Programiranje mikroupravljača u ugradbenim sustavima - Programiranje mikroupravljača ATmega328P

Vrhovski, Zoran; Radočaj, Danijel

Authored book / Autorska knjiga

Publication status / Verzija rada: **Published version / Objavljena verzija rada (izdavačev PDF)**

Publication year / Godina izdavanja: **2022**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:144:180577

Download date / Datum preuzimanja: **2025-02-22**

Repository / Repozitorij:

Digital Repository of Bjelovar University of Applied Sciences

Zoran Vrhovski, Danijel Radočaj
PROGRAMIRANJE MIKROUPRAVLJAČA U UGRADBENIM SUSTAVIMA
Bjelovar, 2022.

dr. sc. Zoran Vrhovski, Danijel Radočaj, mag. inž. meh.

PROGRAMIRANJE MIKROUPRAVLJAČA U UGRADBENIM SUSTAVIMA

Programiranje mikroupravljača ATmega328P

Tehnički urednici:
dr. sc. Zoran Vrhovski
Danijel Radočaj, mag. inž. meh.

Izrada shema:
Goran Benkek, struč. spec. ing. el.

Prijelom i oblikovanje naslovnice / grafički urednik:
dr. sc. Zoran Vrhovski

Udžbenik je financiran iz projekta:
Broj/naziv: UP.03.3.1.04.0002/ "Razvoj kompetencija kroz učenje temeljeno na radu"
Korisnik: Obrtnička škola Koprivnica

ISBN: 978-953-7676-41-4

Veleučilište u Bjelovaru © 2022.

Bez pisanog odobrenja nositelja autorskog prava niti jedan dio ove publikacije ne smije se reproducirati, emitirati ni distribuirati u bilo kojem obliku ili bilo kojim načinom, elektroničkim ili mehaničkim sredstvima, osim na zakonom propisan način.

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Sadržaj

1 Programsko razvojno okruženje *Microchip Studio* **3**

1.1 Prvi projekt u programskom razvojnom okruženju *Microchip Studio* 3

2 Razvojno okruženje s mikroupravljačem ATmega328P **11**

3 Programiranje mikroupravljača ATmega328P **17**

3.1 ISP programiranje mikroupravljača ATmega328P 17

3.2 Programiranje mikroupravljača ATmega328P pomoću Arduino *Bootloader* programa 27

4 Digitalni izlazi mikroupravljača ATmega328P **31**

4.1 Vježbe - digitalni izlazi mikroupravljača ATmega328P 32

5 Digitalni ulazi mikroupravljača ATmega328P **61**

5.1 Vježbe - digitalni ulazi mikroupravljača ATmega328P 63

6 LCD displej **91**

6.1 Vježbe - LCD displej 91

7 Analogno-digitalna pretvorba **131**

7.1 Vježbe - analogno-digitalna pretvorba 132

8 Prekidi mikroupravljača **147**

9 Tajmeri i brojači **151**

9.1 Vježbe - tajmeri i brojači u normalnom načinu rada 155

10 Pulsno širinska modulacija **189**

10.1 Vježbe - PWM načini rada tajmera 191

11 Univerzalna asinkrona serijska komunikacija **221**

11.1 Vježbe - UART komunikacija 223

12 Vanjski prekidi **257**

12.1 Vježbe - vanjski prekidi 257

13 Povezivanje odabranih elektroničkih modula na mikroupravljač **275**

13.1 Rotacijski enkoder 275
13.2 Tranzistor kao sklopka i relej 284
13.3 Ultrazvučni senzor HC-SR04 290
13.4 Numerički displej i posmačni registri 300
13.5 Servomotor 315
13.6 RGB dioda 324

14 Ostale značajke mikroupravljača **339**
14.1 EEPROM memorija 339
14.2 Watchdog tajmer 345
14.3 Sleep modovi rada i upravljanje potrošnjom energije mikroupravljača 349
14.4 Analogni komparator 353
14.5 I2C komunikacija 356

Bibliografija **363**

Predgovor

Digitalni udžbenik PROGRAMIRANJE MIKROUPRAVLJAČA U UGRADBENIM SUSTAVIMA nastao je kao ishod projekta *Razvoj kompetencija kroz učenje temeljeno na radu* koji je sufinancirala Europska unija iz Europskog socijalnog fonda. Nositelj projekta bila je Obrtnička škola Koprivnica. Izrada digitalnog udžbenika za program iz područja mikroupravljača i ugradbenih sustava ishod je koji su ostvarili zaposlenici (autori) Zoran Vrhovski i Danijel Radočaj Veleučilišta u Bjelovaru koje je bilo partner u projektu.

