer, C, STM32F103

ŽIVOTOPIS

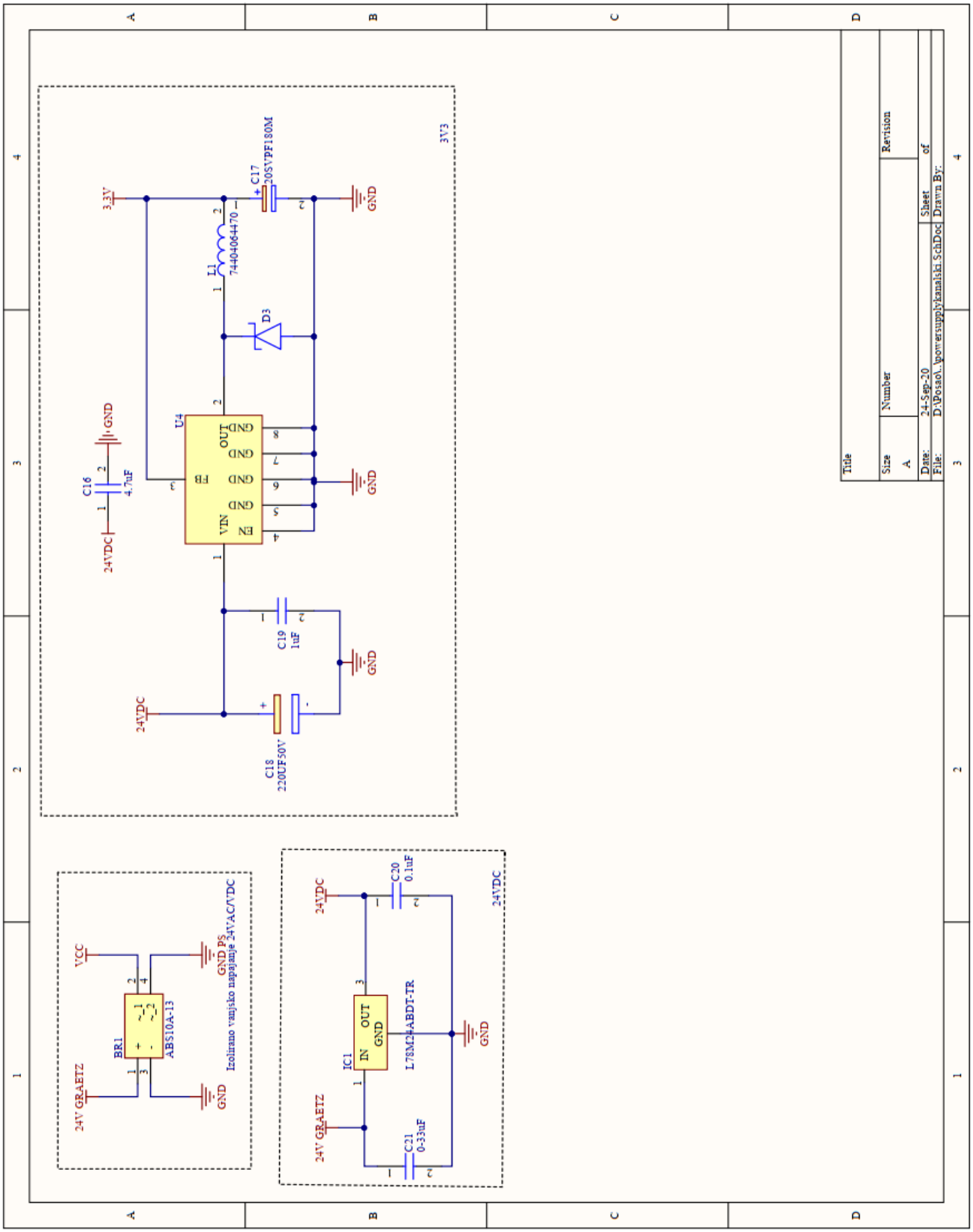
Josip Spišić rođen je 26.srpnja 1995. godine u Đakovu. Osnovnu školu završava 2010. godine u Semeljcima te iste godine upisuje Elektrotehničku i prometnu školu Osijek, smjer tehničar za računarstvo. Kroz srednjoškolsko obrazovanje sudjeluje na projektima vezanim uz električne automobile kao što su dizajniranje i izrada prvog solarnog električnog automobila u srednjim školama pri projektu „Solarni električni automobil“ (SOELA) i ljetna škola „Dizajniranje solarnih automobila“ 2013. godine u slobodno vrijeme volontira kao asistent u nastavi robotike u Centru tehničke kulture Osijek. 2014. godine upisuje Elektrotehnički fakultet Osijek, stručni studij, smjer automatika koji završava 2017. godine. Iste godine upisuje razlikovne obveze na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija, te u jesen 2018. godine upisuje diplomski studij Računarstva, smjer Robotika i umjetna inteligencija. U slobodno vrijeme bavi se elektronikom i sportom. Aktivno se služi engleskim jezikom u govoru i pismu.

