Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primož Skubic

Zajem podatkov s periferno napravo na industrijskem vodilu Modbus

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00358/2012
Datum: 04.12.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PRIMOŽ SKUBIC**

Naslov: **ZAJEM PODATKOV S PERIFERNO NAPRAVO NA INDUSTRIJSKEM VODILU MODBUS**

DATA ACQUISITION WITH MODBUS PERIPHERAL INTERFACE UNIT

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Načrtujte vgrajen sistem, ki kot periferna naprava na vodilu Modbus izvaja meritve temperature in vlage v sistemih ogrevanja in v industrijskem okolju ter izmerjene vrednosti posreduje nadzornemu oziroma krmilnemu sistemu. Vgrajen sistem naj bo zgrajen okrog mikrokrmilnika PIC 16F1827 in naj bo z uporabo standarda RS-485 in protokola Modbus povezan z nadzornim sistemom. Za spreminjanje nastavitev delovanja vgrajenega sistema razvijte programsko opremo v okolju .NET. Za prikazovanje izmerjenih vrednosti uporabite prikazovalnik Fatek FV035ST-C10.

Mentor:
izr. prof. dr.Patricio Bulić

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Primož Skubic, z vpisno številko **63060286**, sem avtor diplomskega dela z naslovom:

Zajem podatkov s periferno napravo na industrijskem vodilu Modbus

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14. marca 2013 Podpis avtorja:

Zahvaljujem se vsem, ki so mi na poti do diplome stali ob strani, še posebej pa svoji družini za vso podporo in potrpežljivost v času študija. Zahvaljujem se Karin za vse nasvete, pomoč in oporo. Zahvaljujem se tudi Marjanu, Milanu, Martinu in Matjažu za vse razlage, opozorila in kritike ter Tjaši, za njen boj z vejicami. Prof. dr. Patrico Bulić pa si zasluži posebno zahvalo za odličnega mentorja in pedagoga; hvala za motivacijo in vso pomoč v času študija.

Kazalo vsebine

Kazalo slik

Kazalo tabel

Seznam uporabljenih kratic in simbolov

ACK	*Acknowledge* – potrditev
ADC	*Analog to Digital Converter* – analogno-digitalni pretvornik
ARM	*Advanced RISC Machine* – napredna RISC naprava
ASCII	*American Standard Code for Information Interchange* – standard za zapis znakov
CR	*Carriage Return* – ukaz za postavitev na začetek vrstice
CRC	*Cyclic Redundancy Check* – algoritem za izračun kontrolne vsote
DF	*Data Fetch* – zahteva za branje podatkov
EEPROM	*Electrically Erasable Programmable Read-Only Memory* – obstojen pomnilnik
EUSART	*Enhanced Universal Asynchronous Receiver Transmitter* – enota za prejemanje in pošiljanje podatkov
FVR	*Fixed Voltage Reference* – stalna referenčna napetost
GND	*ground* – ozemljitev
HMI	*Human-Machine Interface* – vmesnik človek-stroj
I^2C	*Inter-Integrated Circuit* – serijski vmesnik in protokol
ICD	*In Circuit Debugger*
ICE	*In Circuit Emulator*
ICSP	*In-Circuit Serial Programming* – protokol za povezavo mikrokrmilnika in programatorja

IDE *Integrated Development Environment* – razvojno okolje

ISR *Interrupt Service Routine* gl. PSP

LF *Line Feed* – ukaz za zečetek nove vrstice

LSB *Least Significant Bit* – najmanj pomembni bit

MR *Measure Request* – zahteva meritve

MSB *Most Significant Bit* – najpomembnejši bit

MSSP *Master Synchronous Serial Port*
– enota za krmiljenje serijske komunikacije

NACK *Not Acknowledge* – zavrnitev

PDU *Protocol Data Unit* – podatkovna enota protokola

PLC *Programmable Logic Controller* – programabilni logični krmilnik

PPM *Points Per Million*

PSP Prekinitvneno-servisni program

PTAT *Proportional To Absolute Temperature*
– proporcionalno z absolutno temperaturo

RI pretvarjanje električne upornosti v električni tok
(ang. *RC – Resistance to Current*)

RISC *Reduced Instruction Set Computing*

RTU *Remote Terminal Unit* – oddaljeni terminal

RX *Receive* – sprejem

SCL *Serial Clock* – ura serijskega vodila

SDA *Serial Data* – podatki serijskega vodila

SPI *Serial Peripheral Interface* – serijski vmesnik in protokol

TX *Transmit* – oddajanje

UART *Universal Asynchronous Receiver Transmitter*
– univerzalni asinhroni sprejemnik/oddajnik

VS vgrajen sistem (ang. *Embedded System*)

Povzetek

Vsako krmiljenje se začne z meritvami trenutnega stanja in prilagajanjem delovanja glede na izmerjene vrednosti. V ta namen smo v sklopu diplomske naloge zasnovali vgrajen sistem, ki izvaja meritve temperature in vlage ter izmerjene vrednosti posreduje nadzornemu oziroma krmilnemu sistemu. Vgrajen sistem je zgrajen okrog mikrokrmilnika PIC 16F1827, ki je povezan z gonilnikom za serijsko komunikacijo, digitalnim tipalom temperature in vlage HYT 271 ter pretvornikom RI. Za spreminjanje nastavitev delovanja vgrajenega sistema smo razvili programsko opremo z uporabo visokonivojskega programskega jezika C#. Vgrajen sistem smo na koncu z uporabo standarda RS-485 in protokola Modbus povezali z nadzornim sistemom. Uporabili smo prikazovalnik Fatek FV035ST-C10, ki kot gospodar vodila prikazuje rezultate meritev vgrajenega sistema.

Ključne besede:

vgrajen sistem, mikrokrmilnik, analogno-digitalna pretvorba, Modbus, plavajoča vejica

Abstract

Every process control begins with measurements of current state and adjustments that depend on measured values. For the purpose of this thesis an embedded system was designed, that can measure both temperature and relative humidity. The system is able to provide its measurements to the monitoring or controlling system. The main component of the embedded system is a PIC 16F1827 microcontroller connected to a RS-485 driver that provides for serial communication features, digital sensor HYT 271 for temperature and humidity measurement and a RC converter (RC – electrical resistance to electrical current). A complementary application for configuration of the embedded system was also developed using the C# programming language. In the final phase, our embedded system was connected to a HMI display by RS-485 standard and Modbus protocol. The display used for this purpose was a Fatek FV035ST-C10, which acted as a bus master, displaying the measurements taken by the embedded system.

Key words:

embedded system, microcontroller, analog-to-digital conversion, Modbus, floating point

Poglavje 1

Uvod

V času, ko je vse bolj poudarjeno varčevanje z energijo in njen čimboljši izkoristek, se je povečala uporaba mehanizmov in sredstev za nadzor in uravnavanje delovanja energijskih sistemov. Ker so eni od teh tudi toplotni sistemi, so dobrodošli mehanizmi na področju boljšega izkoristka energije pri regulaciji temperature. Čeprav so programabilni logični krmilniki (PLC, ang. *Programmable Logic Controller*) za krmiljenje ogrevalnih sistemov cenovno relativno dostopni, se pojavijo težave, ko je potrebno na krmilnik priklopiti temperaturno tipalo. V mnogih primerih so razširitveni moduli za krmilnike s podporo priklopa analognih temperaturnih tipal celo dražji od samega krmilnika. Vendar pa vsak moderen krmilnik podpira komunikacijo z drugimi napravami prek standarda RS-485 z uporabo vodila Modbus. Ker na en serijski vhod z uporabo vodila Modbus lahko priklopimo do 247 naprav, to izniči potrebo po uporabi razširitvenega modula. To je bilo naše vodilo pri izdelavi vgrajenega sistema; izdelati cenovno dostopen vgrajen sistem, ki bo krmilniku posredoval izmerjene temperature in vlago v opazovanem sistemu, na podlagi katerih bo krmilnik vodil proces ogrevanja oziroma proces, ki za svoje delovanje potrebuje meritve temperature oziroma vlage.

Prvi del diplomskega dela je posvečen izdelavi vgrajenega sistema, zgrajenega okrog mikrokmilnika PIC. Na začetku si bomo ogledali strojne komponente, uporabljene pri izdelavi vgrajenega sistema. Opisali bomo tudi pripomočke, ki neposredno ne vplivajo na delovanje sistema, vendar pa so ključnega pomena za samo izdelavo. Sledi opis razvojnega okolja MPLAB IDE, ki je bilo uporabljeno v fazi razvoja. V nadaljevanju bomo podrobneje pregledali funkcije, ki jih vgrajen sistem podpira, razložili njihovo delovanje in postopke, uporabljene za razvoj posamezne funkcije.

V drugem delu diplomske naloge bomo pozornost preusmerili na aplikacijo, s katero nastavljamo delovanje vgrajenega sistema oziroma preverjamo, ali sistem pravilno deluje. Ker je aplikacija spisana v programskem jeziku C#, bomo nekaj besed namenili tudi opisu razvojnega okolja Visual Studio 2012. Nato se bomo posvetili aplikaciji in razložili glavne lastnosti obeh načinov delovanja ter opisali najpomembnejše metode oziroma gradnike, uporabljene v aplikaciji.

Zadnji del naloge je namenjen opisu nadzornega sistema, ki uporabniku prikazuje rezultate meritev, ki jih je izvedel vgrajen sistem. Na kratko bomo opisali razvojno okolje PM Designer, ter delovanje nadzornega sistema.

Na koncu bomo opisali glavne težave, ki so se pojavile med razvojem, in navedli tudi nekaj možnih izboljšav vgrajenega sistema.

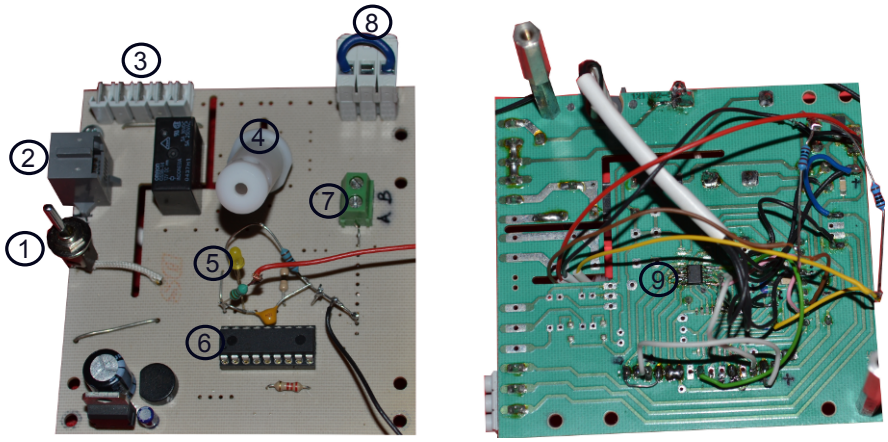
Poglavje 2

Vgrajen sistem

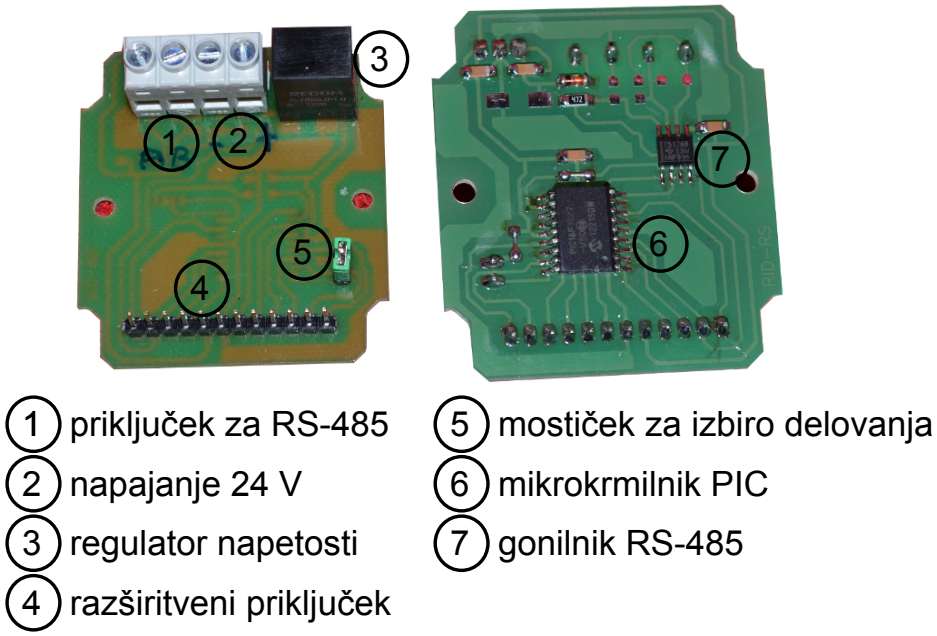
Vgrajen sistem (v nadaljevanju: VS), izdelan za potrebe te diplomske naloge, je zgrajen okrog mikrokrmilnika PIC 16F1827, proizvajalca Microchip. Omenjeni mikrokrmilnik je bil izbran na podlagi dejstva, da je njegova uporaba izredno razširjena, zanj pa je – zaradi velike skupine razvijalcev – na voljo veliko dokumentacije ter primerov uporabe mikrokrmilnika. Med samim načrtovanjem VS smo spoznali, da – ker gre za mikrokrmilnik z majhnim številom nožic (ang. *pin*) – si funkcije, ki jih mikrokrmilnik podpira, delijo vhodno/izhodne nožice (gl. sliko 2.3). Zato je moral izbrani mikrokrmilnik zadostiti naslednjima zahtevama:

- Mikrokrmilnik mora imeti analogno-digitalni pretvornik (ang. *Analog-to-Digital Converter*), EUSART (ang. *Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter*) ter MSSP (ang. *Master Synchronous Serial Port*) s podporo za I^2C (ang. *Inter-Integrated Circuit*).

- Zgoraj naštete funkcije mikrokrmilnika lahko uporabljamo istočasno in si ne delijo vhodno/izhodnih nožic.