Navedeni digitalni udžbenik literatura je razvijenom programu usavršavanja za poslove programiranja mikroupravljača koji je razvijen u sklopu projekta *Razvoj kompetencija kroz učenje temeljeno na radu.* Udžbenik se bavi razvojem programa za mikroupravljač ATmega328P u programskom okruženju *Microchip Studio.* Programska rješenja napisana su u programskom jeziku C koji je najrašireniji programski jezik u svijetu programiranja mikroupravljača.

Sva programska rješenja koja se nalaze u ovom udžbeniku razvijena su u programskom okruženju *Microchip Studio* i nalaze se na mrežnoj stranici `www.vub.hr/atmega328p`. U udžbeniku su opisana rješenja ukupno 50 vježbi koje su vrlo praktične i daju dobar uvod u svijet razvoja programske podrške za ugradbene računalne sustave.

Digitalni udžbenik PROGRAMIRANJE MIKROUPRAVLJAČA U UGRADBENIM SUSTAVIMA nastao je zajedničkim radom dvojice autora: dr. sc. Zorana Vrhovskog i Danijela Radočaja, mag. inž. meh. Poglavlja koja je napisao autor Zoran Vrhovski su:

- Programsko razvojno okruženje *Microchip Studio*
- Razvojno okruženje s mikroupravljačem ATmega328P
- Programiranje mikroupravljača ATmega328P
- Digitalni izlazi mikroupravljača ATmega328P
- Digitalni ulazi mikroupravljača ATmega328P
- LCD displej
- Analogno-digitalna pretvorba
- Prekidi mikroupravljača
- Tajmeri i brojači
- Pulsno širinska modulacija
- Univerzalna asinkrona serijska komunikacija

- Vanjski prekidi
- Ostale značajke mikroupravljača
  - EEPROM memorija
  - Whatchdog tajmer
  - Sleep modovi rada i upravljanje potrošnjom energije
  - Analogni komparator
  - I2C komunikacija

Poglavlje koje je napisao autor Danijel Radočaj je:

- Povezivanje odabranih elektroničkih modula na mikroupravljač
  - rotacijski enkoder,
  - tranzistor kao sklopka i relej,
  - ultrazvučni senzor HC-SR04,
  - numerički displej i posmačni registar,
  - servomotor i
  - RGB dioda.

Zahvaljujemo se kolegi Goranu Benkeku koji nam je ustupio sheme didaktičkog učila koje je izradio u sklopu projekta *Razvoj kompetencija kroz učenje temeljeno na radu*. Didaktičko učilo prikazano je i opisano u poglavlju *Razvojno okruženje s mikroupravljačem ATmega328P*.

Autori:
Zoran Vrhovski
Danijel Radočaj

Poglavlje 1

Programsko razvojno okruženje *Microchip Studio*

Razvoj programskih rješenja mikroupravljača zahtjeva korištenje programskih razvojnih okruženja za programiranje mikroupravljača. Na tržištu su danas brojna razvojna okruženja za programiranje mikroupravljača. Pri odabiru programskog razvojnog okruženja za programiranje mikroupravljača valja voditi računa o porodici mikroupravljača koju koristite u svojim rješenjima. Ova knjiga bavi se programiranjem mikroupravljača ATmega328P koji je porodice AVR. Proizvođač AVR mikroupravljača je *Microchip*[1].

Mikroupravljač ATmega328P moguće je programirati u brojnim programskim razvojnim okruženjima, a neka od njih su *Microchip Studio*, MPLAB, *Arduino IDE*, *CodeVision* i drugi.

Programska razvojna okruženja *Microchip Studio* i MPLAB razvila je tvrtka *Microchip* pri čemu je programsko razvojno okruženje MPLAB, uz programiranje mikroupravljača porodice AVR, namijenjeno programiranju i mikroupravljača porodice PIC i dsPIC. Vrlo popularno programsko razvojno okruženje *Arduino IDE* može se koristiti za programiranje mikroupravljača porodice AVR, no pri tome se treba odmaknuti od tradicionalnog programiranja u *Arduino* programskom okruženju te programirati u programskom jeziku C ili C++.

Od navedenih programskih razvojnih okruženja za programiranje mikroupravljača porodice AVR odabrat ćemo razvojno okruženje *Microchip Studio* u kojem ćemo razvijati programska rješenja za mikroupravljač ATmega328P. Razlog tomu je što je ovo programsko razvojno okruženje besplatno, otvoreno i koristi sve značajke razvojnog okruženja za programiranje *Visual Studio*. Najnoviju inačicu programskog razvojnog okruženja *Microchip Studio* možete preuzeti na poveznici `https://www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices`.