Josip Spišić

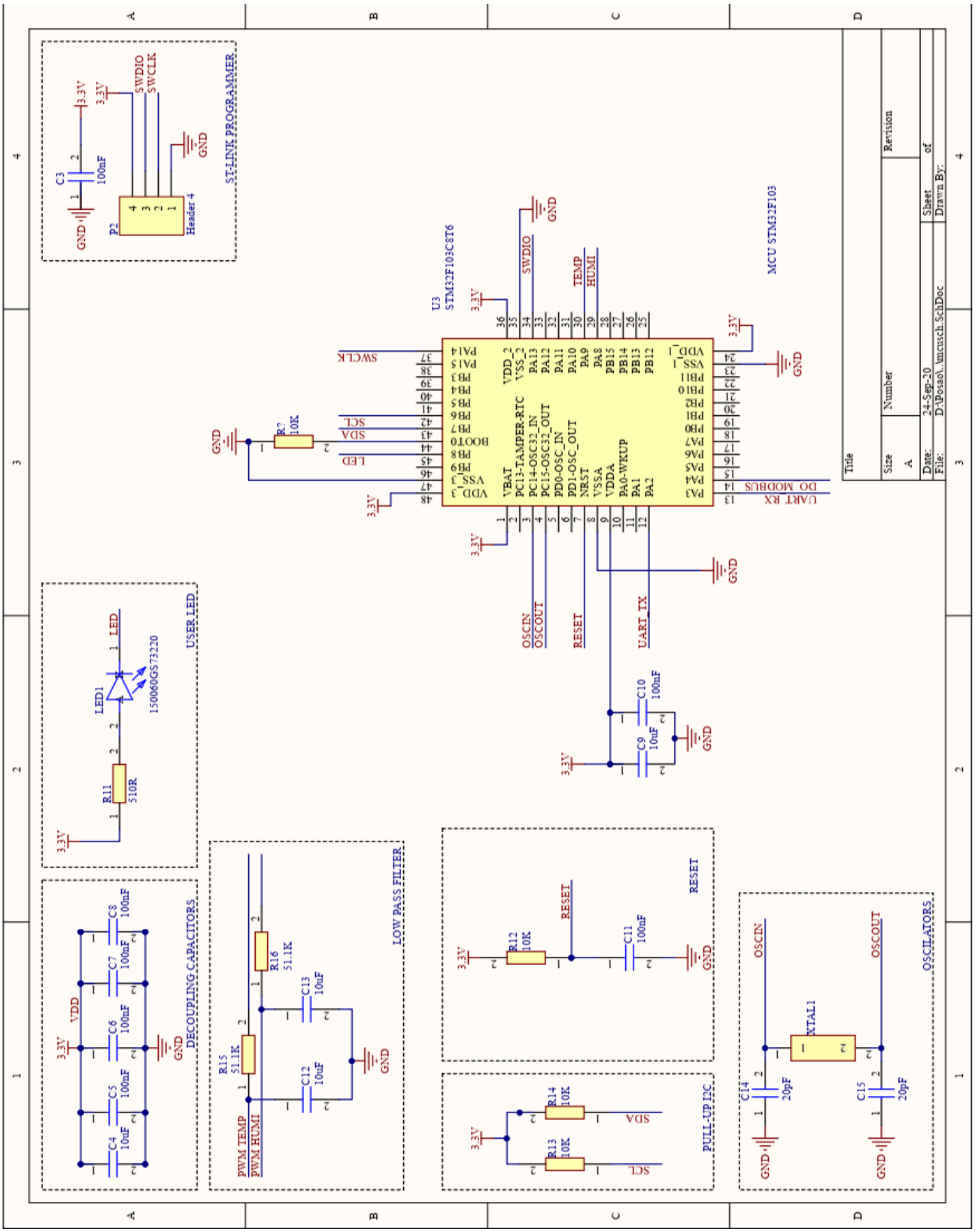
_____

PRILOZI

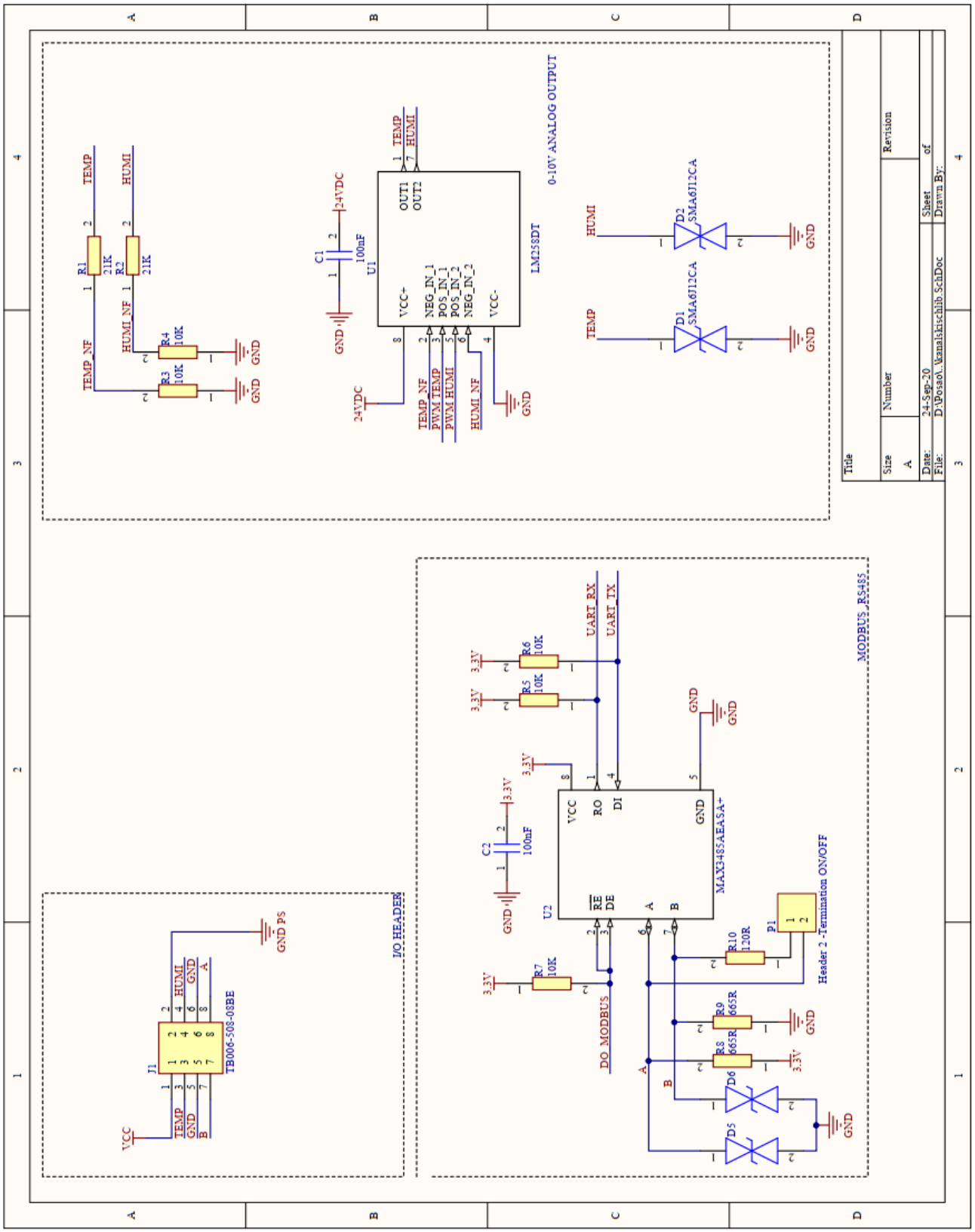
P1 Shema napajanja



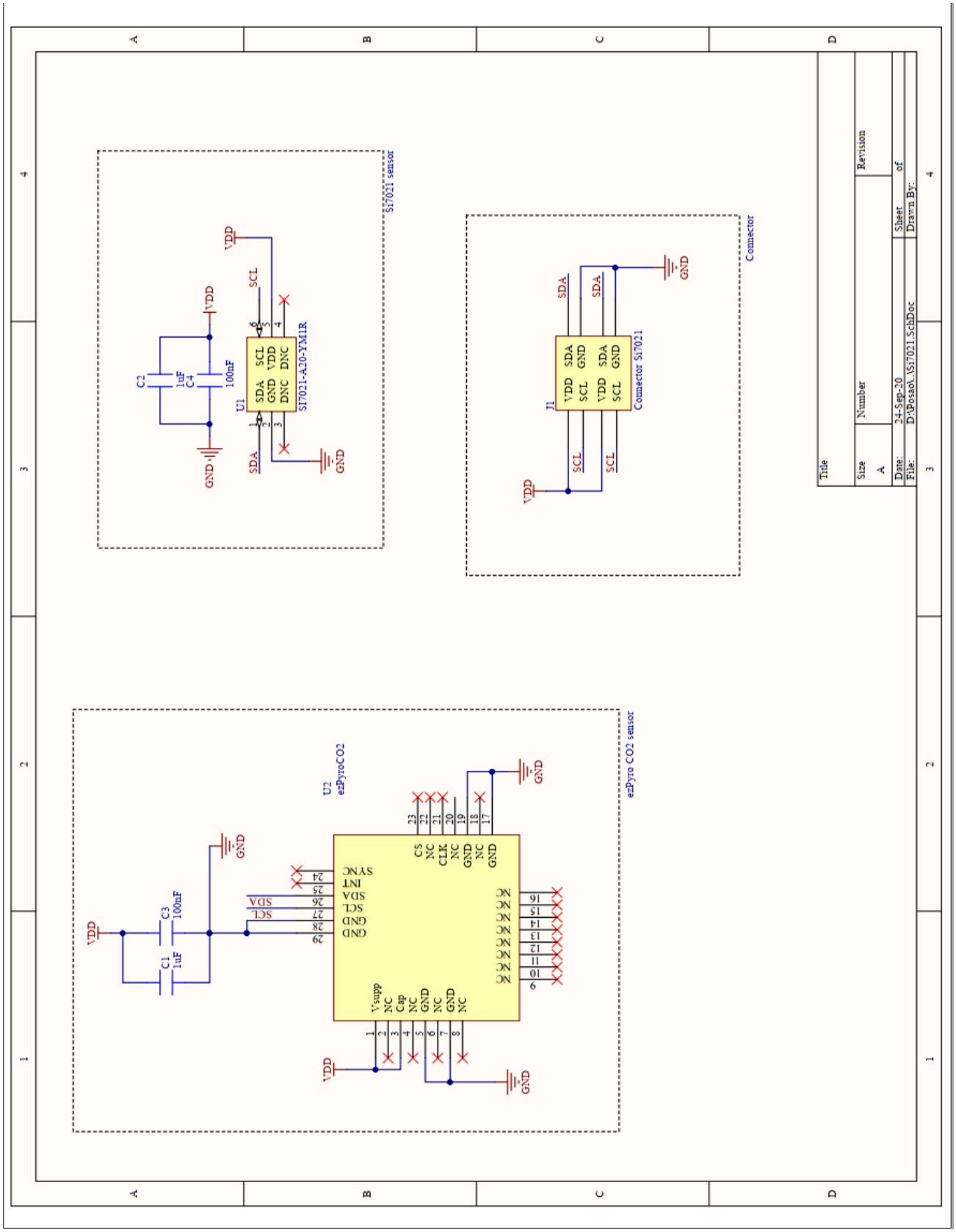
P2 Shema mikrokontrolera



P3 Shema pojačala signala i RS-485 odašiljača



P4 Shema senzora Si7021 i EPY



P5 Popis komponenti

Comment	Description	Designator	Footprint	Quantity
ABS10A-13	Bridge Rectifier	BR1	ABS10A13	1
100nF	Capacitor	C1, C2, C3, C5, C6, C7, C8, C10, C11	CAPC2012X88N	9
10uF	Capacitor	C4, C9, C12, C13	CAPC2012X88N	4
20pF	Capacitor	C14, C15	CAPC2012X88N	2
4.7uF	Capacitor	C16	CAPC2012X88N	1
20SVPF180M	Capacitor Polarised	C17	CAPAE830X700N	1
220UF50V	CAP ALUM 220UF 20% 50V SMD	C18	220UF50V	1
1uF	Capacitor	C19	CAPC2012X88N	1
0.1uF	Capacitor	C20	CAPC2012X88N	1
0-33uF	Capacitor	C21	CAPC2012X88N	1
SMA6J12CA	TVS Diode (Bi-directional)	D1, D2, D5, D6	DIOM5026X229N	4
SS34N	DIODE SCHOTTKY 20V 1A SOD123	D3	D1	1
1SMA40CAT3G	Diode	D4	DIONM5226X220N	1