Slika 2.1: Prototipno vezje.

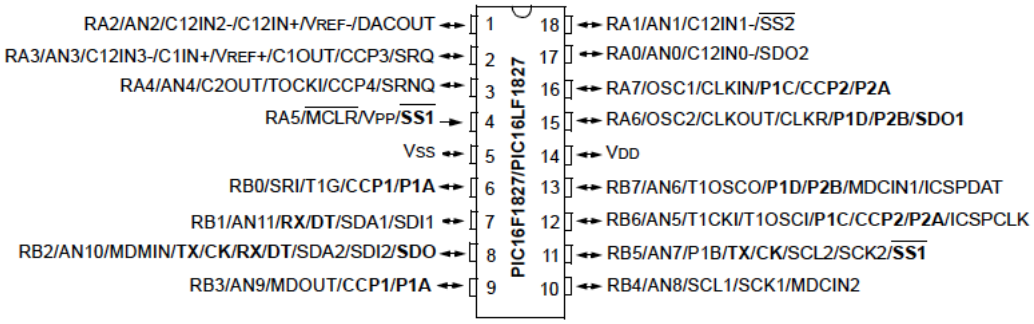


Slika 2.2: Prva različica tiskanega vezja.

2.1 Strojna oprema in komponente

2.1.1 Mikrokrmilnik PIC 16F1827

Mikrokrmilnik PIC 16F1827 je 8-bitni mikrokrmilnik tipa RISC (ang. *Reduced Instruction Set Computing*) z vgrajenim natančnim oscilatorjem in ločenim programskim ter podatkovnim pomnilnikom (Harvard arhitektura). Oba pomnilnika sta izdelana v t. i. "flash" tehnologiji. Funkcije, vgrajene v sam mikrokrmilnik, krmilimo prek t. i. SFR registrov (ang. *Special Function Register*). Na voljo imamo tudi EEPROM ter množico digitalnih oziroma analognih vhodov.



Slika 2.3: Razporeditev vhodov mikrokrmilnika PIC 16F1827 [1].

Specifikacije mikrokrmilnika, pomembne za izdelavo vgrajenega sistema [1]:

- obratovalna napetost: 5 V
- procesor: 8-bitni RISC
- oscilator: 32 MHz (precizni)
- kapaciteta programskega pomnilnika: 7 KB (FLASH)
- kapaciteta podatkovnega pomnilnika: 384 B (FLASH)
- kapaciteta EEPROM pomnilnika: 256 B

- Analogno-digitalni pretvornik: 12 kanalov, 10-bitna resolucija
- Komunikacija s prerifernimi napravami: MSSP, EUSART
- Časovniki: WDT ter štirje poljubno nastavljivi časovniki

2.1.2 MPLAB ICD3

MPLAB ICD3 je programator/razhroščevalnik proizvajalca Microchip in je namenjen programiranju in razhroščevanju vgrajenih sistemov, ki temeljijo na PIC mikrokrmilnik. Za delovanje poleg PIC mikrokrmilnika potrebuje na računalniku nameščeno programsko opremo MPLAB IDE.

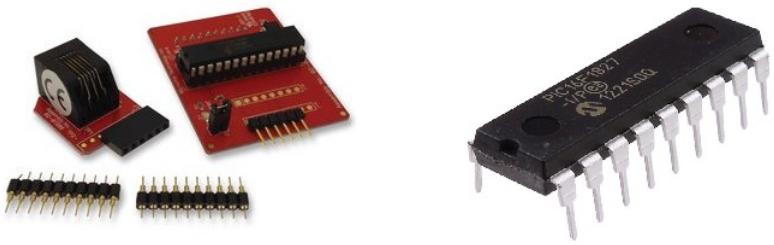
MPLAB ICD3 temelji na ICSP metodi programiranja mikrokrmilnikov. Na samo vezje je povezan prek petih signalov (gl. sliko 2.4). Dva signala sta uporabljena za napajanje ICD-ja, preostali pa se uporabljajo za komunikacijo med mikrokrmilnikom in programatorjem/razhroščevalnikom [2]:

- **Vpp** – Napetost za programiranje (ang. *programming mode voltage*) je signal s katermi sprožimo, da se mikrokrmilnik postavi v način za programiranje. Gre za napetost višjo od 5 V; v našem primeru je bila 9,05 V.
- **ICSPCLK** – Ura serijske komunikacije (ang. *serial clock*) je signal med 0 V in 5 V, ki ga določa programator. Podatki se prenašajo na negativno fronto.
- **ICSPDAT** – Podatki serijske komunikacije (ang. *serial data*) je signal med 0 V in 5 V, ki ga lahko generirata tako programator kot tudi mikrokrmilnik, odvisno od operacije, ki se trenutno izvaja.

Kot smo že omenili, je ena od večjih omejitev našega mikrokmilnika število nožic. Ker se ICD3 povezuje na naš mikrokrmilnik prek treh signalov, to pomeni še dodatno izgubo funkcionalnosti. Prav tako nam naš mikrokrmilnik v

načinu za razhroščevanje ponuja uporabo samo ene prekinitvene točke (ang. *breakpoint*), kar razhroščevanje še oteži. Zato smo se odločili, da za potrebe razhroščevanja uporabimo posebni mikrokrmilnik PIC16F1847-ICE, ki je namenjen prav razhroščevanju. Njegova največja prednost je, da ima signale, potrebne za povezavo z ICD3, ločene od preostalih vhodov/izhodov. Tako ne pride do izgube funkcionalnosti, prav tako pa nam omogoča tudi večje število prekinitvenih točk [3]. Priklop na razvojno ploščico ostane enak, ICD3 pa se priklopi prek priloženega konektorja RJ-11.

Slika 2.4: Primer povezave ICD programatorja.



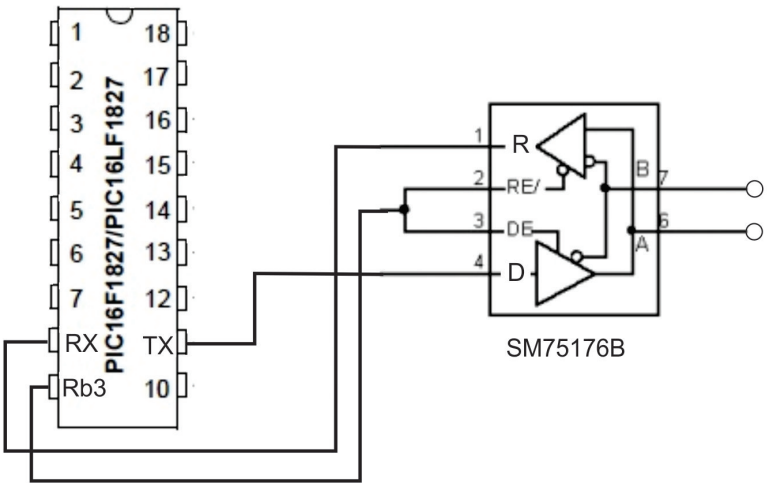
Slika 2.5: PIC16F1847-ICE [3] ter PIC16F1827 [1].

2.1.3 RS485 gonilnik

Ena od zahtev, ki jo mora naš VS izpolnjevati, je komunikacija z zunanjimi napravami prek RS-485. Za to potrebujemo gonilnik (ang. *driver*), ki VS doda

možnost komunikacije s periferijo z uporabo standarda RS-485 [5]. Odločili smo se za *SM75176 differential bus transceiver* proizvajalca Texas Instruments [4]. Iz slike 2.6 je razvidno, da sta gonilnik in mikrokrmilnik povezana z uporabo treh povezav:

- izhod enote UART mikrokrmilnika TX je povezan z vhodom gonilnika D (ang. *drive*),
- izhod gonilnika R (ang. *recive*) je povezan z vhodom RX enote UART,
- krmilni izhod smeri prenosa mikrokrmilnika $RB3$ pa je povezan tako z vhodom gonilnika $\overline{RE}$ (ang. *recive enable*) kot tudi z vhodom DE (ang. *drive enable*).

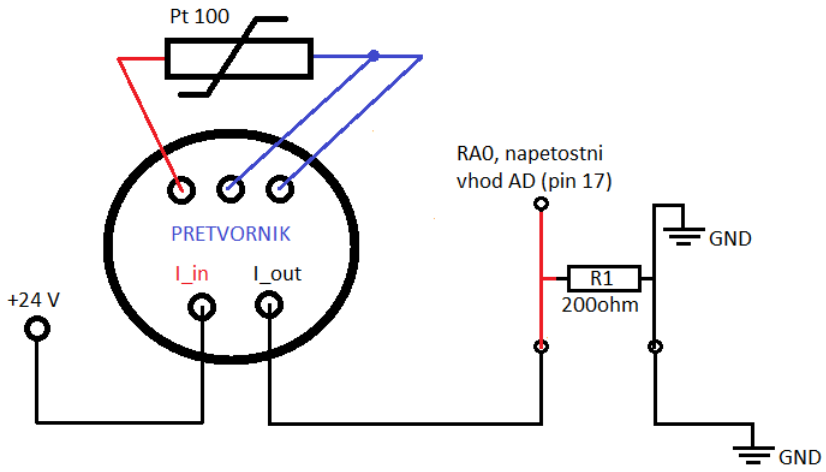


Slika 2.6: Povezava mikrokrmilnika z RS485 gonilnikom.

Ker sta signala $\overline{RE}$ in DE med seboj izključujoča, je možno oba krmiliti samo z enim izhodom mikrokrmilnika, na ta način pa zmanjšamo število povezav. Seveda je potrebno krmilnik priklopiti tudi na napajanje (+5 V) in ozemljitev (GND). Vhoda oziroma izhoda gonilnika A in B pa sta prek sponk povezana z vodilom.

2.1.4 Pretvornik LKM120 in pretvornik MO4201

Poleg mikrokrmilnika je eden glavnih gradnikov našega vgrajenega sistema pretvornik, ki skrbi za pretvarjanje upornosti uporovnega temperaturnega tipala Pt100 v električni tok (4 – 20 mA). Uporabili smo dva različna pretvornika. Njuna edina skupna lastnost je, da spremembo upornosti tipala pretvorita v spremembo električnega toka (RI pretvornik). Izbira pretvornika ne vpiliva na vezavo v sistem (gl. sliko 2.7).



Slika 2.7: Vezava RI pretvornika v sistem.

Pretvornik LKM120 nemškega proizvajalca LKM Electronic je digitalni natančni programabilni pretvornik. Ima majhno linearno napako meritve (0,1 %), nastavljivo merilno območje (od –200 °C do +600 °C) in podpira tudi tipala tipa Ni100. Njegova najpomembnejša lastnost pa je, da gre za digitalni pretvornik, ki svoj izhod povprečuje. S tem se nekoliko zmanjša odzivnost, vendar pa se obenem poveča robustnost [6].

Pretvornik MO4201 slovenskega proizvajalca Elektronika Pahor je analogni neprogramabilni pretvornik z možnostjo korekcije. Nanj lahko priključimo samo tipala tipa Pt100, pri čemer smo omejeni na dvožilni priklop brez kompenzacije upornosti, temperaturno območje je fiksno določeno od 0 °C do 100 °C in, ker gre za analogni pretvornik, svojega izhoda ne povprečuje, kar poveča njegovo odzivnost. Vendar pa je zaradi izdelave z analognimi gradniki nekoliko bolj občutljiv na zunanje motnje.

Slika 2.8: Pretvornika MO4202 ter LKM120 [6].

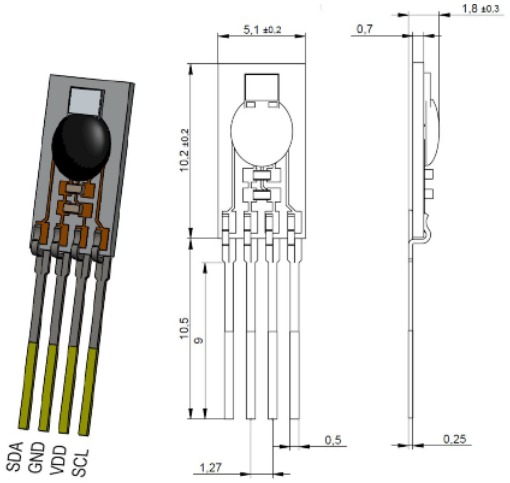
Iz slike 2.7 je razvidno, da imamo med napetostnim vhodom analogno-digitalnega pretvornika (v nadaljevanju: AD) RA0 in ozemljitvijo mikrokrmilnika vezan 200-ohmski upor. Na tem mestu je potrebno omeniti le to, da je upor potreben, saj je izhodni signal uporabljenega pretvornika električni tok, za naš AD pa potrebujemo napetost. Zato z uporabo Ohmovega zakona AD enota meri padec napetosti na uporu. Več o tem v podpoglavju 2.3.1.

2.1.5 HYT 271

HYT 271 proizvajalca IST je natančno kalibrirano temperaturno tipalo ter tipalo relativne vlažnosti zraka z možnostjo komunikacije prek protokola I^2C [7]. Njegove najpomembnejše tehnične specifikacije so:

- Merjenje vlage
  - merilno območje: od 0 do 100 % rH (ang. *relative humidity*),
  - merilna natančnost: ± 1.8 % rH (v območju od 0 do 80 % rH),
  - meritvena resolucija: 0,03 % rH,
  - način merjenja: kapacitivno polimerno tipalo (ang. *capacitive polymer humidity sensor*).
- Merjenje temperature
  - merilno območje: od –40 do 125 °C,
  - temperaturna natančnost: ± 0,2 °C (v območju od 0 do 60 °C),
  - merilna resolucija: 0,015 °C,
  - način merjenja: PTAT (ang. *proportional to absolute temperature*).

Tipalo HYT 271 za delovanje potrebuje napajanje (+5 V ter GND) ter signala SCL (ang. *serial clock*) in SDA (ang. *serial data*) za komunikacijo prek vodila I^2C. Ker sta SCL in SDA tipa odprti ponor (ang. *open drain*) jih moramo prek t. i. "pull up" uporov vezati na napajanje. Slika 2.9 prikazuje izgled tipala, dimenzije ter razporeditev nožic. Več o samem delovanju tipala in komunikaciji prek protokola I^2C je v podpoglavju 2.3.3.