1.1 Prvi projekt u programskom razvojnom okruženju *Microchip Studio*

Microchip Studio je integrirano razvojno okruženje (engl. *Integrated Development Environment (IDE)*) za razvijanje te otklanjanja pogrešaka programa na mikroupravljačima porodice AVR i SAM. Podržava pisanje programa u programskim jezicima C i C++. Prethodni naziv programskog razvojnog okruženja *Microchip Studio* bio je *Atmel Studio*. Iako programsko razvojno okruženje *Microchip Studio* dolazi s novim imenom i novim izgledom, za učenje

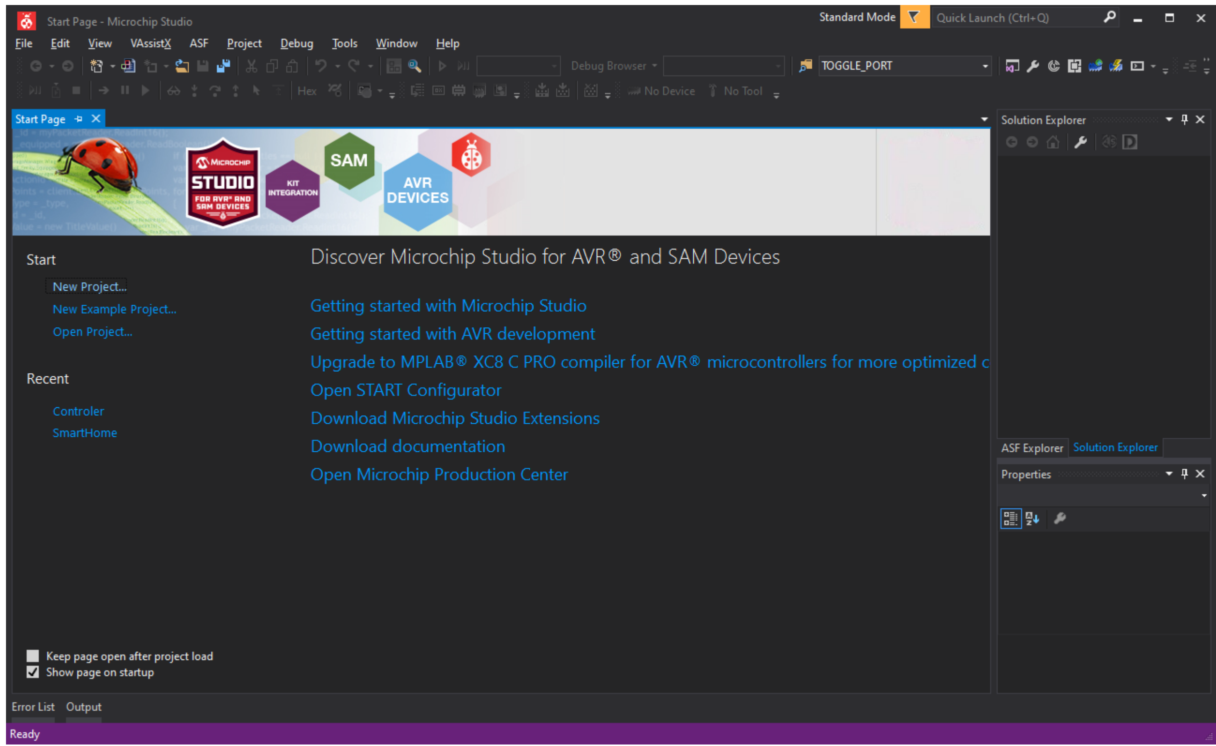
[1] Do 2016. godine AVR mikroupravljače proizvodila je tvrtka Atmel koju je kupila tvrtka Microchip. Iz tog je razloga na tržištu moguće naći mikroupravljače porodice AVR koji imaju logotip tvrtke Atmel.

programiranja u programskom razvojnom okruženju *Microchip Studio* može se koristiti sva dokumentacija programskog razvojnog okruženja *Atmel Studio*.

Programsko razvojno okruženje *Microchip Studio* pokreće se dvostrukim klikom na ikonu sa slike 1.1.