L78M24ABDT-TR	L78M Series 500 mA 24 V Three Terminal Positive Voltage LDO Regulator - TO-252	IC1	DPAK228P1010X220-3N	1
TB006-508-08BE	Connector	J1	TB00650808BE	1
74404064470	Inductor	L1	74404064470	1
150060GS73220	LED	LED1	1608[0603]	1
Header 2 - Termination ON/OFF	Header, 2-Pin	P1	HDR1X2	1
Header 4	Header, 4-Pin	P2	HDR1X4	1
21K	Resistor	R1, R2	RESC2012X65N	2
10K	Resistor	R3, R4, R5, R6, R7, R12, R13, R14, R17	RESC2012X65N	9
665R	Resistor	R8, R9	RESC2012X65N	2
120R	Resistor	R10	RESC2012X65N	1

510R	Resistor	R11	RESC2012X65N	1
51.1K	Resistor	R15, R16	RESC2012X65N	2
LM258DT	LM258 Series 30 V 1.1 MHz Low Power Dual Operational Amplifier - SOIC-8	U1	SOIC127P600X175-8N	1
MAX3485AEASA+	+3.3V-Powered, A±20kV ESD-Protected, 20Mbps and Slew-Rate-Limited RS-485/RS-422	U2	SOIC127P600X175-8N	1
STM32F103C8T6	Integrated Circuit	U3	QFP50P900X900X160-48N	1
XL15093_3	DC/DC Converter	U4	SOIC127P600X175-8N	1
ECS-80-S-5PX-TR	CRYSTAL 8.0000MHZ SERIES SMD	XTAL1	CRYSTAL 8.0000MHZ SERIES SMD	1