Slika 2.9: Tipalo HYT 271 [7].

2.1.6 PD10 USB/RS485 pretvornik

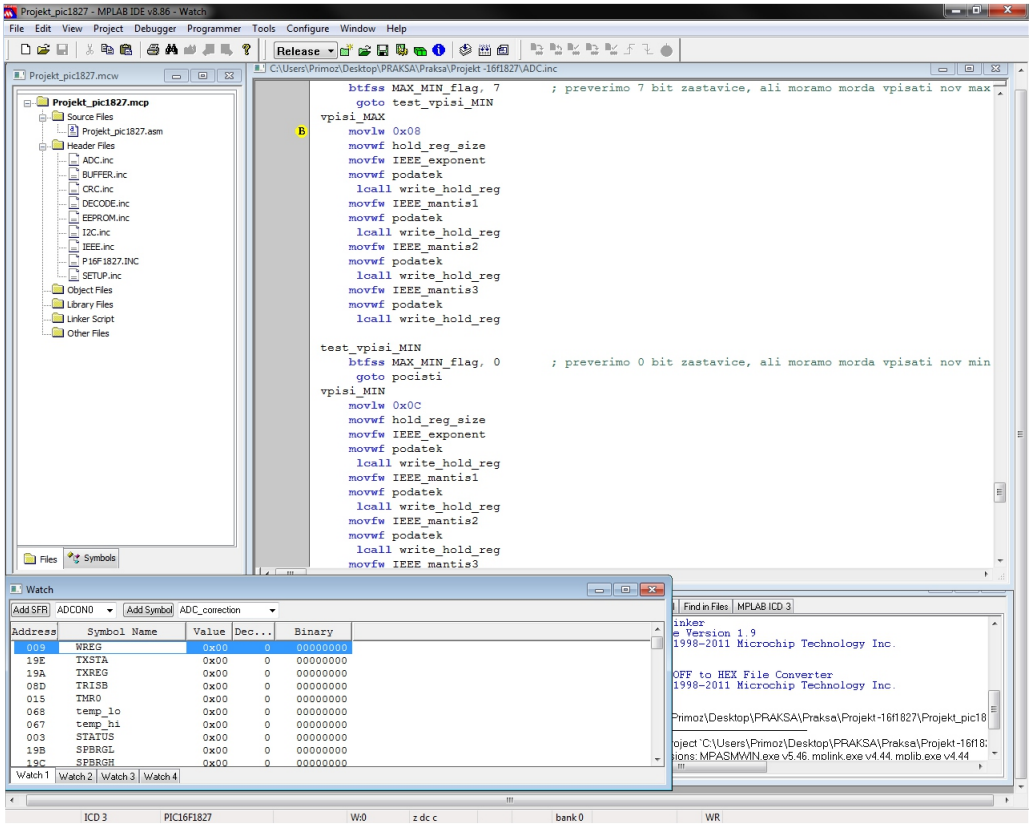
Ker smo pri razvoju vgrajenega sistema razvijali tudi komunikacijo našega sistema z zunanjimi napravami z uporabo standarda RS-485, smo potrebovali pretvornik, ki nam na našem računalniku omogoča serijsko komunikacijo. Odločili smo se za pretvornik PD10 poljskega proizvajalca Lumel [9].

2.2 Razvojno okolje MPLAB IDE

Pri izdelavi vgrajenega sistema za diplomsko nalogo smo uporabljali razvojno okolje MPLAB IDE (različica 8.86) [10]. Gre za razvojno okolje proizvajalca Microchip in je popolnoma združljivo s programatorjem/razhroščevalnikom MPLAB ICD3 (gl. podpoglavje 2.1.2). Prav tako ima polno podporo za delo s PIC mikrokrmilniki, kar se je izkazalo za ključni element pri razhroščevanu samega sistema. Omogoča dva načina delovanja, s katerima MPLAB IDE krmili delovanje ICD3-ja:

- programski način (ang. *release mode*); v tem načinu programator ICD3 v mikrokrmilnik zapisuje končno različico programa (ang. *release build*). Razhroščevanje ter spremljanje vrednosti registrov ni možno.

- razhroščevalni način (ang. *debug mode*); v tem načinu razhroščevalnik ICD3 v mikrokrmilnik zapisuje razhroščevalno različico programa (ang. *debug build*). Možno je prekinjanje izvajanja, vstavljanje prekinitvenih točk ter spremljanje vrednosti registrov in pomnilnika.



Slika 2.10: Razvojno okolje MPLAB IDE 8.86 v programskem načinu.

2.3 Funkcije in delovanje vgrajenega sistema

Vgrajen sistem ima dva načina delovanja, med katerima izbiramo z mostičkom (gl. sliki 2.1(8), 2.2(5)). Položaj mostička je pomemben, saj določa, ali se v glavnem programu izvaja rutina za komunikacijo z gospodarjem vodila Modbus ali pa rutina za konfiguracijo delovanja vgrajenega sistema. Na druge funkcije sistema mostiček ne vpliva. Ker je delovanje VS odvisno tudi od nekaterih parametrov, ki, na primer, določajo podatkovno hitrost, naslov naprave in podobno, te parametre hranimo v obstojnem EEPROM pomnilniku, saj bi se v podatkovnem flash pomnilniku ob morebitni izgubi napajanja izbrisali, kar bi močno okrnilo funkcionalnost sistema.

2.3.1 Ureditev rezultata AD pretvorbe

AD pretvornik, uporabljen v tej diplomski nalogi, je vgrajena komponenta PIC mikrokrmilnika z naslednjimi specifikacijami:

- podpira priklop zunanje reference, lahko pa uporabimo tudi vgrajen referenčni modul FVR (ang. *Fixed Voltage Reference*); maksimalna referenčna napetost je 5 V,
- pretvornik je 10-bitni (1024 različnih vrednost), rezultat pa zapisuje v dva 8-bitna registra `ADRESH in ADRESL`, tako da sta najpomembnejša bita na zadnjih dveh mestih v `ADRESH` (zgornjih 6 bitov = 0), spodnjih 8 bitov pa je v `ADRESL`.

Resolucijo pretvornika lahko preprosto izračunamo po enačbi 2.1. V našem primeru smo za referenčno napetost izbrali 4,096 V, ki je ena od napetosti FVR modula. Pri izbrani napetosti je resolucija pretvornika 0,004 V. Če bi uporabili referenco 5 V, bi imeli AD z resolucijo 0,00489 V.

$$resolucija = \frac{V_{ref}}{1024} \tag{2.1}$$

V podpoglavju 2.1.4 smo omenili vezavo 200 Ω upora med analognim vhodom AD enote ter GND (gl. sliko 2.7). Ker AD enota pretvarja napetost v digitalno vrednost, moramo analogni izhod pretvornika (LKM120 ali MO4201), ki se nahaja v območju od 4 do 20 mA, pretvoriti v napetost. To dosežemo z vezavo upora in merjenjem padca napetosti na uporu (uporabimo Ohmov zakon). Uporabili smo natančen metal-plastni upor, s toleranco $\pm 0,1\%$ ter izredno nizko temperaturno odvisnostjo (15 ppm/°C) [17].

Torej izhod RI pretvornika z uporabo upora pretvorimo iz 4 – 20 mA v 0,8 – 4 V. Napetostni razpon je torej 3,2 V. Če razpon delimo z resolucijo našega AD pretvornika, dobimo 800 različnih digitaliziranih vrednosti napetosti. V tem primeru smo izgubili 224 vrednosti (od 0 do 0,8 V ter od 4 do 4,096 V), ki jih bomo uporabili kasneje.

Z namenom, da predstavimo podrobnejše delovanje AD pretvornika, se bomo posvetili pretvorbi z uporabo RI pretvornika TKM120, saj nam ta ponuja več možnosti na področju izvajanja meritev (ena od možnih je popolnoma enaka kot pri uporabi MO4201). V podpoglavju 2.1.4 smo omenili, da je pretvornik LKM120 programabilen. To pomeni, da ima nastavljivo območje izvajanja meritev. Naš sistem podpira tri razpone temperaturnih meritev ter pri vsakem razponu tri različna območja (gl. tabelo 2.1).

Razpon	**Območje**	**Natančnost**	**Zamik**
50 °C	–20...+30 °C	0,0625 °C	32,5 °C
	–10 ... +40 °C		22,5 °C
	0 ... +50 °C		12,5 °C
100 °C	–30 ... +70 °C	0,1250 °C	55 °C
	–20 ... +80 °C		45 °C
	0 ... +100 °C		25 °C
200 °C	–100 ... +100 °C	0,2500 °C	150 °C
	–50 ... +150 °C		100 °C
	0 ... +200 °C		50 °C

Tabela 2.1: Možne nastavitve AD pretvorbe.

Pomembno je, da se ujemata nastavitvi RI in AD pretvornika, saj drugače med pretvarjanjem pride do napak. Ko je ta pogoj izpolnjen, pa se pretvorba lahko začne. Pretvorbo bomo opisali na primeru, ko sta oba pretvornika nastavljena na merilni razpon 100 °C na območju od 0 do 100 °C. Za RI pretvornik to pomeni, da bo pri upornosti tipala, specifični za 0 °C, njegov izhod 4 mA, padec napetosti na uporu pa 0,8 V. Pri upornosti, specifični za 100 °C, bo izhod 20 mA, padec napetosti pa 4 V. Ker lahko na razponu 3,2 V zapišemo 800 različnih vrednosti, lahko tudi temperaturo na razponu 100 °C razdelimo na 800 vrednosti. Posledično lahko vsako stopinjo Celzija razdelimo na osem vrednosti, kar nam da natančnost na osminko stopinje (gl. tabelo 2.1, st. 3).

Pri preračunavanju temperature iz vrednosti AD pretvorbe je potrebno upoštevati tudi zamik (ang. *offset*) (gl. tabelo 2.1, stolpec 4), saj je napetost na vhodu AD pretvornika pri 0 °C enaka 0,8 V, kar se prek pretvorbe prezrcali v vrednost 200 v registrih AD pretvornika. Če to vrednost delimo z osem, dobimo rezultat 25, kar pomeni da pri pretvorbi na razponu 100 °C vedno kažemo 25 °C preveč. Zamike za posamezna območja izračunamo po enačbi 2.2:

$$zamik = \frac{200}{800/razpon} - |spodnja\ meja\ območja| \tag{2.2}$$

Torej pravo vrednost temperature izračunamo z uporabo enačbe 2.3:

$$temperatura = \frac{vrednost\ AD\ pretvorbe}{800/razpon} - zamik \tag{2.3}$$

Na samem mikrokrmilniku pa je postopek za pridobitev temperature sledeč:

1. Časovnik mikrokrmilnika (`TIMER0`) vsakih 0,001 s sproži prekinitev. Prekinitvena rutina preveri zahtevo po prekinitvi in kliče ustrezni prekinitveno-servisni program (ang. *ISR, Interrupt Service Routine*). V primeru časovnika je to rutina `ADC_routine`.

2. Rutina `ADC_routine` požene AD pretvorbo. Nato preveri, če smo dobili vrednost, ki je pod 0,8 V oziroma nad 4 V (preostalih 224 vrednosti). Če se je to zgodilo, javi napako, tako da postavi ustrezen bit za napako v statusni register pretvornika `ADC_napaka`. Če napake ni, rezultat pretvorbe (`ADRESH in ADRESL`) prišteje v dva začasna registra (`ADC_rtemp_hi in ADC_rtemp_lo`) in zmanjša števec časovnikovih prekinitev (na začetku ima vrednost 64). Nato preveri, ali je števec morda prišel do 0. Če ni, zaključi prekinitev in se vrne v glavni program. V nasprotnem primeru postavi števec prekinitev nazaj na začetno vrednost in začne z izvajanjem točke 3.

3. V registrih `ADC_rtemp_hi in ADC_rtemp_lo` imamo seštete vrednosti zadnjih 64-ih meritev AD pretvornika. Izračunamo njihovo povprečje, tako da registra šestkrat zapored pomaknemo (ang. *shift*) v desno (deljenje s 64). Zaporedje pomikov je: `ADC_rtemp_hi`→`ADC_rtemp_lo`. S tem zmanjšamo vpliv morebitnih napak.

4. Nato se izvede preverjanje, ali je izmerjena vrednost nov maksimum oziroma minimum. Če je nova vrednost minimum, se postavi zastavica `new_MIN`, ob maksimumu pa zastavica `new_MAX`. Zastavico preverja rutina ob zaključevanju svojega delovanja.

5. Naslednji korak je zaporedno deljenje rezultata glede na razpon. Pri razponu 100 °C je potrebno opraviti deljenje z osem, pri 200 °C s štiri in pri 50 °C s 16. Deljenje znova opravimo s pomočjo pomikov v desno: `ADC_rtemp_hi`→`ADC_rtemp_lo`→`ADC_rtemp_dec`.

6. Nadaljujemo z izračunom uteženega povprečja trenutne meritve in zadnjih treh meritev. Povprečje izračunamo z uporabo enačbe 2.4. To storimo, da je naraščanje oziroma padanje bolj postopno.

$$povprecje = \frac{meritev_0 + 3 * meritev_{-1} + 2 * meritev_{-2} + 2 * meritev_{-3}}{8} \tag{2.4}$$

7. Sedaj lahko glede na razpon in območje meritev od izračunanega povprečja (v `ADC_res_lo` se nahaja vrednost pred decimalno vejico, v `ADC_decimals` pa vrednost za decimalno vejico) odštejemo odmik, kot je prikazano v tabeli 2.1. Pri tem preverjamo, ali smo pri odštevanju prišli v negativne vrednosti. Če se je to zgodilo (postavila se je zastavica prenos (ang. *carry*)), je potrebno s funkcijo ekskluzivni ali (ang. *xor*) invertirati vrednosti obeh registrov ter ustrezno nastaviti vrednost v registru `IEEE_predznak`, ki ga upoštevamo pri pretvarjanju v format IEEE 754.