Slika 1.1: *Microchip Studio* - ikona

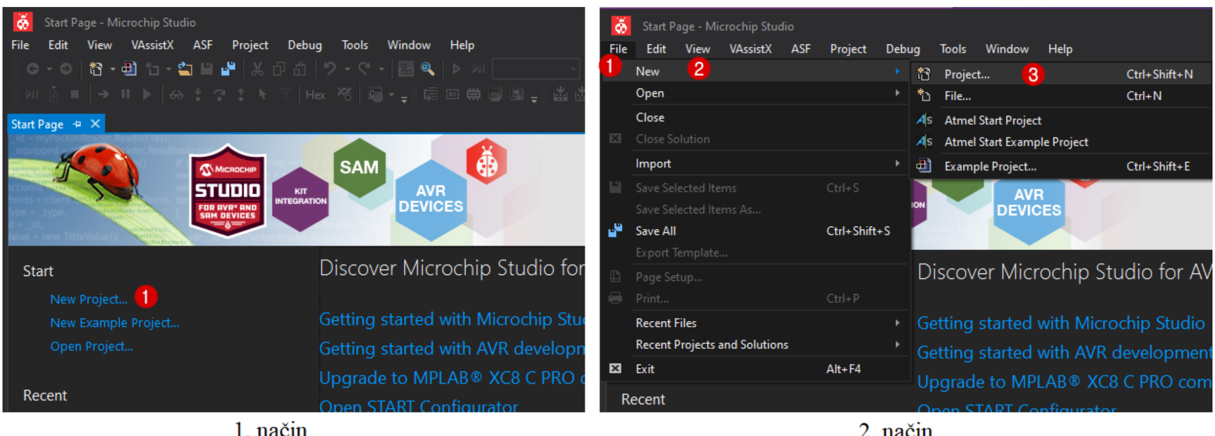
Pri pokretanju programskog razvojnog okruženja *Microchip Studio*, otvorit će se početni prozor prikazan na slici 1.2.



Slika 1.2: Programsko razvojno okruženje *Microchip Studio* - početni prozor

Unutar početnog prozora možemo stvoriti novi projekt, otvoriti postojeći projekt ili otvoriti projekte na kojima ste radili u prethodnom razdoblju. Novi projekt u programskom razvojnom okruženju *Microchip Studio* možemo kreirati na jedan od dva načina:

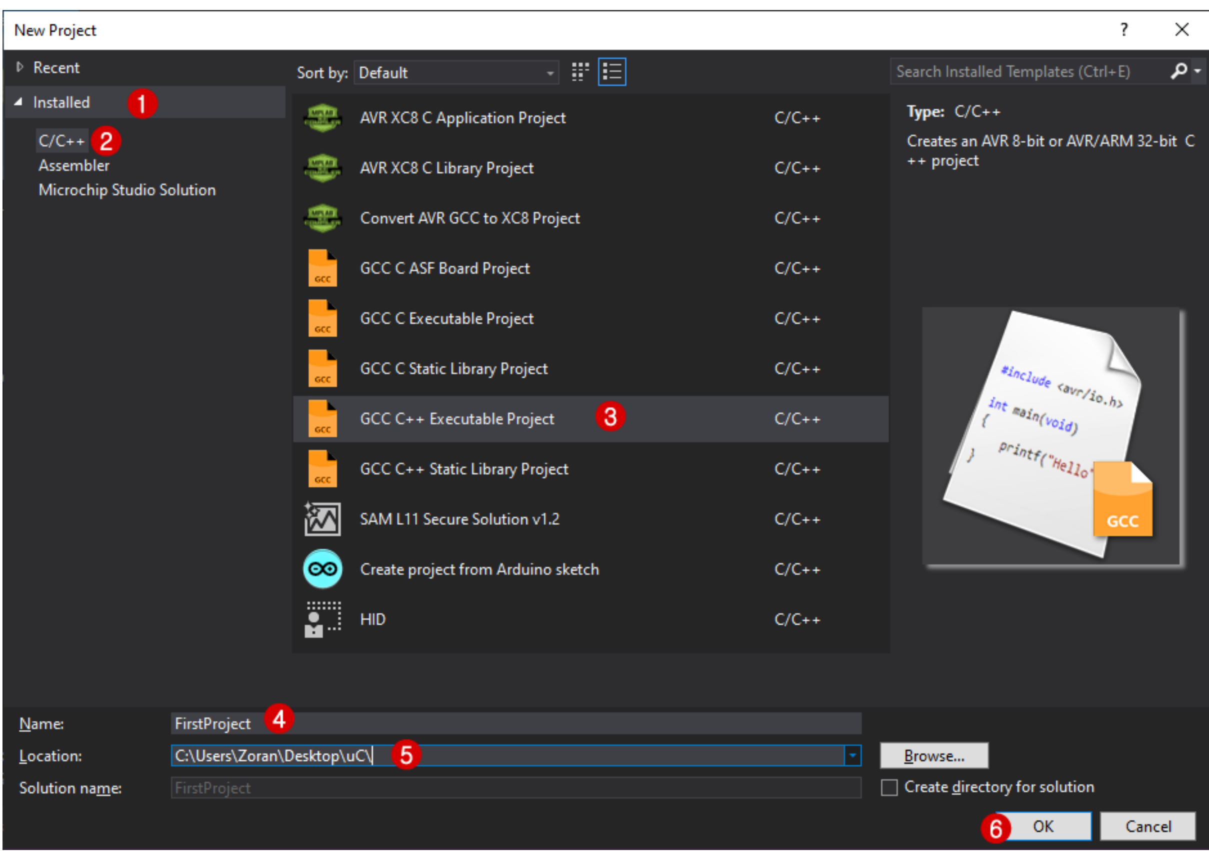
- Prvi način
    1. u početnom prozoru odaberite *New Project...* (slika 1.3, 1. način),
- Drugi način
    1. u početnom prozoru u izborniku odaberite *File*,
    2. u izborniku *File* odaberite podizbornik *New*,
    3. u podizborniku *New* odaberite *Project...* (slika 1.3, 2. način).