P6 Biblioteka Si7021

```
/*
 * Si7021.c
 *
 *  Created on: Nov 7, 2019
 *      Author: Josip
 */

#include"Si7021.h"

void Si7021_Config(uint8_t param){
	uint8_t temp[3];
	temp[0] = READ_UR1;
	HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,&temp[0],1,100);

	HAL_I2C_Master_Receive(&hi2c3,ADDRESS_Si7021,temp,1,100);
	temp[1] = temp[0];
	switch (param) {
	case M_RES_RH12_T14:
		temp[0] = WRITE_UR1;
		temp[1] &= ~0x01;
		temp[1] &= ~0x80;

		HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,&temp[1],2,100);
		break;

	case M_RES_RH08_T12:
		temp[0] = WRITE_UR1;
		temp[2] |=  0x01;
		temp[2] &= ~0x80;

		HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,&temp[1],2,100);
		break;

	case M_RES_RH10_T13:
		temp[0] = WRITE_UR1;
		temp[1] &= ~0x01;
		temp[1] |=  0x80;

		HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,&temp[1],2,100);
		break;

	case M_RES_RH11_T11:
		temp[0] = WRITE_UR1;
		temp[2] |= 0x01;
		temp[2] |= 0x80;

		HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,&temp[1],2,100);
		break;

	default:

		break;
	}
}

/*
```

```
 * @param command -> MEASURE_T_HM, MEASURE_T_NHM, READ_TEMP_FROM_PREVIOUS_RH_M
 *
 */

float Si7021_readTemperature(uint8_t command){
        uint8_t buff[3];
        buff[0] = command;
        HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,buff,1,100);

        HAL_I2C_Master_Receive(&hi2c3,ADDRESS_Si7021,buff,2,100);
        buff[1] &= ~0x0003;
        return (-46.85 + 175.72 / 65536.0 *(buff[0]<<8 |buff[1] ));

}

/*
 * @param command -> MEASURE_RH_HM, MEASURE_RH_NHM
 *
 */
float Si7021_readHumidity(uint8_t command){
        uint8_t buff[3];
        buff[0] = command;
        HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,buff,1,100);

        HAL_I2C_Master_Receive(&hi2c3,ADDRESS_Si7021,buff,2,100);
        buff[1] &= ~0x0003;
        return (-6.0 + 125.0 / 65536.0 *(buff[0]<<8 |buff[1] ));

}


int Si7021_ReadEHRegister(void){
      uint8_t temp[3] = {0,0,0};
      uint8_t EHRegister[1];

      temp[0] = 0x11;
      HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,temp,1,100);

      HAL_I2C_Master_Receive(&hi2c3,ADDRESS_Si7021,EHRegister,1,100);

      return EHRegister[0];
      }

int Si7021_WriteEHRegister(uint8_t EH_Value){
            uint8_t temp[3] = {0,0,0};
            uint8_t EHRegister[1];

            temp[0] = 0x11;
            HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,temp,1,100);

            HAL_I2C_Master_Receive(&hi2c3,ADDRESS_Si7021,EHRegister,1,100);

            if(EHRegister[0] < 0 || EHRegister[0] > 15){
                  EHRegister[0] = 0;
            }
            else{
                  EHRegister[0] = EH_Value;
            }
```

```
		temp[0] = 0x51;
		temp[1] = EHRegister[0];
		HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,temp,2,100);

		if(Si7021_ReadEHRegister() != EHRegister[0] ){
			return HAL_ERROR;
		}
		else{
			return EHRegister[0];
		}
	}

void Si7021_Reset(void){
	uint8_t temp[3] = {0,0,0};

	temp[0] = 0xFE;
	HAL_I2C_Master_Transmit(&hi2c3,ADDRESS_Si7021,temp,1,100);

}
```