8. Vrednosti registrov `ADC_res_lo` in `ADC_decimals` prenesemo v vhodne registre funkcije `BIN2IEEE` ter pokličemo funkcijo za pretvorbo. Po vrnitvi iz funkcije rezultat shranimo v rezerviran del pomnilnika, do katerega imajo zunanje naprave dostop prek protokola Modbus. Več o tem v podpoglavju 2.3.4.

Uspeli smo pretvoriti napetost na vhodu AD pretvornika v decimalno vrednost, pri čemer vrednost pred decimalno vejico in za vejico hranimo v ločenih registrih, posebej pa imamo shranjen tudi predznak rezultata.

2.3.2 Pretvorba vrednosti v format IEEE754

Za pretvorbo poskrbi rutina `BIN2IEEE`, ki zahteva, da je vrednost pred decimalno vejico v registru `IEEE_calc_hi`, vrednost za decimalno vejico v registru `IEEE_calc_lo`, predznak pa mora biti zadnji bit registra `IEEE_predznak`. Torej moramo pred vsakim klicem te rutine vhodne podatke pripravti v ustrezne registre.

Rutina pa se izvaja po naslednjem postopku:

1. V zbirniški različici CASE zanke določimo vrednost eksponenta. To storimo tako, da preverimo, na katerem mestu je prvi postavljen bit v registru `IEEE_calc_hi` (premikamo se od najbolj proti najmanj pomembnemu bitu registra). Na mestu, na katerem se nahaja prva enica, napišemo ničlo, indeksu mesta pa prištejemo 127 in vrednost shranimo v register `IEEE_eksponent`. Če so v registru `IEEE_calc_hi` le ničle, poskusimo določiti eksponent s preverjanjem registra `IEEE_calc_lo`, pri čemer tu indeksu mesta prve enice prištejemo 119, enico pa prav tako zbrišemo (vodilno enico zbrišemo, saj jo predpostavlja že IEEE 754).

2. Po določitvi eksponenta pa je potrebno urediti še mantiso zapisa. Najprej prenesemo vrednost registra `IEEE_calc_lo` v register `IEEE_mantis_1`. V primeru eksponenta, manjšega od 127 (prva enica je bila šele v registru `IEEE_calc_lo`), to storimo tako, da od vrednosti eksponenta odštejemo 119 in rezultat shranimo v register `IEEE_stevec`. Nato pa izvedemo za vrednost `IEEE_stevec` levih pomikov mantise. V primeru eksponenta, večjega oziroma enakega 127, pa je v `IEEE_stevec` vrednost eksponenta

zmanjšana za 127. Pomik mantise se izvaja v desno, pri čemer je potrebno upoštevati tudi bite v registru `IEEE_calc_hi`, ki jih pomikamo najprej.

3. Na koncu pretvorjeni vrednosti dodamo že prej pripravljeni predznak, s čimer se pretvorba zaključi.

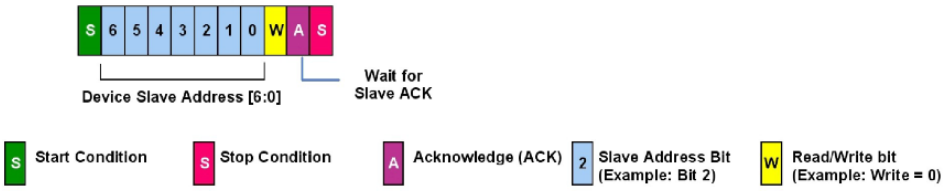
Rezultat se po koncu pretvorbe nahaja v štirih 8-bitnih registrih. Prvi je register `IEEE_eksponent`, v katerem MSB bit predstavlja predznak, naslednjih sedem bitov pa so zgornji biti eksponenta. Naslednji je `IEEE_mantis1`, katerega MSB je LSB eksponenta, vsi preostali biti pa so zgornji del mantise. Registra `IEEE_mantis2` in `IEEE_mantis3` prav tako predstavljata mantiso števila v formatu IEEE 754.

2.3.3 Temperatura in vlaga čipa HYT 271

Omenili smo že, da naprave s tipalom HYT 271 lahko komunicirajo prek vodila I^2C. Opisane so lastnosti vodila, pomembne za izdelavo te diplomske naloge, za več informacij o vodilu pa svetujemo pregled dokumentacije podjetja NXP, v kateri je delovanje vodila obširno opisano [11]. V našem primeru je komunikacija temeljila na principu gospodar/suženj, pri čemer je bil mikrokrmilnik gospodar, tipalo pa suženj. Vodilo I^2C omogoča priklop 128 naprav na vodilo, v danem trenutku pa se odzove le naslovljena naprava. Če želimo imeti 127 naprav na vodilu, moramo imeti 7-bitne naslove. Prednastavljeni naslov našega tipala je 0x28, ki ga je možno programsko spreminjati (ker imamo na vodilu samo enega sužnja, naslova nismo spremenili) [7]. Komunikacija se vedno začne s start sekvenco, konča s stop sekvenco, vmes pa mora sprejemnik vedno potrditi (ang. *acknowledge*) oziroma zavrniti (ang. *not acknowledge*) prejete podatke. Komunikacijo vedno začne gospodar vodila s start sekvenco, nato pa komunikacija poteka z bajtnimi podatki. Prvih sedem bitov prvega bajta je naslov naslovnjene naprave, sledi pa bit, ki določa branje ali pisanje v napravo [8].

Tipalo HYT 271 nam ponuja dve funkciji, s katerima lahko spremljamo njegovo delovanje. Imenujeta se MR (ang. *Measure Request*) in DF (ang. *Data Fetch*).

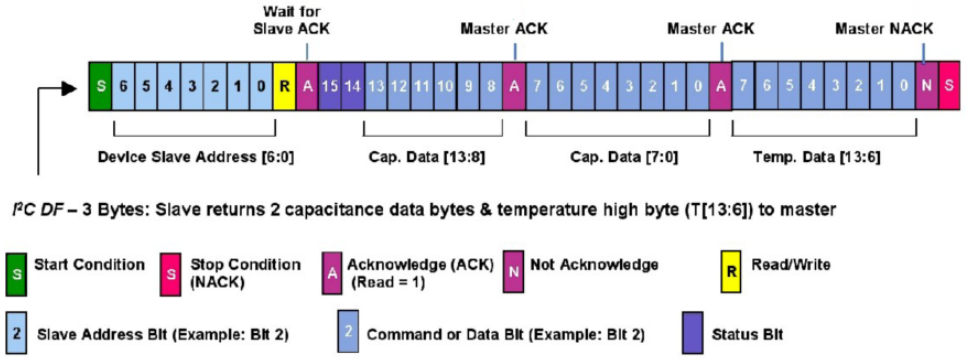
Z ukazom MR sprožimo merilni cikel tipala. Začne se s start sekvenco, sledi 7-bitni naslov ter zadnji bit, ki je v tem primeru 0 in pomeni pisanje v napravo. Če je tipalo ukaz sprejelo, odgovori z ACK, gospodar pa zaključi komunikacijo s stop sekvenco. Komunikacija je nazorno prikazana na sliki 2.11.



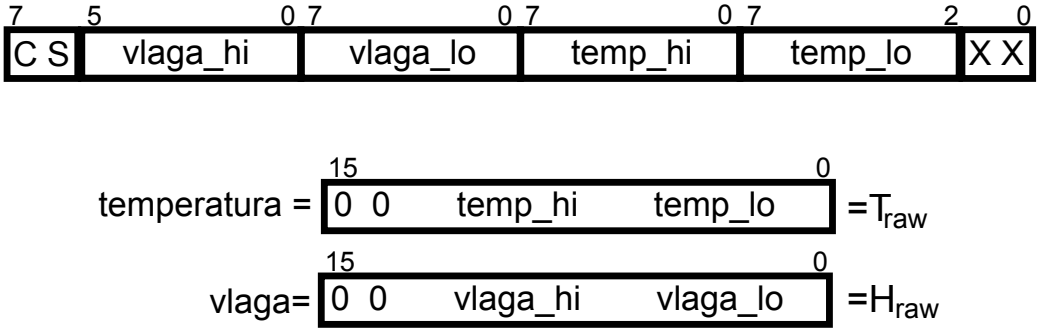
Slika 2.11: Ukaz MR. [8]

Ukaz DF pa uporabimo, ko želimo prebrati vrednosti, ki jih je tipalo izmerilo. Postopek je enak kot pri pošiljanju ukaza MR, le da je v tem primeru zadnji bit prvega bajta enak 1, kar pomeni branje naprave. V primeru pravilnega naslavljanja tipalo odgovori z ACK ter nam začne pošiljati podatke. Gospodar vodila (mikrokrmilnik) mora vsak prejeti bajt potrditi ali zavrniti. Odgovor tipala na ukaz DF je dolg štiri bajte. Najvišja (MSB) dva bita prvega bajta odgovora sta statusna bita, pri čemer bit 7 pove, ali je naprava v načinu za pisanje, bit 6 pa nam pove, ali je od zadnjega branja naprave na voljo nova vrednost. Če je katerikoli od statusnih bitov postavljen, branje ni smiselno, tako da se lahko že po prvem bajtu odločimo, ali bomo podatke potrdili, saj so ažurni, ali zavrnili, ker so nespremenjeni. Podoben primer prikazuje slika 2.12. Za pošiljanje ukaza DF je zadolžena rutina `I2C_preberi_meritev`. Ta pošlje 0x51 kot prvi bajt, počaka potrditev sužnja in se po potrditvi prestavi v način za sprejemanje. Ko prejme prvi bajt podatkov, preveri statusna bita. Če je

eden od njiju 1, pošlje NACK in zaključi komunikacijo. Če pa sta statusna bita prazna, pa v zanki potrjuje prejete podatke in jih shranjuje na ustrezna mesta. Po prejetem četrtem bajtu pošlje NACK in zaključi komunikacijo. Tako vlaga kot temperatura sta vrednosti, zapisani s 14 biti. Slika 2.13 prikazuje štiri bajte, ki smo jih prejeli.



Slika 2.12: Ukaz DF. [8]



Slika 2.13: Podatki, prejeti s strani tipala HYT 271.

V nadaljevanju pa je potrebno pretvoriti prejeto vlago in temperaturo v format, ki ga podpira naša funkcija `BIN2IEEE`. To pomeni, da je potrebno v register `IEEE_calc_hi` shraniti vrednost temperature oziroma vlage pred decimalno

vejico, v register `IEEE_calc_lo` vrednost za decimalno vejico, v `IEEE_predznak` pa predznak števila (slednje velja samo za temperaturo, kajti vlaga ne more biti negativna). V dokumentaciji tipala [8] sta za izračun temperature in vlage podani enačbi 2.5 in 2.6.

$$temperatura[℃] = \frac{165 * T_{raw}}{2^{14}} - 40 \tag{2.5}$$

$$vlaga[\%] = \frac{100 * H_{raw}}{2^{14}} \tag{2.6}$$

Če enačbi uporabimo na robnih primerih temperature in vlage, dobimo naslednje rezultate:

- $T_{raw} = 0 \times 0000 \rightarrow T = 0$ ℃

  $T_{raw} = 0 \times 3FFF \rightarrow T = 124,98993$ ℃
- $H_{raw} = 0 \times 0000 \rightarrow H = 0$ %

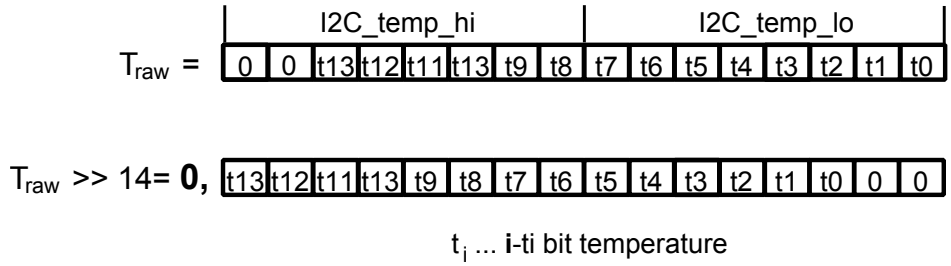
  $H_{raw} = 0 \times 3FFF \rightarrow H = 99,993896$ %

Iz zgoraj prikazanega primera je razvidno, da pri maksimalni vrednost pride do manjše napake.

Torej mora tudi naša rutina za pretvorbo v ustrezne vrednosti uporabiti enačbi 2.5 in 2.6, vendar z manjšimi popravki. Najprej je potrebno omeniti, da že ob sprejemu podatkov v primeru, da so vrednosti ažurne, poskrbimo, da so prejete vrednosti v ustreznih registrih. Zgornjih šest bitov vrednosti vlage se nahaja v registru `I2C_hum_hi`, spodnjih osem pa v `I2C_hum_lo`. Vrednost temperature se nahaja v enaki razporeditvi v registrih `I2C_temp_hi` in `I2C_temp_lo`. Postopek reševanja enačb 2.5 in 2.6 je zelo podoben, zato bomo opisali samo primer reševanja enačbe za temperaturo, ki je malo bolj zahteven:

1. Najprej izvedemo deljenje z 2^{14}. Na sliki 2.14 je prikazano, da nam 14 pomikov v desno da enak rezultat, kot če bi izvedli dva pomika v levo ter

decimalno vejico prestavili pred vrednost T_{raw}. Torej moramo za izvedbo deljenja samo izvesti dva zaporedna pomika registrov v levo in vedeti, da decimalna vejica stoji spredaj.



Slika 2.14: Deljenje temperature z 2^{14} s pomočjo shift operatorja.

2. Naslednji korak v enačbi je množenje s 165. Ker mikrokrmilnik podpira le množenje s potencami števila dva (pomiki v levo), smo množenje s 165 realizirali kot vsoto produktov z uporabo enačbe 2.7.

$$T_{raw} \times 165 = T_{raw} \times 128 + T_{raw} \times 32 + T_{raw} \times 4 + T_{raw} \tag{2.7}$$

   Vsa množenja v enačbi 2.7 je možno realizirati s pomiki v levo, potrebno pa je uporabiti še dodatni register za vrednost pred decimalno vejico `I2C_hi_shift`. Na koncu predelanega postopka za množenje se rezultat nahaja v treh registrih. Vrednost pred decimalno vejico je v registru `I2C_res_hi_temp`, vrednosti za decimalno vejico pa v registrih `I2C_res_med_temp` in `I2C_res_lo_temp`.

3. Na koncu reševanja enačbe za izračun temperature je od vrednosti pred decimalno vejico potrebno le še odšteti 40. Če je končna vrednost temperature negativna, moramo poskrbeti za inverze vseh registrov temperature ter postaviti LSB registra `IEEE_predznak`.

4. Nato prestavimo vrednosti registrov temperature v vhodne registre funkcije `BIN2IEEE` 2.3.2 ter pokličemo funkcijo za pretvarjanje. Po vrnitvi

iz funkcije rezultat shranimo v rezerviran del pomnilnika, do katerega imajo dostop zunanje naprave prek protokola Modbus. Več o tem v podpoglavju 2.3.4.

Med preračunavanjem temperature in vlage so razlike minimalne. Množenje s 100 lahko realiziramo kot vsoto množenj s 64, 32 in 4, medtem ko nam odštevati odmika in preverjati negativnosti sploh ni potrebno.

2.3.4 Komunikacija z zunanjmi napravami

Komunikacija med našim vgrajenim sistemom in zunanjimi napravami poteka z uporabo standarda RS-485 in protokola Modbus za serijsko komunikacijo. Možnost komunikacije prek RS-485 nam omogoča gonilnik SM75176 (gl. podpoglavje 2.1.3), za implementacijo protokola Modbus pa smo morali poskrbeti sami.

Modbus gospodar/suženj

Implementirali smo različico Modbus gospodar/suženj (ang. *master/slave*), ki za svoje delovanje postavlja naslednje zahteve [13, 14]:

1. Vse naprave, povezane v sistem, morajo imeti enake parametre prenosa.
2. Vse naprave, povezane v sistem, morajo imeti unikaten naslov.
3. Vse naprave, povezane v sistem, morajo uporabljati enak način delovanja.
4. Komunikacijo začenja samo gospodar vodila, sužnji samo odgovarjajo.
5. Suženj mora imeti implementirani funkciji za branje iz in pisanje v napravo.

Za zadostitev prvi zahtevi ima naša naprava na voljo pet različnih podatkovnih hitrosti (ang. *baud rate*) (gl. tabela 2.2, vrstica 7), med njimi tudi najpogosteje

uporabljeno pri komunikaciji z uporabo standarda RS-485, in sicer 9600. Prav tako lahko izbiramo med enim ali dvema stop bitoma, medtem ko je število podatkovnih bitov vedno osem (obstaja samo ena nestandardizirana različica Modbus komunikacije, ki uporablja sedem podatkovnih bitov). Ker vgrajen sistem podpira vse dovoljene naslove naprav, s tem izpolni tudi drugi pogoj.

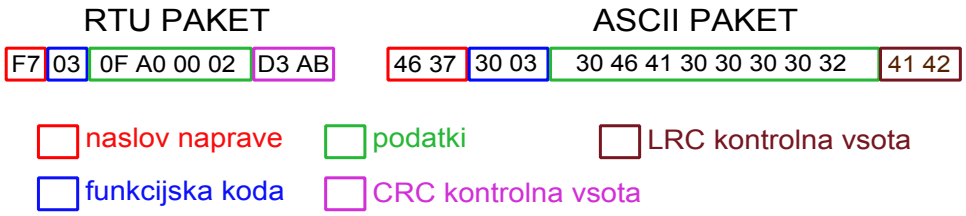
Način delovanja	**RTU**		**ASCII**	
Različica načina	RTU 8n1	RTU 8n2	ASCII 8n1	ASCII 8n2
Št. podatkovnih bitov	8			
Št. stop bitov	1	2	1	2
Naslov naprave	1 – 247 (0 – broadcast, 248 – 255 rezervirani)			
Kontrolna vsota	CRC		LRC	
Podatkovna hitrost	9600 10417 19200 57600 115200			

Tabela 2.2: Možne nastavitve komunikacije Modbus.

Iz tabele 2.2 je razvidno, da obstajata dva načina delovanja RTU (ang. *Remote Terminal Unit*) in ASCII, ki se delita glede na število stop bitov. Čeprav se najpogosteje uporablja način RTU, ki ga mora vsebovati vsaka Modbus naprava, smo se odločili tudi za implementacijo načina ASCII, da napravi povečamo nabor komunikacijskih zmožnosti. Da zadostimo zahtevi št. 4, moramo pošiljati podatke samo takrat, ko je naslovljena naša naprava, funkciji za branje in pisanje pa sta podrobneje razloženi v nadaljevanju.

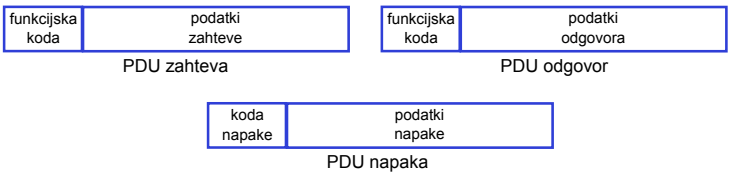
Medtem ko Modbus ASCII za kontrolno vsoto (ang. *checksum*) uporablja LRC (ang. *Longitudinal Redundancy Check*), Modbus RTU uporablja CRC (ang. *Cyclic Redundancy Check*). VS ima implementirano zančno različico

izračuna kontrolne vsote, saj bi prednastavljena tabela vrednosti za izračun CRC kontrolne vsote zasedla prevelik del podatkovnega pomnilnika [12]. Vendar pa kontrolna vsota ni edina razlika med načinoma delovanja. Načina se najbolj razlikujeta po kodiranju posameznih bajtov sporočila. Modbus RTU pošilja surove podatkovne bajte, prenos sporočila mora potekati neprekinjeno, sporočila pa se med seboj ločuje s t. i. tihimi periodami (ang. *idle periods*). Pri načinu Modbus ASCII pa se vsak podatkovni bajt zakodira v dva ASCII znaka (od tod tudi ime načina ASCII). S tem se velikost posameznega sporočila poveča za faktor dva. Za ločevanje sporočil se uporablja ločilo **:** na začetku prenosa in **CR LF** (ang. *Carriage Return, Line Feed*) na koncu prenosa. Slika 2.15 prikazuje isto sporočilo v različnih načinih delovanja [13].



Slika 2.15: Modbus paket v RTU in ASCII načinu delovanja.

Paket – ne glede na način delovanja – sestavljajo naslov naprave, podatkovna enota (PDU ang. *Protocol Data Unit*), sestavljena iz funkcijske kode in podatkov, ter kontrolna vsota. Poznamo tri različne s standardom definirane podatkovne enote (gl. sliko 2.16).



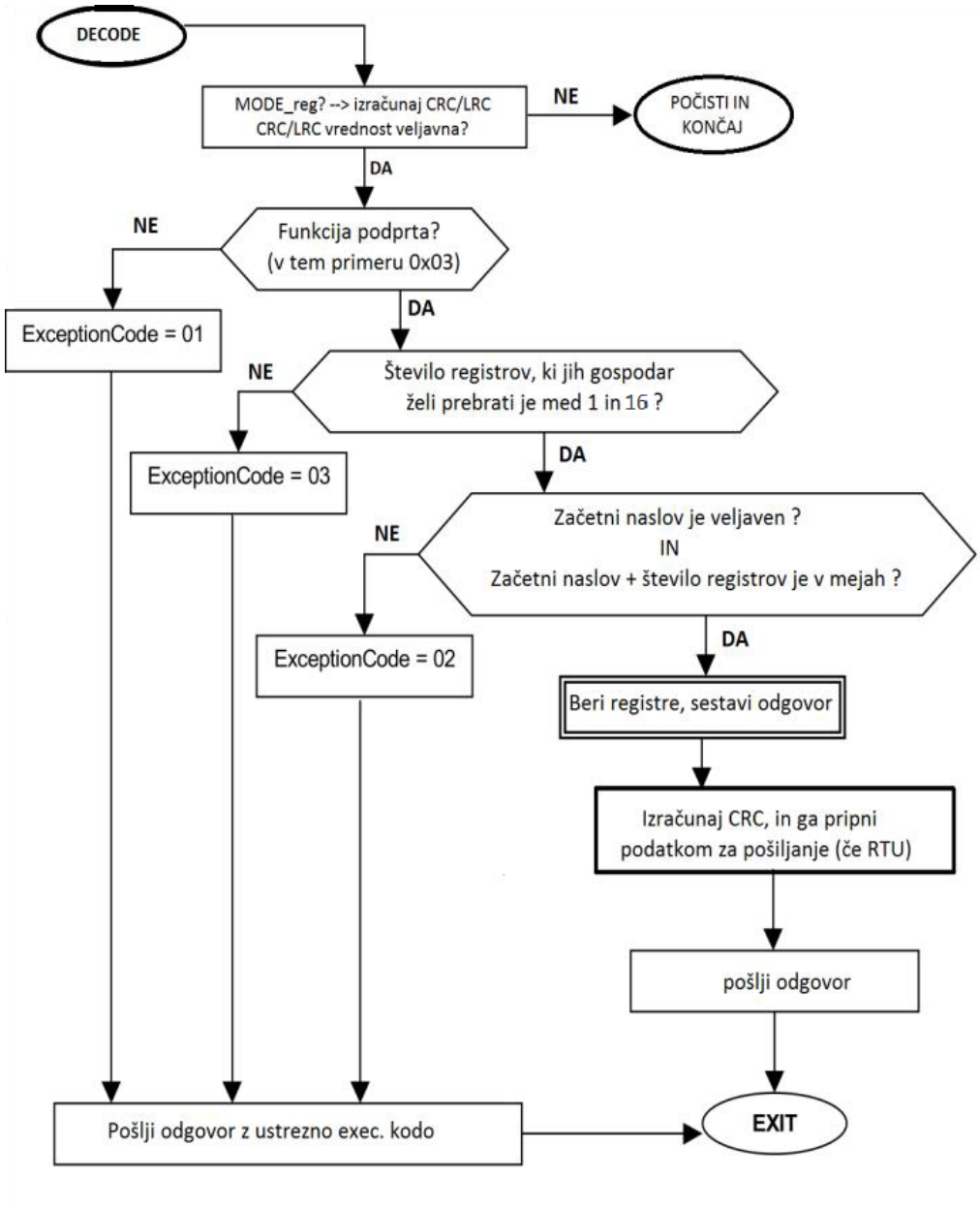
Slika 2.16: Podatkovne enote pri protkolu Modbus.

Implementacija protokola Modbus v vgrajenem sistemu

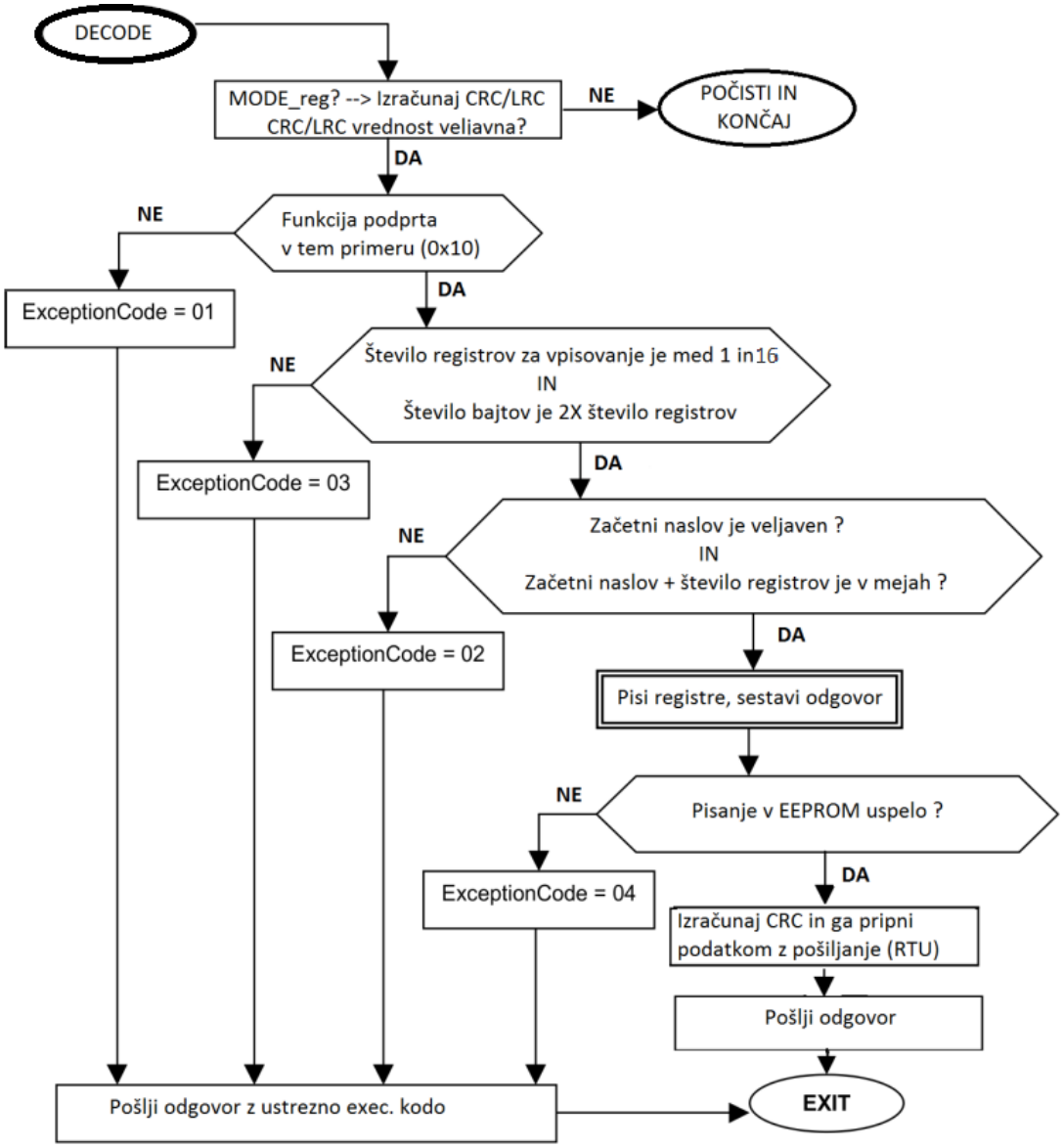
Vsak bajt prejet preko UART enote sproži prekinitev in posledično prek preverbe klic prekinitveno-servisnega programa (v nadaljevanju: PSP) UART enote. Vsak prejeti bajt PSP zapiše v programsko realiziran medpomnilnik (ang. *buffer*) z imenom `buffer`. Pred zapisom bajta v `buffer` pa:

- V RTU načinu preveri, ali je `buffer` še prazen in je to prvi bajt paketa, ki ga prejemamo. Če je, preveri, ali se prejeti bajt ujema z naslovom naše naprave. Če ni, ga zavrže, saj paket ni namenjen nam, v nasprotnem primeru pa ga zapiše v `buffer` in nadaljuje s prejemanjem naslednjih bajtov paketa. Med samim prejemanjem PSP razbere, koliko bajtov mora prejeti do konca paketa. Ko se to zgodi, postavi zastavico za konec.
- V ASCII načinu preveri, ali je `buffer` še prazen in je to prvi bajt paketa, ki ga prejemamo. Če je, preveri, ali se prejeti bajt ujema s prvim bajtom v ASCII pretvorjenega naslova naše naprave. Če ni, ga zavrže, saj paket ni namenjen nam, v nasprotnem primeru pa ga zapiše v `buffer` in nadaljuje s prejemanjem naslednjega bajta paketa, ki mora, če je naslovljena naša naprava, ustrezati drugemu bajtu pretvorjenega naslova. Odločitev je enaka kot pri prejemanju prvega bajta naslova. Po prejeti **CR LF** sekvenci PSP postavi zastavico za konec.

Glavni program pa neprestano izvaja tako imenovano izpraševanje (ang. *polling*) registra, v katerem se nahaja zastavica, ki določa, kdaj smo prejeli celotni paket. Ko se zastavica postavi, glavni program kliče rutino `DECODE`, katere delovanje se razlikuje glede na prejeto funkcijsko kodo. Diagrama poteka rutine `DECODE` sta prikazana na slikah 2.17 in 2.18.



Slika 2.17: Branje registrov iz naprave.



Slika 2.18: Pisanje v registre naprave.

V protokolu Modbus so registri, iz katerih lahko beremo podatke, in registri, v katere lahko vpisujemo podatke, 16-bitni. Če želimo zapisati 32-bitno vrednost, za vpis potrebujemo dva taka registra. Priporočljivo je, da so 32-bitne vrednosti zapisane tako, da je prva vrednost v registru s sodim naslovom. Pravilo je, da so pomembnejši biti v registru z nižjim naslovom (pravilo debelega konca). Ker moramo poskrbeti za naslovni prostor v mikrokrmilniku, do katerega lahko z ukazom za branje registrov dostopajo zunanje naprave, smo morali rezervirati nekaj podatkovnega flash pomnilnika in izvesti preslikavo Modbus naslovov v naslove pomnilniških celic mikrokrmilnika. Prav tako smo s podobno preslikavo poskrbeli, da z ukazom za pisanje v registre vrednosti zapisujemo v EEPROM mikrokrmilnika.

Naslovni prostor mikrokrmilnika smo poimenovali `hold_reg`. Vsebuje 32 bajtov, torej 16 Modbus registrov. Naslov prvega registra je 4001_{10}, naslov zadnjega veljavnega registra za branje pa 4016_{10}. Vsebino pomnilniškega polja `hold_reg` pa prikazuje tabela 2.3.

Pomnilniško polje `hold_reg` za en Modbus register združuje dve pomnilniški celici. Po polju se pomikamo z uporabo posrednega (indirektnega) naslavljanja, pri čemer se za bazni naslov uporabi naslov prve celice, odmik pa se računa z uporabo naslova prvega registra v zahtevi Modbus gospodarja. Ušpoštevati moramo dejstvo, da je odmik za en Modbus register v zahtevi pravzaprav odmik za dve pomnilniški celici.

Ker pa VS poleg branja registrov podpira tudi pisanje v registre, je bilo pomembno poskrbeti tudi za pisanje v podatkovni prostor mikrokrmilnika. Vendar pa moramo tu upoštevati, da flash podatkovni pomnilnik ni obstojen, posledničnо pa se ob vsaki izgubi napajanja vsebina izgubi. To lahko predstavlja oviro, saj ne želimo, da bi v primeru vpisovanja parametrov v napravo prek Modbus zahtev morali postopek nastavljanja ponoviti po vsaki izgubi napaja-

Naslov registra	**Vsebina registra**
4001_{10} $(0 \times FA1)$	status naprave
4002_{10} $(0 \times FA2)$ 4003_{10} $(0 \times FA3)$	IEEE754 zapis trenutne temperature, izmerjene s tipalom Pt100
4004_{10} $(0 \times FA4)$ 4005_{10} $(0 \times FA5)$	IEEE754 zapis maksimalne temperature, izmerjene s tipalom Pt100
4006_{10} $(0 \times FA6)$ 4007_{10} $(0 \times FA7)$	IEEE754 zapis minamalne temperature, izmerjene s tipalom Pt100
4008_{10} $(0 \times FA8)$ 4009_{10} $(0 \times FA9)$	IEEE754 zapis trenutne temperature, izmerjene s tipalom HYT 271
4010_{10} $(0 \times FAA)$ 4011_{10} $(0 \times FAB)$	IEEE754 zapis trenutne vlage, izmerjene s tipalom HYT 271
4012_{10} $(0 \times FAB)$ . . . 4016_{10} $(0 \times FB0)$	Neuporabljeno (vrednosti so neznane)

Tabela 2.3: Vsebina registrov, do katerih lahko dostopamo z ukazom za branje registrov.

nja. Zato smo implementirali preslikavo EEPROM mikrokrmilnika v registre za vpisovanje. Naslovni prostor za vpisovanje je prav tako velik 32 bajtov (16 EEPROM celic), zapisovanje vrednosti pa je prav tako realizirano z uporabo posrednega naslavljanja z baznim naslovom prvega registra, natančneje z naslovom prve EEPROM celice v pomnilniškem prostoru `EE_user_regs`. Naslov prvega registra, v katerega je možno vpisovati vrednosti je 7001_{10}, zadnjega pa 7016_{10}. Ker vse nastavitve naprave spreminjamo prek aplikacije za nastavljanje (poglavje 3), vpisovanje vrednosti v te pomnilniške lokacije ne spreminja ničesar v delovanju samega VS; implementacija je bila izvedena zaradi lažje razširitve sistema z novimi funkcijami.

Smiselno je omeniti tudi to, da zaradi elektromagnetnih lastnosti pomnilnika vsako pisanje v EEPROM pomnilnik, potrebuje časovno zakasnitev po pisanju. V primeru našega mikrokrmilnika mora biti med vsakim pisanjem v EEPROM najmanj 8 ms presledka. Ker je možno z enim ukazom pisati v vseh 32 celic pomnilnika EEPROM, ki sestavljajo naslovni prostor dostopen Modbus zahtevam za pisanje, to skupaj pomeni 250 ms, ko je mikrokrmilnik neodziven. Zato je smiselno poskrbeti, da v primeru pisanja v napravo povečamo čas, ki ga ima naš VS na razpolago, da sestavi odgovor na zahtevo (privzeta nastavitev naprav za čas, v katerem se mora naprava odzvati (ang. *timeout*), je 1 s, kar se je med testiranjem izkazalo za zadnostno vrednost).

2.3.5 Nastavitveni način

Že v uvodu tega podpoglavja smo omenili dva načina delovanja. Prvi t. i. `runtime` način je opisan v podpoglavju 2.3.4. Način delovanja, ki je opisan v tem podpoglavju, pa je t. i. `setup` način delovanja in se izvaja, če mostiček ni prisoten. V tem načinu je možno spreminjati delovanje posameznih komponent (funkcij oziroma rutin) vgrajenega sistema. V `setup` načinu delovanja VS podatkovno hitrost UART enote nastavi na 9600, prenos podatkov pa poteka z uporabo pravila **8n2** (osem podatkovnih bitov, brez paritete, en stop bit). Ko so parametri UART enote nastavljeni, lahko naprava začne s prejemanjem nastavitvenih ukazov, ki jih pošilja aplikacija, nameščena na računalniku po vodilu z uporabo standarda RS-485. Postopek prejemanja nastavitvenih ukazov je naslednji:

1. Vsak prejeti bajt enote UART sproži prekinitev. PSP ga zapiše v `buffer`. Ko prejme bajt $0 \times 0A$ (stisk tipke enter), postavi zastavico (`end_flag`), ki označuje prejetje novega konfiguracijskega ukaza.

2. Glavni program neprestano preverja, ali je prišlo do postavitve te zastavice; ko se to zgodi, kliče rutino `razberi_ukaz`, ki zahtevo obdela.

3. V rutini `razberi_ukaz` se zahteva obdela. Glede na vrsto ukaza se pripravi odgovor. Če je bilo zahtevano branje parametrov iz naprave, se ti preberejo iz EEPROM naprave ter prek enote UART v surovi obliki pošiljajo aplikaciji, ki jih posreduje uporabniku. V primeru vpisovanja parametrov rutina bere podatke iz medpomnilnika `buffer` ter jih zapisuje v EEPROM na ustrezna mesta.

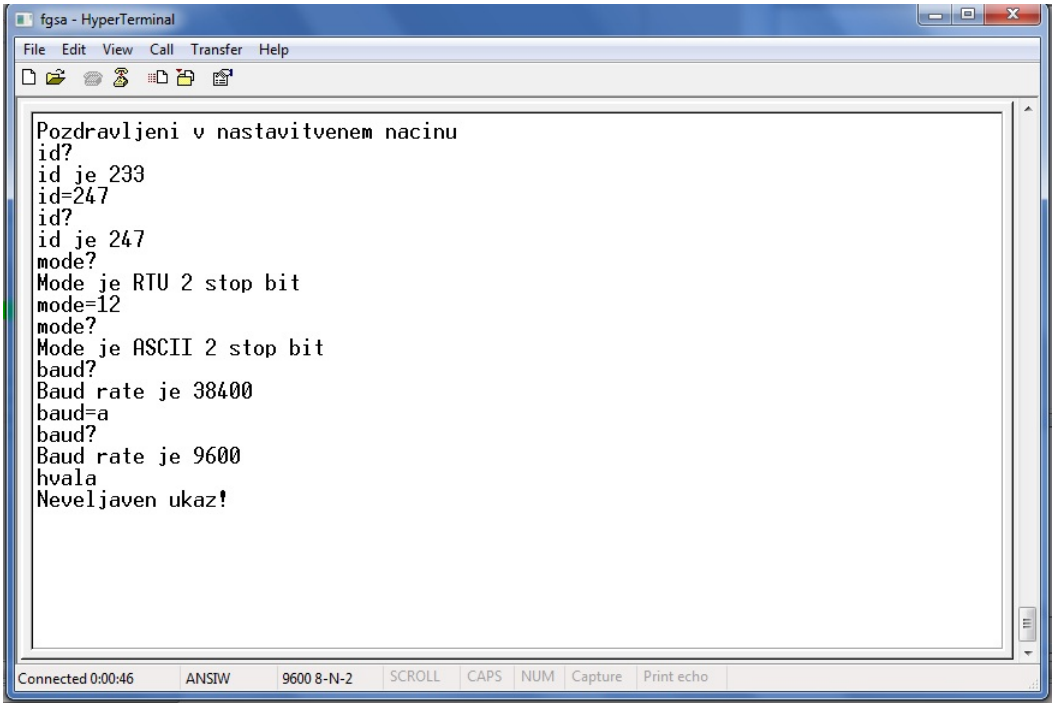
4. Ko konča z obdelavo zahteve, rutina počisti svoje delovne registre, prav tako pa tudi pobriše vsebino medpomnilnika `buffer` in se vrne v glavni program.

Več o samih parametrih, ki jih je možno nastavljati v `setup` načinu, je napisano v poglavju 3.

Poglavje 3

Aplikacija "Konfigurator"

Ker ima vgrajen sistem funkcije, ki jih lahko s spreminjanjem določenih parametrov prilagajamo trenutnim potrebam, je smiselno, da te parametre lahko nastavljamo kadarkoli in brez posegov v samo programsko kodo vgrajenega sistema. To dosežemo z aplikacijo s pomočjo katere v VS, ko je ta v nastavitvenem načinu (gl. podpoglavje 2.3.5) pošiljamo ukaze. Prvotna aplikacija za nastavljanje (konfiguriranje) VS je bil preprost Windowsov terminal, ki se je preko pretvornika PD10 (2.1.6) s privzetimi nastavitvami povezave povezal na VS. V terminal je bilo potrebno vpisovati ukaze ter parametre, ki jih je VS razumel. Samo delovanje najbolje prikazuje slika 3.1. Ker pa se nastavitev VS ponavadi izvede samo na začetku uporabe, lahko poznavanje vseh ukazov in parametrov predstavlja težavo za uporabnika. Zato smo se odločili za izvedbo aplikacije, ki bo za uporabnika bolj intuitivna in ne bo zahtevala poznavanja sintakse za nastavljanje. Spisali smo jo v programskem jeziku C# z uporabo razvojnega okolja Visual Studio 2012. To poglavje se osredotoča predvsem na zadnjo različico aplikacije za nastavljanje, saj se je različica s terminalom izkazala za zastarelo.