P7 Biblioteka ezPyros CO2 senzora

```
/*
 * EPY12231.c
 *
 *  Created on: May 21, 2020
 *      Author: Josip
 */

#include "EPY12231.h"



void EPY12231_Init(TypeDefHandleEPY12231 *Handle) {
	EPY_EPY12231_Test(Handle);
	for (int i = 1; i <= 127; i++) {

		asm("NOP");
	}
}

/*
 * testr
 * @arg
 *     TypeDefHandleEPY12231 *Handle
 * @return
 *     01 - OK
 *     02 - Error
 */
uint8_t EPY_EPY12231_Test(TypeDefHandleEPY12231 *Handle) {
	uint8_t ReturnBuffer[1];
	if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 10, I2C_TIMEOUT)
			!= HAL_OK) {
		return HAL_ERROR;
	} else {
		return (HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, TEST, 1, ReturnBuffer, 1,
				I2C_TIMEOUT));
	}
}

/*
 * version
 * @arg
 *     TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY12231_Version(TypeDefHandleEPY12231 *Handle, uint8_t *ReturnBuffer) {

	if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
		return HAL_ERROR;
	} else {
		return (HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, VERSION, 1, ReturnBuffer,
				1, I2C_TIMEOUT));
	}

}
/*
```

```
 * status
 * @arg
 *     TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY12231_Status(TypeDefHandleEPY12231 *Handle) {
	uint8_t ReturnBuffer[1];

	if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
		return HAL_ERROR;
	} else {
		HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, FIFO_STATUS, 1, ReturnBuffer, 1,
				I2C_TIMEOUT);
	}
	Handle->InversedStatus = ReturnBuffer[0] & 0x01;
	Handle->FIFO_Count = (ReturnBuffer[0] & 0x1E) >> 1;
	Handle->Error = (ReturnBuffer[0] & 0x30) >> 5;
	Handle->WakeDetected = ReturnBuffer[0] & 0x80 >> 7;

	return 1;
}

/*
 * read full
 * @arg
 *     TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY12231_FIFO_Read_Full(TypeDefHandleEPY12231 *Handle) {

	uint8_t ReturnBuffer[17];
		if(HAL_I2C_IsDeviceReady(Handle->I2C,ADDRESS,2,I2C_TIMEOUT) != HAL_OK){
	 return HAL_ERROR;
	 }else{
	 (HAL_I2C_Mem_Read(Handle-
>I2C,ADDRESS,FIFO_READ_FULL,1,ReturnBuffer,17,I2C_TIMEOUT));

	 Handle->Channel1_CO2 = 0x00 << 24 | ReturnBuffer[0] << 16 | ReturnBuffer[1]
<< 8 | ReturnBuffer[2];
	 Handle->Channel2_CO2 = 0x00 << 24 | ReturnBuffer[3] << 16 | ReturnBuffer[4]
<< 8 | ReturnBuffer[5];
	 Handle->Channel3_CO2 = 0x00 << 24 | ReturnBuffer[6] << 16 | ReturnBuffer[7]
<< 8 | ReturnBuffer[8];
	 Handle->Channel4_CO2 = 0x00 << 24 | ReturnBuffer[9] << 16 | ReturnBuffer[11]
<< 8 | ReturnBuffer[11];
	 Handle->Channel5_CO2 = 0x00 << 24 | ReturnBuffer[12] << 16 | ReturnBuffer[13]
<< 8 | ReturnBuffer[14];
	 Handle->Count = ReturnBuffer[15] << 8 | ReturnBuffer[16];
	 }


}
//
///*
// * FIFO_Read_Active
// * @arg
// *   TypeDefHandleEPY12231 *Handle
```

```
// * @return
// * version
// */
//uint8_t EPY_EPY12231_FIFO_Read_Active(TypeDefHandleEPY12231 *Handle){
//
//    uint8_t TxBuffer[17];
//    uint8_t RxBuffer[17];
//    TxBuffer[0] = FIFO_READ_ACTIVE;
//    HAL_I2C_Master_Transmit(&hi2c,ADDRESS,TxBuffer,1,I2C_TIMEOUT);
//    HAL_I2C_Master_Receive(&hi2c,ADDRESS,RxBuffer,17,I2C_TIMEOUT);
//    return (RxBuffer[0]);
//}
//
/*
 * FIFO_Clear
 * @arg
 *    TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY_EPY12231_FIFO_Clear(TypeDefHandleEPY12231 *Handle) {
      uint8_t ReturnBuffer[1];
      if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
            return HAL_ERROR;
      } else {
            (HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, FIFO_CLEAR, 1,
                        ReturnBuffer, 1, I2C_TIMEOUT));
            return ReturnBuffer[0];
      }
}
//
/*
 * FIFO_Reset
 * @arg
 *    TypeDefHandleEPY12231 *Handle
 * @return
 * version
 */
uint8_t EPY_EPY12231_FIFO_Reset(TypeDefHandleEPY12231 *Handle) {
      uint8_t ReturnBuffer[1];
      if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
            return HAL_ERROR;
      } else {
            HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, FIFO_RESET, 1,
                        ReturnBuffer, 1, I2C_TIMEOUT);

             return ReturnBuffer[0];
      }
}

// * FIFO_CH_Read
// * @arg
// *   TypeDefHandleEPY12231 *Handle
// * @return
// * version
//
uint8_t EPY12231_CH_Read(TypeDefHandleEPY12231 *Handle) {
      uint8_t ReturnBuffer[5];
      if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
```

```
			return HAL_ERROR;
		} else {
			if ((HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, CH_READ, 1, ReturnBuffer, 5,
					I2C_TIMEOUT)) != HAL_OK) {
				return HAL_ERROR;
			} else {
				Handle->Channel_1_Status = (ReturnBuffer[0] & 0x01);
				Handle->Channel_1_FeedbackCapacitorSelection = (ReturnBuffer[0]
						& 0x0E) >> 1;
				Handle->Channel_1_HighPassSignalFilterFrequencySelection =
						(ReturnBuffer[0] & 0x30) >> 4;
				Handle->Channel_1_FrontEndTransconductanceSelection =
						(ReturnBuffer[0] & 0xC0) >> 6;

				Handle->Channel_2_Status = (ReturnBuffer[1] & 0x01);
				Handle->Channel_2_FeedbackCapacitorSelection = (ReturnBuffer[1]
						& 0x0E) >> 1;
				Handle->Channel_2_HighPassSignalFilterFrequencySelection =
						(ReturnBuffer[1] & 0x30) >> 4;
				Handle->Channel_2_FrontEndTransconductanceSelection =
						(ReturnBuffer[1] & 0xC0) >> 6;

				Handle->Channel_3_Status = (ReturnBuffer[2] & 0x01);
				Handle->Channel_3_FeedbackCapacitorSelection = (ReturnBuffer[2]
						& 0x0E) >> 1;
				Handle->Channel_3_HighPassSignalFilterFrequencySelection =
						(ReturnBuffer[2] & 0x30) >> 4;
				Handle->Channel_3_FrontEndTransconductanceSelection =
						(ReturnBuffer[2] & 0xC0) >> 6;

				Handle->Channel_4_Status = (ReturnBuffer[3] & 0x01);
				Handle->Channel_4_FeedbackCapacitorSelection = (ReturnBuffer[3]
						& 0x0E) >> 1;
				Handle->Channel_4_HighPassSignalFilterFrequencySelection =
						(ReturnBuffer[3] & 0x30) >> 4;
				Handle->Channel_4_FrontEndTransconductanceSelection =
						(ReturnBuffer[3] & 0xC0) >> 6;

				Handle->Channel_5_Status = (ReturnBuffer[4] & 0x01);
				Handle->Channel_5_FeedbackCapacitorSelection = (ReturnBuffer[4]
						& 0x0E) >> 1;
				Handle->Channel_5_HighPassSignalFilterFrequencySelection =
						(ReturnBuffer[4] & 0x30) >> 4;
				Handle->Channel_5_FrontEndTransconductanceSelection =
						(ReturnBuffer[4] & 0xC0) >> 6;
				return HAL_OK;
			}

		}

		return HAL_OK;
}

// * FIFO_CH_Write
// * @arg
// *   TypeDefHandleEPY12231 *Handle
// * @return
// * version
//
```

```
uint8_t EPY12231_CH_Write(TypeDefHandleEPY12231 *Handle) {
	uint8_t ReturnBuffer[5];
	if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
		return HAL_ERROR;
	} else {
		ReturnBuffer[0] = (Handle->Channel_1_FrontEndTransconductanceSelection)
				<< 6
				| (Handle-
>Channel_1_HighPassSignalFilterFrequencySelection) << 4
				| (Handle->Channel_1_FeedbackCapacitorSelection) << 1
				| (Handle->Channel_1_Status);

		ReturnBuffer[1] = (Handle->Channel_2_FrontEndTransconductanceSelection)
				<< 6
				| (Handle-
>Channel_2_HighPassSignalFilterFrequencySelection) << 4
				| (Handle->Channel_2_FeedbackCapacitorSelection) << 1
				| (Handle->Channel_2_Status);

		ReturnBuffer[2] = (Handle->Channel_3_FrontEndTransconductanceSelection)
				<< 6
				| (Handle-
>Channel_3_HighPassSignalFilterFrequencySelection) << 4
				| (Handle->Channel_3_FeedbackCapacitorSelection) << 1
				| (Handle->Channel_3_Status);

		ReturnBuffer[3] = (Handle->Channel_4_FrontEndTransconductanceSelection)
<< 6
				| (Handle-
>Channel_4_HighPassSignalFilterFrequencySelection) << 4
				| (Handle->Channel_4_FeedbackCapacitorSelection) << 1
				| (Handle->Channel_4_Status);

		ReturnBuffer[4] = (Handle->Channel_5_FrontEndTransconductanceSelection)
<< 6
				| (Handle-
>Channel_5_HighPassSignalFilterFrequencySelection) << 4
				| (Handle->Channel_5_FeedbackCapacitorSelection) << 1
				| (Handle->Channel_5_Status);

		return (HAL_I2C_Mem_Write(Handle->I2C, ADDRESS, CH_WRITE, 1,
				ReturnBuffer, 5, I2C_TIMEOUT));
	}
	return HAL_OK;
}

uint8_t EPY12231_Ana_Read(TypeDefHandleEPY12231 *Handle) {
	uint8_t ReturnBuffer[2];
	if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
		return HAL_ERROR;
	} else {
		if ((HAL_I2C_Mem_Read(Handle->I2C, ADDRESS, ANA_READ, 1, ReturnBuffer,
				2, I2C_TIMEOUT)) != HAL_OK) {
			return HAL_ERROR;
		} else {
			Handle->SamplingRate = (ReturnBuffer[0]);
			Handle->INTEnable = (ReturnBuffer[1] & 0x01);
			Handle->SYNC = (ReturnBuffer[1] & 0x04) >> 2;
			Handle->CLK_OUT = (ReturnBuffer[1] & 0x08) >> 3;
```

```
                    Handle->LowPassSignalFilterFrequencySelection = (ReturnBuffer[1]
                                & 0x30) >> 4;
                    Handle->HP = (ReturnBuffer[1] & 0x40) >> 6;
                    Handle->LP = (ReturnBuffer[1] & 0x80) >> 7;
              }
       }
       return HAL_OK;
}
uint8_t EPY12231_Ana_Write(TypeDefHandleEPY12231 *Handle) {
       uint8_t WriteBuffer[2];
       if (HAL_I2C_IsDeviceReady(Handle->I2C, ADDRESS, 2, I2C_TIMEOUT) != HAL_OK) {
              return HAL_ERROR;
       } else {

                    WriteBuffer[0] = Handle->SamplingRate;
                    WriteBuffer[1] = Handle->LP << 7 | Handle->HP << 6
                                | Handle->LowPassSignalFilterFrequencySelection << 4
                                | Handle->CLK_OUT << 3 | Handle->SYNC << 2
                                | Handle->INTEnable;

                    Handle->INTEnable = (WriteBuffer[1] & 0x01);
                    Handle->SYNC = (WriteBuffer[1] & 0x04) >> 2;
                    Handle->CLK_OUT = (WriteBuffer[1] & 0x08) >> 3;
                    Handle->LowPassSignalFilterFrequencySelection = (WriteBuffer[1]
                                & 0x30) >> 4;
                    Handle->HP = (WriteBuffer[1] & 0x40) >> 6;
                    Handle->LP = (WriteBuffer[1] & 0x80) >> 7;

                    HAL_I2C_Mem_Write(Handle->I2C, ADDRESS, ANA_WRITE, 1,
WriteBuffer,
                                          2, I2C_TIMEOUT);

       }
       return HAL_OK;
}
```