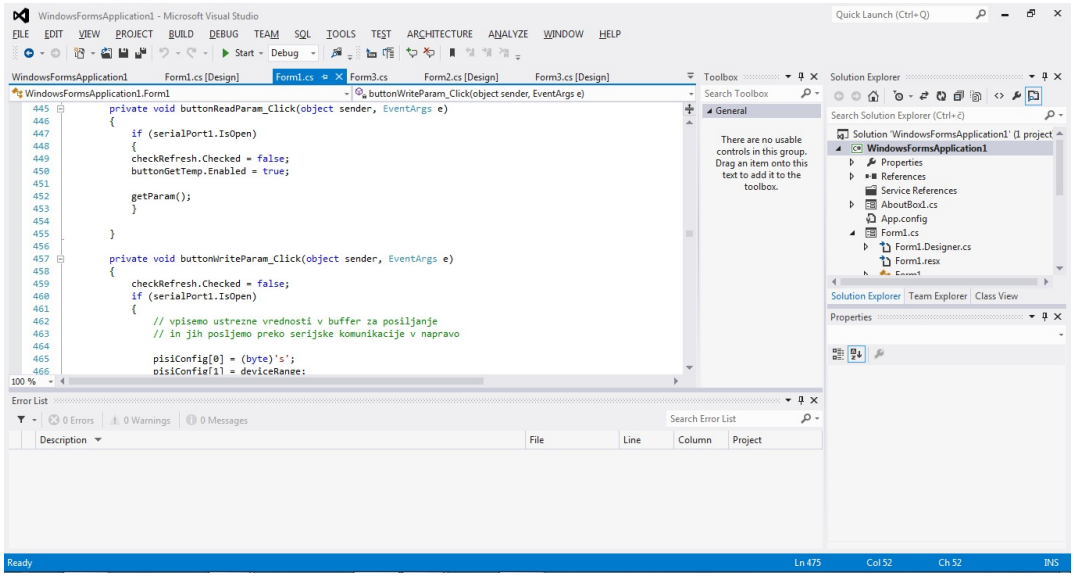


Slika 3.1: Nastavljanje VS z uporabo terminala.

3.1 Razvojno okolje Visual Studio 2012

Visual Studio 2012 je razvojno okolje proizvajalca programske opreme Microsoft. Vgrajeno ima široko podporo za različne programske jezike (C, C++, C#), vsebuje knjižnice in primere, ki nam olajšajo izdelavo aplikacije. Vgrajeni urejevalnik besedila samodejno poskrbi za obarvanje programske kode, tako da je ta bolj berljiva, z uporabo tehnologije *IntelliSense* pa nam pomaga s stalnim ponujanjem predlogov med pisanjem programske kode. Pri izdelavi GUI (ang. *Graphical User Interface*) aplikacije smo uporabili komponento Windows Forms Designer. Uporaba je preprosta in omogoča relativno hitro izgradnjo GUI, saj imamo široko paleto že pripravljenih komponent, ki jih prenašamo v glavno okno in jim dodeljujemo prametre ter funkcije, ki jih posamezna komponenta lahko ima. GUI se z glavnim programom poveže prek

dogodkovnega modela (ang. *event-driven programming model*). To pomeni, da posamezna komponenta GUI lahko sproži dogodek, ki ga moramo v glavnem programu obdelati.



Slika 3.2: Razvojno okolje Visual Studio 2012.

3.2 Delovanje aplikacije

Aplikacija, nameščena na računalniku, za svoje delovanje potrebuje delujočo povezavo z vgrajenim sistemom preko standarda RS-485. Ker so serijska vrata pri novejših računalnikih bolj izjema kot pravilo, moramo uporabiti pretvornik (gl. podpoglavje 2.1.6). Da smo aplikaciji omogočili serijsko komunikacijo, smo uporabili vgrajeno knjižnico (System.IO.Ports), ki vsebuje vse potrebno za delovanje serijske komunikacije. Sama aplikacija ima tako kot VS dva različna načina delovanja. V prvem t. i. konfiguracijskem načinu se znajdemo ob zagonu aplikacije. V tem načinu lahko spreminjamo delovanje VS, kot je opisano

v podpoglavju 3.2.1. V drugem t. i. testnem načinu (gl. podpoglavje 3.2.2) pa je na voljo orodje, s katerim lahko preverjamo, ali se naprava na nastavljene parametre za komunikacijo odziva ali ne, kar nam pomaga pri ugotavljanju, ali je prišlo do napak na samem vodilu.

3.2.1 Konfiguracija VS

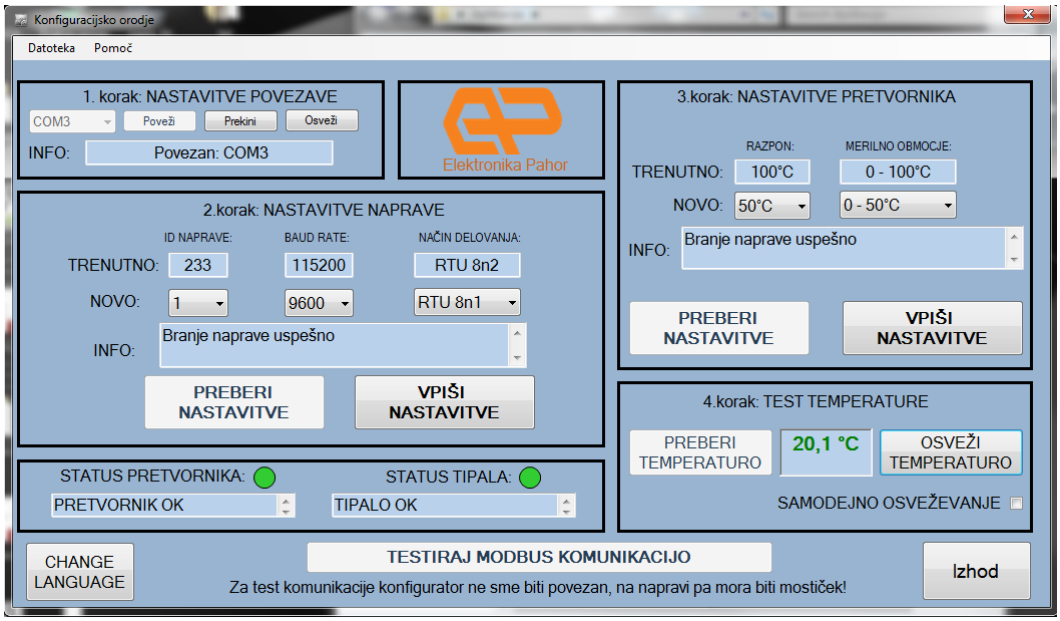
Konfiguracija VS se izvaja po korakih. Najprej je potrebno prek serijskih vrat vzpostaviti povezavo med aplikacijo ter VS. Ko je povezava vzpostavljena, lahko iz VS preberemo parametre povezave (ID naprave, hitrost in različico Modbus-a, ki se izvaja na napravi). Ko se izvede uspešno branje parametrov povezave, lahko te spreminjamo. Če so parametri povezave nastavljeni ustrezno, lahko nadaljujemo z branjem nastavitev pretvornika. Vse nastavitve, ki jih je možno spreminjati z uporabo aplikcije, so prikazane v tabeli 3.1.

Komunikacijske nastavitve	
Parameter	**Vrednost**
Naslov naprave	1 ... 248 (broadcast ni podprt)
Podatkovna hitrost	9600, 10417, 19200, 57600, 115200
Način kodiranja	RTU 8n2, RTU 8n1, ASCII 8n2, ASCII 8n1
Nastavitve pretvornika	
Razpon meritve	**Območje meritve**
50 °C	–20 ... 30 °C, –10 ... 40 °C, 0 ... 50 °C
100 °C	–30 ... 70 °C, –20 ... 80 °C, 0 ... 100 °C
200 °C	–100 ... 100 °C, –50 ... 150 °C, 0 ... 200 °C

Tabela 3.1: Možne nastavitve VS.

Ko je VS nastavljen tako, kot želimo, lahko poženemo testno pretvorbo temperature in s tem preverimo, ali se nastavitve VS in zunanjega pretvornika (MO4021 ali LKM120) ujemajo. Na levi strani aplikacije se nahaja tudi kon-

trolno okno, kjer se izpisujejo morebitne napake zunanjega pretvornika ali nanj priklopljenega tipala. Ob vsaki osvežitvi temperature se osveži tudi status pretvornika, tako da lahko vedno preverimo, kje se nahaja napaka. Aplikacija nam ponuja tudi shranjevanje konfiguracijske datoteke, v katero se zapišejo trenutne nastavitve naprave, prav tako pa lahko preberemo vrednosti iz konfiguracijske datoteke, shranjene v preteklosti. Konfiguracijsko okno predstavlja slika 3.3.

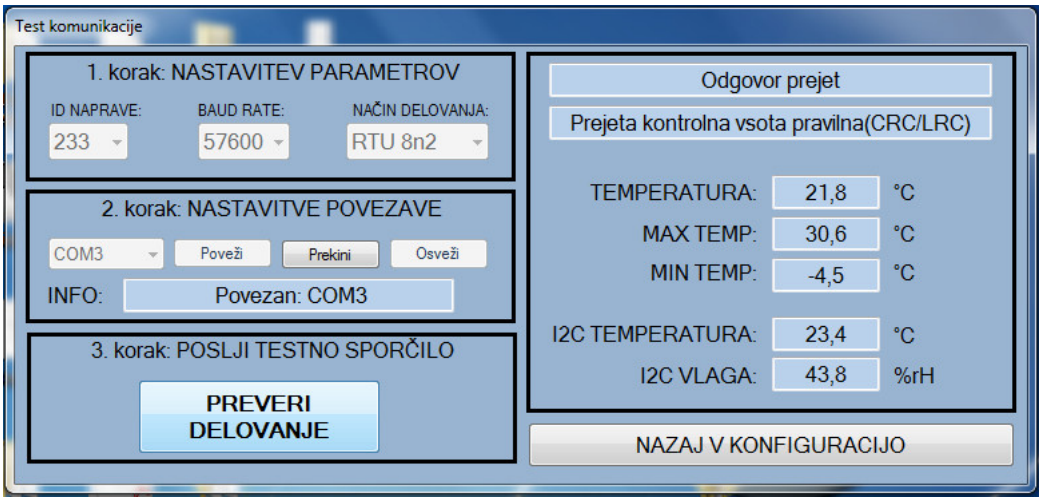


Slika 3.3: Konfiguracijsko okno aplikacije.

3.2.2 Test delovanja Modbus komunikacije

Za delovanje drugega dela aplikacije, ki skrbi za preverjanje delovanja Modbus komunikacije, je potrebno poskrbeti, da se VS nahaja v t. i. `runtime` načinu delovanja ter da je povezava med konfiguratorjem in VS prekinjena (če je povezava vzpostavljena, zagon testnega dela aplikacije ni mogoč). Ko sta zgornja pogoja izpolnjena, lahko začnemo s testom komunikacije. Najprej moramo

izbrati naslov naprave ki jo želimo preizkusiti. Nato določimo podatkovno hitrost in različico Modbus-a, ki se izvaja na VS. Po izbranih nastavitvah se lahko na napravo povežemo in pošljemo testno zahtevo. Aplikacija za testiranje pošlje privzeto zahtevo za branje petih 32-bitnih registrov vgrajenega sistema. Sekundo po pošiljanju zahteve aplikacija preveri odgovor VS. V primeru, da odgovora ni prejela, to sporoči uporabniku in mu svetuje, naj še enkrat preveri nastavitve VS, na katerega se želi povezati. Druga možnost je, da je aplikacija odgovor prejela, vendar je kontrolna vsota napačna. To pomeni, da vgrajeni sistem ne deluje več tako kot bi moral. V primeru da je aplikacija odgovor prejela in je kontrolna vsota ustrezna, se vsebina petih prebranih registrov izpiše v pripadajoča polja.



Slika 3.4: Testno okno aplikacije.

Poglavje 4

Nadzorni sistem

Namen vgrajenega sistema je, da vse svoje meritve posreduje zunanjm napravam v obdelavo in morebitno krmiljenje s temperaturo in vlago povezanih procesov. Zato smo se odločili za povezavo našega VS s prikazovalnikom HMI (ang. *Human Machine Interface*), zmožnim komunikacije prek standarda RS-485 na vodilu Modbus.

4.1 HMI panel FATEK

Za gospodarja vodila, na katerega je priklopljen naš vgrajen sistem, je bil izbran prikazovalnik FV035ST-C10 proizvajalca Fatek. Gre za prikazovalnik z na dotik občutljivim zaslonom velikosti 3,5", resolucijo 1024 × 1024 ter dvema serijskima vhodoma (oba s podporo za standard RS-485). Za preračunavanje podatkov skrbi 32-bitna procesna enota ARM (RISC) [15].