P8 Biblioteka Modbus RTU

```
/*
 * modbuslite.c
 *
 *  Created on: Oct 14, 2019
 *      Author: Josip
 */
#include "modbuslite.h"
#include "main.h"

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
//x=(~y)+1 - dual complemetn
//make changes as you need

void MODBUS_UART_Init(Modbus_HandleTypeDef* Modbus)
{


  /* USER CODE BEGIN USART3_Init 0 */
       GPIO_InitTypeDef GPIO_InitStruct = {0};
       /*Configure GPIO pin : CONTROL_Pin */
         GPIO_InitStruct.Pin = Modbus->GPIO_Control_Pin;
         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
         //GPIO_InitStruct.Pull = GPIO_PULLUP;
         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
         HAL_GPIO_Init(Modbus->GPIOx, &GPIO_InitStruct);
  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */

         ModbusCOnfigComm(Modbus);

  /* USER CODE BEGIN USART3_Init 2 */
  HAL_GPIO_WritePin(Modbus->GPIOx, Modbus->GPIO_Control_Pin, 0);
    /* USER CODE END USART3_Init 2 */

}

void
MODBUS_RECIEVE_DMA(Modbus_HandleTypeDef* Modbus){
       int timeout = 1000000;
       while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_IDLE) != 1)
       {
              timeout--;
              if(timeout<= 0) break;
       }
       HAL_UART_Receive_DMA(Modbus->UART,MODBUS_RX_BUFFER,8);

}

void
```

```
MODBUS_SLAVE(Modbus_HandleTypeDef* Modbus)
{
	MODBUS_POOL++;
	MODBUS_RECIEVE_DMA(Modbus);

	if(MODBUS_RX_BUFFER[0] == Modbus->mb_address && CRC16(MODBUS_RX_BUFFER,8) ==
CRC_OK){
		MODBUS_OK++;
		if( (MODBUS_RX_BUFFER[1] == READ_HOLDING_REGISTER &&
(MODBUS_RX_BUFFER[2] << 8 | MODBUS_RX_BUFFER[3]) < NUM_OF_HOLDING_REGISTERS)||
			(MODBUS_RX_BUFFER[1] == WRITE_HOLDING_REGISTER &&
(MODBUS_RX_BUFFER[2] << 8 | MODBUS_RX_BUFFER[3]) < NUM_OF_HOLDING_REGISTERS) ||
			(MODBUS_RX_BUFFER[1] == READ_INPUT_REGISTER &&
(MODBUS_RX_BUFFER[2] << 8 | MODBUS_RX_BUFFER[3]) < NUM_OF_INPUT_REGISTERS)){

			HAL_GPIO_WritePin(Modbus->GPIOx, Modbus->GPIO_Control_Pin, 1);
	//transmit

			switch (MODBUS_RX_BUFFER[1]){
			case READ_HOLDING_REGISTER:
				if((MODBUS_RX_BUFFER[4] << 8 | MODBUS_RX_BUFFER[5]) == 1){


	READ_ONE_HOLDING_REGISTER_RESPONSE(Modbus,MODBUS_RX_BUFFER[2] << 8 |
MODBUS_RX_BUFFER[3], MODBUS_RX_BUFFER[4] << 8 | MODBUS_RX_BUFFER[5]);
					CLEAR_RX_BUFFER();
				}
				else {

	READ_HOLDING_REGISTER_RESPONSE(Modbus,MODBUS_RX_BUFFER[2] << 8 |
MODBUS_RX_BUFFER[3], MODBUS_RX_BUFFER[4] << 8 | MODBUS_RX_BUFFER[5]);
					CLEAR_RX_BUFFER();
				}
				break;

			case WRITE_HOLDING_REGISTER:

	WRITE_HOLDING_REGISTER_RESPONSE(Modbus,(uint16_t)(MODBUS_RX_BUFFER[2] << 8 |
MODBUS_RX_BUFFER[3]));

					CLEAR_RX_BUFFER();
					break;

			case READ_INPUT_REGISTER:
					READ_INPUT_REGISTER_RESPONSE(Modbus,MODBUS_RX_BUFFER[2] <<
8 | MODBUS_RX_BUFFER[3], MODBUS_RX_BUFFER[4] << 8 | MODBUS_RX_BUFFER[5]);
					CLEAR_RX_BUFFER();
					break;
			}

		}else
		{
			asm("NOP");
		}
	}
	else {
		asm("NOP");

	}
```

```
	HAL_GPIO_WritePin(Modbus->GPIOx, Modbus->GPIO_Control_Pin, 0);   //listen

}

uint16_t CHECK_CRC(unsigned char *puchMsg)
{
	unsigned short msgSize = ((sizeof(puchMsg) / 1) - 2);
	if((HIGH_BYTE(CRC16(MODBUS_RX_BUFFER,msgSize)) << 8 |
LOW_BYTE(CRC16(MODBUS_RX_BUFFER,msgSize))) != CRC16(MODBUS_RX_BUFFER, msgSize)){

		return CRC_NOK;
	}
	return CRC_OK;

}

uint8_t
LOW_BYTE (uint16_t NUMBER)
{
  return (NUMBER & 0x00FF);
}

uint8_t
HIGH_BYTE (uint16_t NUMBER)
{
  return (NUMBER >> 8);
}


/*
 *
 *
 */
void
READ_ONE_HOLDING_REGISTER_RESPONSE(Modbus_HandleTypeDef* Modbus,uint16_t
REGISTER_ADDRESS, uint16_t NUM_OF_REGS)
{
if((REGISTER_ADDRESS + NUM_OF_REGS) <= NUM_OF_HOLDING_REGISTERS){
		unsigned char MODBUS_TX_BUFFER[7];

		uint16_t CRC_;

		MODBUS_TX_BUFFER[0] = Modbus->mb_address;
		MODBUS_TX_BUFFER[1] = READ_HOLDING_REGISTER;
		MODBUS_TX_BUFFER[2] = 0x02;

		MODBUS_TX_BUFFER[3] = (Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]) >> 8;
		MODBUS_TX_BUFFER[4] = (Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]) &
0x00FF;

		CRC_ = CRC16(MODBUS_TX_BUFFER,5);

		MODBUS_TX_BUFFER[5] = LOW_BYTE(CRC_);
		MODBUS_TX_BUFFER[6] = HIGH_BYTE(CRC_);

		asm("NOP");
		HAL_UART_Transmit_DMA(Modbus->UART,MODBUS_TX_BUFFER,7);
		while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_TC) != 1);
```

```
            //CLEAR_RX_BUFFER();

}else{
      //CLEAR_RX_BUFFER();
}
}

/*
 *
 *
 */
void
READ_HOLDING_REGISTER_RESPONSE(Modbus_HandleTypeDef* Modbus,uint16_t
REGISTER_ADDRESS, uint16_t NUM_OF_REGS)
{
if((REGISTER_ADDRESS + NUM_OF_REGS) <= NUM_OF_HOLDING_REGISTERS){
            unsigned char MODBUS_TX_BUFFER[5 + (NUM_OF_REGS * 2)];

            unsigned char buff_size = sizeof(MODBUS_TX_BUFFER);

            uint16_t CRC_;
            unsigned char hi = 3,lo = 4;

            MODBUS_TX_BUFFER[0] = Modbus->mb_address;
            MODBUS_TX_BUFFER[1] = READ_HOLDING_REGISTER;
            MODBUS_TX_BUFFER[2] = NUM_OF_REGS * 2;
            for(int i = REGISTER_ADDRESS ; i < (REGISTER_ADDRESS + NUM_OF_REGS);
i++){

            MODBUS_TX_BUFFER[hi] = HIGH_BYTE(Modbus->HOLDING_REGISTER[i]);
            MODBUS_TX_BUFFER[lo] = LOW_BYTE(Modbus->HOLDING_REGISTER[i]);

            hi += 2;
            lo += 2;
            }

            CRC_ = CRC16(MODBUS_TX_BUFFER,buff_size - 2);

            MODBUS_TX_BUFFER[buff_size - 2] = LOW_BYTE(CRC_);
            MODBUS_TX_BUFFER[buff_size - 1] = HIGH_BYTE(CRC_);

            asm("NOP");
            HAL_UART_Transmit_DMA(Modbus->UART,MODBUS_TX_BUFFER,buff_size);
            while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_TC) != 1);
            //CLEAR_RX_BUFFER();

}else{
      //CLEAR_RX_BUFFER();
}
}

/*
 *
 *
 */
void
WRITE_HOLDING_REGISTER_RESPONSE(Modbus_HandleTypeDef* Modbus,uint16_t
REGISTER_ADDRESS)
{
```

```
	unsigned char MODBUS_TX_BUFFER[8];
	uint16_t CRC_;
	Modbus->HOLDING_REGISTER[REGISTER_ADDRESS] =(uint16_t) (MODBUS_RX_BUFFER[4] <<
8 | MODBUS_RX_BUFFER[5]);


	MODBUS_TX_BUFFER[0] = Modbus->mb_address;
	MODBUS_TX_BUFFER[1] = WRITE_HOLDING_REGISTER;
	MODBUS_TX_BUFFER[2] = HIGH_BYTE(REGISTER_ADDRESS);
	MODBUS_TX_BUFFER[3] = LOW_BYTE(REGISTER_ADDRESS);
	MODBUS_TX_BUFFER[4] = HIGH_BYTE(Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]);
	MODBUS_TX_BUFFER[5] = LOW_BYTE(Modbus->HOLDING_REGISTER[REGISTER_ADDRESS]);
	CRC_ = CRC16(MODBUS_TX_BUFFER,sizeof(MODBUS_TX_BUFFER) - 2);
	MODBUS_TX_BUFFER[6] = LOW_BYTE(CRC_);
	MODBUS_TX_BUFFER[7] = HIGH_BYTE(CRC_);

	HAL_UART_Transmit_DMA(Modbus->UART,MODBUS_TX_BUFFER,8);