4.2 Razvojno okolje PM Designer

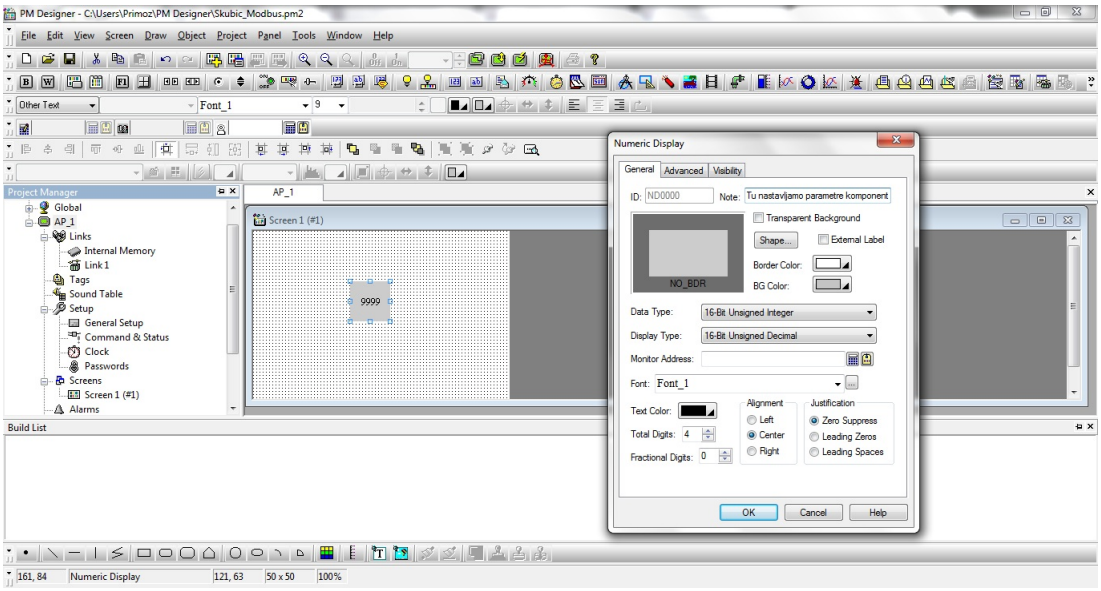
Uporabili smo razvojno okolje PM Designer proizvajalca Fatek, ki je bilo priloženo HMI prikazovalniku. Ob ustvarjanju novega projekta razvojno okolje od nas zahteva, da izberemo, za kateri tip prikazovalnika želimo ustvariti

Slika 4.1: HMI prikazovalnik Fatek FV035ST-C10 [15].

projekt. Glede na izbrani prikazovalnik razvojno okolje onemogoči dodajanje funkcij in gradnikov, ki jih prikazovalnik ne podpira [16].

V osrednjem oknu razvojnega okolja se nahaja urejevalnik zaslona (ang. *screen*), ki je trenutno izbran. Vsi gradniki, ki jih je možno dodati, so v orodni vrstici na zgornjem delu urejevalnika. Izgradnja posameznih zaslonov poteka tako, da v orodni vrstici izberemo gradnik ter ga prenesemo v zaslon (ang. *drag and drop*). Nato mu nastavimo parametre delovanja. Postopek je podoben izdelovanju GUI v okolju Visual Studio (gl. podpoglavje 3.1).

Na levi strani razvojnega okolja se nahaja projektno drevo (ang. *project manager*), ki prikazuje katere komponente sestavljajo projekt. Poleg že omenjenih zaslonov projekt sestavljajo tudi makro ukazi za obdelovanje podatkov, zapisovalniki podatkov (ang. *data logger*), časovniki (ang. *timers*) ter komponenta za upravljanje povezave (ang. *links*) in komponenta za delo z značkami (ang. *tags*).



Slika 4.2: Razvojno okolje PM Designer.

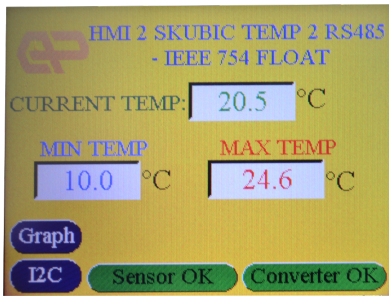
4.3 Delovanje nadzornega sistema

Za delovanje nadzornega sistema je potrebno pravilno nastaviti parametre povezave. To storimo v komponenti za upravljanje povezave. Prikazovalnik moramo nastaviti tako, da je gospodar vodila, podatke pa od sužnjev pridobiva prek različice Modbus RTU. Nastaviti pa je potrebno tudi podatkovno hitrost, število podatkovnih bitov, pariteto ter število stop bitov. Vse naprave, ki jih povežemo na vodilo, morajo imeti te nastavitve enake nastavitvam gospodarja vodila.

Prikazovalnik zahteve za branje generira na podlagi vnosov v urejevalnik značk. Zato je potrebno vsakemu podatku, ki ga želimo pridobiti od sužnja, definirati značko. To storimo tako, da izbranemu podatku določimo ime, naslov naprave ter naslov podatka v napravi, s katere želimo prebrati podatek. Izbrati moramo tudi, za kakšen tip podatka gre (16-bitno celo število, 32-bitno število

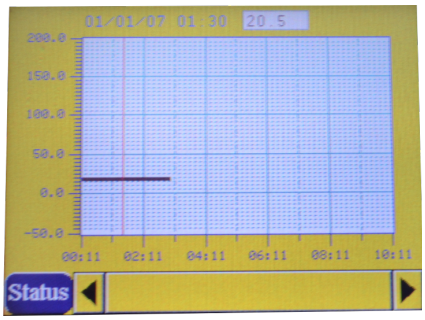
v plavajoči vejici ...). Če želimo podatke naknadno obdelovati, to storimo z zapisom makro ukazov.

Nadzorni sistem sestavljajo trije nadzorni zasloni. Prvi zaslon prikazuje trenutno ter maksimalno in minimalno temperaturo. Vse tri prikazane temperature je vgrajen sistem pridobil s preračunano pretvorbo izhodnega toka zunanjega pretvornika RI. Na zaslonu se nahajata tudi dva prikazovalnika statusa vgrajenega sistema, ki opozorita na morebitne napake pri pretvorbi.



Slika 4.3: Nadzorni zaslon s prikazom temperatur in statusa pretvornika.

Na drugem zaslonu se izrisuje potek trenutne temperature, izmerjene z uporabo zunanjega pretvornika.

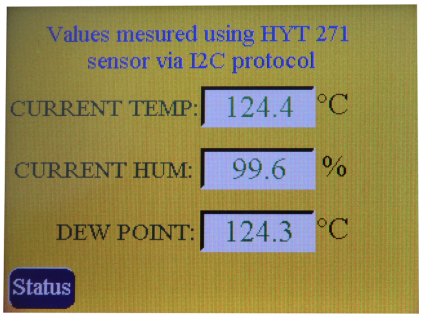


Slika 4.4: Prikazovalnik poteka temperature.

Če želimo prikazovati zgodovino poteka temperature, moramo uporabiti komponento prikazovalnika za zapisovanje podatkov, ki trenutno temperaturo shrani v podatkovno bazo vsakih 10 sekund. Kapaciteta podatkvne baze zadošča za hrambo 1000 temperatur, kar zadostuje za zgodovino meritev zadnjih 2,7 ur.

Tretji zaslon prikazuje temperaturo in vlago, ki jo vgrajen sistem pridobi s preračunanjem podatkov, pridobljenih od tipala HYT 271. Prav tako prikazuje tudi temperaturo rosišča (ang. *dew point*), ki jo prikazovalnik izračuna s pomočjo makro ukazov. Za izračun točke rosišča je bila uporabljena enačba 4.1:

$$rosisce[°C] = temperatura - \frac{100 - vlaga}{5} \tag{4.1}$$



Slika 4.5: Nadzorni zaslon tipala HYT 271.

Poglavje 5

Sklepne ugotovitve

Eden glavnih motivov za izvedbo projekta izdelave vgrajenega sistema je bila zagotovo želja, spoznati težave in posledično tudi rešitve, na katere naletimo pri izdelavi vgrajenega sistema. Na začetku je bila največja ovira pomanjkljivo znanje s področja elektrotehničnih omejitev pri sami izdelavi, ki pa se je izpopolnjevalo z vsakim uspešno rešenim problemom. Med samim izdelovanjem pa se pojavljajo tudi nove ideje za izboljšanje vgrajenega sistema. Zato je prva revizija tiskanega vezja opremljena z razširitvenim konektorjem, ki omogoča priklop t. i. razširitvenih kartic (ang. *daughter board*) in s tem povečanje nabora funkcij vgrajenega sistema.

Nekaj težav nam je povzročal tudi mikrokrmilnik oziroma programiranje mikrokrmilnika v zbirniku. Za programiranje v zbirniku smo se odločili na podlagi optimizacije delovanja vgrajenega sistema, delno pa nas je v to prisilil mikrokrmilnik z majhno kapaciteto pomnilnika namenjenega programskim ukazom. Čeprav izkušenj iz programiranja ne manjka, zna biti reševanje matematičnih problemov v zbirniku z omejenim naborom ukazov zahtevno opravilo. Zato bi bilo smiselno – v primeru nadaljnega razvoja vgrajenega sistema – uporabiti zmogljivejši mikrokrmilnik PIC 18F2520 [18], ki poleg preostalih lastnosti ponuja tudi arhitekturo, optimizirano za uporabo s C prevajalniki. Tudi za ta

mikrokrmilnik obstaja velika razvijalska skupnost, prednost pa je tudi to, da obstajajo odprtokodne rešitve za uporabo naprednih mikrokrmilnikovih funkcij, saj je to mikrokrmilnik, namenjen izdelavi malo bolj zahtevnih vgrajenih sistemov.

Veliko možnosti je tudi za izboljšave na področju merjenja temperature. Prva, ki najbolj izstopa, je eliminacija uporabe zunanjega pretvornika RI oziroma njegova zamenjava s pretvornikom na razširitveni kartici. Vlogo pretvornika bi prevzel čip XTR 108 [21], na katerega lahko neposredeno priklopimo uporovno tipalo (Pt100, Pt1000). Ponuja nam tako tokovni kot tudi napetostni izhod pretvorjene upornosti, njegovo delovanje pa je možno krmiliti prek vmesnika SPI. Nadaljne izboljšave je možno doseči z uporabo zunanjega pretvornika AD z višjo resolucijo, kar bi pomenilo večjo natančnost meritev tudi pri širšem merilnem območju. Ena od možnosti je uporaba pretvornika ADS1100 proizvajalca Burr-Brown, ki se ponaša s 16-bitno resolucijo ter komunikacijo s preostalimi napravami preko vodila I^2C [19]. Za zagotovitev stabilnejše meritve – ne glede na zunanje vplive (temperatura mikrokrmilnika ima velik vpliv na natančnost reference) – bi bilo smiselno uporabiti zunanjo referenco. Že med samim načrtovanjem je bilo predvideno, da se vgrajenemu sistemu doda referenčni modul LM4030 [20], vendar se do sedaj to še ni izkazalo kot potrebno. Res pa je, da je bilo le malo testiranj sistema, izvedenih izven razvojne pisarne, zato ta ocena ni povsem realna.

Za izboljšanje komunikacijskih sposobnosti vgrajenega sistema bi bilo smiselno med mikrokrmilnik in gonilnik RS-485 dodati izolatorski čip. Z galvansko ločitvijo linije od mikrokrmilnika bi lahko preprečili škodljivim motnjam, ki se prenašajo po linijah, da poškodujejo mikrokrmilnik.

Kot lahko vidimo, je možnosti za izboljšanje vgrajenega sistema še veliko. Vendar pa je potrebno premisliti, katere izboljšave so smiselne in bi izboljšale

uporabnost sistema, katere pa bi delovanje samega sistema le otežile. Nekaterim spremembam sistema botruje tudi cenovna dostopnost strojnih komponent, ki so z uporabo tujih dobaviteljev velikokrat bolj ugodne kot pa pri slovenskih dobaviteljih. Zagotovo pa bo eden največjih pokazateljev, kaj je potrebno spremeniti in kaj izboljšati, izčrpno preizkušanje vgrajenega sistema v realnih okoliščinah in ne le na razvijalčevi mizi.

Literatura

[1] Microchip, “PIC 16F1827 datasheet”. Dostopno na:
http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en538963

[2] Microchip “MPLAB ICD3 User’s guide”. Dostopno na:
http://ww1.microchip.com/downloads/en/DeviceDoc/51766B.pdf

[3] Microchip “AC244046 Debug header”. Dostopno na:
http://www.farnell.com/datasheets/1605550.pdf

[4] Texas Instruments “SN75176B Differential bus transciever”. Dostopno na:
http://www.datasheetcatalog.org/datasheet/texasinstruments/sn65176b.pdf

[5] Texas Instruments “RS-422 and RS-485 Standards Overview and System Configurations”
Dostopno na: http://www.ti.com/lit/an/slla070d/slla070d.pdf

[6] LKMelectronic “LKM120 datasheet”. Dostopno na:
http://www.lkmelectronic.de/english/pdfs/kopfmontage/datenblaetter/Typ120.pdf

[7] IST “Digital humidity sensor HYT 271”. Dostopno na:
http://www.farnell.com/datasheets/1681510.pdf

[8] IST “HYT 271 protocol description”. Dostopno na:
http://www.servoflo.com/environmental-sensors/humidity-sensors/digital-humidity-sensors/256-humidity/833-i2c-protocol-hyt.html

[9] LUMEL "USB/RS485 converter PD10". Dostopno na: http://www.lumel.com.pl/en/download/elements_of_integration_systems/pd10/

[10] Microchip "MPLAB IDE user's guide". Dostopno na: http://ww1.microchip.com/downloads/en/devicedoc/51519a.pdf

[11] Microchip "Using the MSSP module for master i2c communication". Dostopno na: http://ww1.microchip.com/downloads/en/AppNotes/00735a.pdf

[12] Microchip "CRC generating and checking". Dostopno na: http://ww1.microchip.com/downloads/en/AppNotes/00730a.pdf

[13] Modbus "Modbus application protocol specification". Dostopno na: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[14] Modbus-IDA "Modbus over Serial Line". Dostopno na: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[15] Fatek "Fatek FV series operator terminal". Dostopno na: http://www.globalautomation.com.au/products/control/fatekview/FV035ST-C10.pdf

[16] Fatek "Fatek View Software manual". Dostopno na: http://www.globalautomation.com.au/products/control/fatekview/FV_Manual.pdf

[17] Welwyn Components "Precision Metal Film Resistors". Dostopno na: http://www.farnell.com/datasheets/66234.pdf

[18] Microchip "PIC 18F2520 data sheet". Dostopno na: http://ww1.microchip.com/downloads/en/DeviceDoc/39631E.pdf

[19] Burr-Brown Products "ADS1100 Analog to digital converter". Dostopno na: http://www.ti.com/lit/ds/symlink/ads1100.pdf

[20] TI "LM4030 SOT-23 Ultra-High Precision Shunt Voltage Reference". Dostopno na: http://www.farnell.com/datasheets/1642270.pdf

[21] Burr-Brown Products “XTR108 data sheet”. Dostopno na:
http://www.ti.com/lit/ds/symlink/xtr108.pdf