	while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_TC) != 1);
	//CLEAR_RX_BUFFER();

}

/*
 *
 *
 */
void
READ_INPUT_REGISTER_RESPONSE(Modbus_HandleTypeDef* Modbus, uint16_t REGISTER_ADDRESS,
uint16_t NUM_OF_REGS)
{

		if((REGISTER_ADDRESS + NUM_OF_REGS) <= NUM_OF_INPUT_REGISTERS){
			unsigned char MODBUS_TX_BUFFER[5 + (NUM_OF_REGS * 2)];

			unsigned char buff_size = sizeof(MODBUS_TX_BUFFER);

			uint16_t CRC_;
			int cnt = 0;
			unsigned char hi = 3,lo = 4;

			MODBUS_TX_BUFFER[0] = Modbus->mb_address;
			MODBUS_TX_BUFFER[1] = READ_INPUT_REGISTER;
			MODBUS_TX_BUFFER[2] = NUM_OF_REGS * 2;
			for(int i = REGISTER_ADDRESS ; i < (REGISTER_ADDRESS +
NUM_OF_REGS); i++){
			cnt++;
			cnt += REGISTER_ADDRESS;

			MODBUS_TX_BUFFER[hi] = HIGH_BYTE(Modbus-
>HOLDING_REGISTER[i]);
			MODBUS_TX_BUFFER[lo] = LOW_BYTE(Modbus-
>HOLDING_REGISTER[i]);
			hi = hi + 2;
			lo = lo + 2;
			}
			CRC_ = CRC16(MODBUS_TX_BUFFER,buff_size - 2);
			MODBUS_TX_BUFFER[buff_size - 2] = LOW_BYTE(CRC_);
			MODBUS_TX_BUFFER[buff_size - 1] = HIGH_BYTE(CRC_);
```

```
                        HAL_UART_Transmit_DMA(Modbus-
>UART,MODBUS_TX_BUFFER,buff_size);
                        while(__HAL_UART_GET_FLAG(Modbus->UART, UART_FLAG_TC) !=
1);
                        //CLEAR_RX_BUFFER();

            }else{
                  //CLEAR_RX_BUFFER();

            }

}

/*
 *
 *
 */
void
ERROR_RESPONSE(Modbus_HandleTypeDef* Modbus, unsigned char CODE_IN_REQUEST)
{
      uint8_t MODBUS_TX_BUFFER[5];
      uint8_t temp_error_code;
      uint16_t CRC_;

switch(CODE_IN_REQUEST)
{
      case READ_COIL_STATUS:
                  temp_error_code = _ERROR_READ_COIL_STATUS_;
                  break;

      case READ_INPUT_STATUS:
                  temp_error_code = _ERROR_READ_INPUT_STATUS_;
                  break;

      case READ_HOLDING_REGISTER:
                  temp_error_code = _ERROR_READ_HOLDING_REGISTER_;
                  break;

      case READ_INPUT_REGISTER:
                  temp_error_code = _ERROR_READ_INPUT_REGISTER_;
                  break;

      case FORCE_SINGLE_COIL:
                  temp_error_code = _ERROR_FORCE_SINGLE_COIL_;
                  break;

      case WRITE_HOLDING_REGISTER:
                  temp_error_code = _ERROR_WRITE_HOLDING_REGISTER_;
                  break;

      case WRITE_MULTIPLE_COILS:
                  temp_error_code = _ERROR_WRITE_MULTIPLE_COILS_;
                  break;

      case WRITE_MULTIPLE_REGISTERS:
                  temp_error_code = _ERROR_WRITE_MULTIPLE_REGISTERS_;
                  break;

}
```

```
	MODBUS_TX_BUFFER[0] = Modbus->mb_address;		//Device address
	MODBUS_TX_BUFFER[1] = HIGH_BYTE(temp_error_code);
	MODBUS_TX_BUFFER[2] = 0x01;
	CRC_ = CRC16(MODBUS_TX_BUFFER, sizeof(MODBUS_TX_BUFFER)-2);
	MODBUS_TX_BUFFER[3] = LOW_BYTE(CRC_);
	MODBUS_TX_BUFFER[4] = HIGH_BYTE(CRC_);
	HAL_UART_Transmit(Modbus-
>UART,MODBUS_TX_BUFFER,sizeof(MODBUS_TX_BUFFER),TRANSMIT_DELAY);
	//CLEAR_RX_BUFFER();
}

void CLEAR_RX_BUFFER(){
	MODBUS_RX_BUFFER[0] = 0;
	MODBUS_RX_BUFFER[1] = 0;
	MODBUS_RX_BUFFER[2] = 0;
	MODBUS_RX_BUFFER[3] = 0;
	MODBUS_RX_BUFFER[4] = 0;
	MODBUS_RX_BUFFER[5] = 0;
	MODBUS_RX_BUFFER[6] = 0;
	MODBUS_RX_BUFFER[7] = 0;

}

void ModbusCOnfigComm(Modbus_HandleTypeDef* Modbus){

if (1 == Modbus->mb_baud){
	Modbus->UART->Init.BaudRate = 4800;
}
else if(2 == Modbus->mb_baud){
	Modbus->UART->Init.BaudRate = 19200;
}
else if(3 == Modbus->mb_baud){
	Modbus->UART->Init.BaudRate = 38400;
}
else if(4 == Modbus->mb_baud){
	Modbus->UART->Init.BaudRate = 57600;
}
else if(5 == Modbus->mb_baud){
	Modbus->UART->Init.BaudRate = 115200;
}
else{
	Modbus->UART->Init.BaudRate = 9600;
}

if(1 == Modbus->mb_parity){
  Modbus->UART->Init.Parity = UART_PARITY_EVEN;
  Modbus->UART->Init.WordLength = UART_WORDLENGTH_9B;
  Modbus->UART->Init.StopBits = UART_STOPBITS_1;
  }
  else if (2 == Modbus->mb_parity){
  Modbus->UART->Init.Parity = UART_PARITY_ODD;
  Modbus->UART->Init.WordLength = UART_WORDLENGTH_9B;
  Modbus->UART->Init.StopBits = UART_STOPBITS_1;
  }
  else{
  Modbus->UART->Init.Parity = UART_PARITY_NONE;
  Modbus->UART->Init.WordLength = UART_WORDLENGTH_8B;
  Modbus->UART->Init.StopBits = UART_STOPBITS_1;
```

```
  }

  Modbus->UART->Instance = USART2; /********************** PAZI tu
***************************/
  Modbus->UART->Init.Mode = UART_MODE_TX_RX;
  Modbus->UART->Init.HwFlowCtl = UART_HWCONTROL_NONE;
  Modbus->UART->Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(Modbus->UART) != HAL_OK)
  {
        //while(1);
  }

}

void ModbusPoll(Modbus_HandleTypeDef* Modbus){
      //pool every 50ms

      long elapsedTime = 0;
      static long lastExecution = 0;
      static int taskDelay = 1;


      elapsedTime = HAL_GetTick();
      if(elapsedTime - lastExecution >= taskDelay){
            lastExecution = HAL_GetTick();
            elapsedTime = HAL_GetTick();
            MODBUS_SLAVE(Modbus);
      }

}
```

P9 PI regulator

```
  float PID_Regulator(bool PID_Enable, float PID_SP,float PID_PV, float Kp, float
Ki,float Kd){

            const float PID_Min = 4000; // min 0-255
            const float PID_Max = 10000; // max 0-65535
            const float Ts = 0.1; // sampling time in s
            const float epsilon = 5.0;

            static float integral;
            float proportional, derivative;
            static float previous_error, error,  PID_out;
            if(PID_Enable){

                  error = (float)PID_SP - (float)PID_PV;

                  global_err = error;

                  proportional = Kp * (error);
                  if(abs(error) > epsilon) {
                        integral = integral + ((Kp * Ts) / Ki) * error;
                        }
                  derivative = Kd * (error - previous_error);

                  global_p = proportional;
                  global_i = integral;
                  global_derivative = derivative;

                  PID_out = (PID_out + proportional + integral + derivative);

                  previous_error = error;
                  global_err_1  = previous_error;

                  if(PID_out < PID_Min){
                        PID_out = PID_Min;
                  }
                  else if(PID_out > PID_Max){
                        PID_out = PID_Max;
                  }
                  return PID_out;
                  }
                  else {
                  return PID_out = 0;
                  }


      }
```