1. način 2. način

Slika 1.3: Programsko razvojno okruženje *Microchip Studio* - početni prozor

Prvim ili drugim načinom pokrenut ćete stvaranje novog projekta pri čemu će se otvoriti prozor prikazan na slici 1.4.



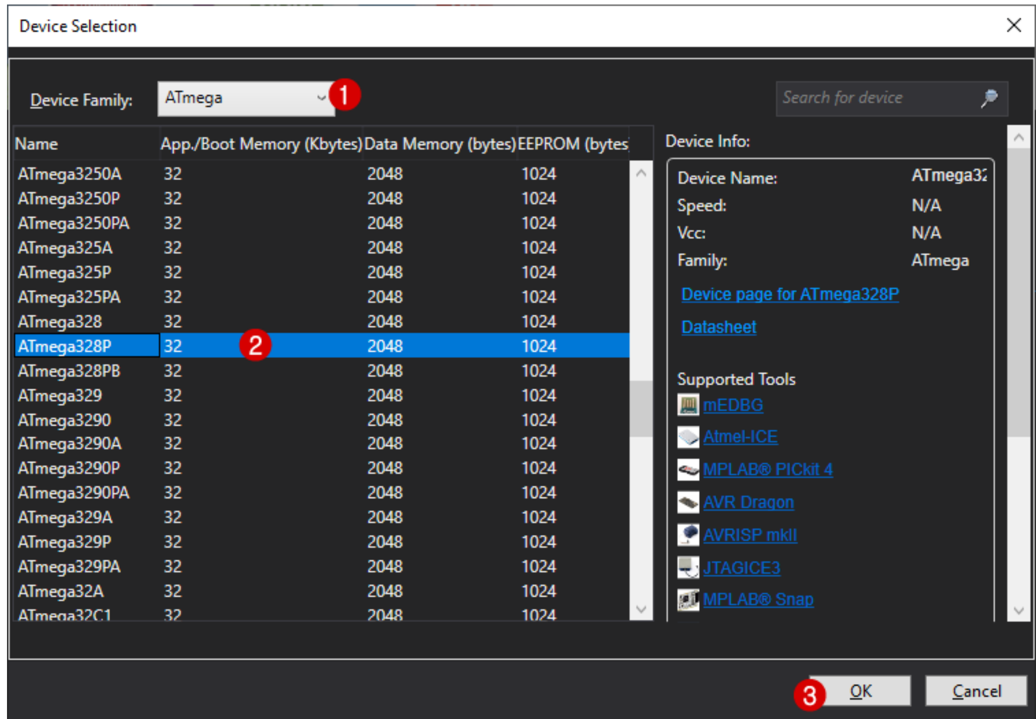
Slika 1.4: Programsko razvojno okruženje *Microchip Studio* - prozor za unos tipa, imena i lokacije projekta

U prozoru sa slike 1.4 potrebno je provesti korake označene brojevima od 1 do 6:

1. odaberite izbornik *Installed Templates*,
2. u izborniku *Installed Templates* odaberite C/C++,
3. u popisu tipova projekata odaberite *GCC C++ Executable Project*,

4. unesite ime projekta (npr. `FirstProject`),
5. unesite putanju (lokaciju) projekta na svom računalu,
6. odaberite OK za nastavak stvaranje projekta.

Nakon unosa tipa, imena i lokacije projekta na slici 1.4, otvara se novi prozor u kojem je moguće odabrati mikroupravljač kojeg ćemo programirati (slika 1.5).

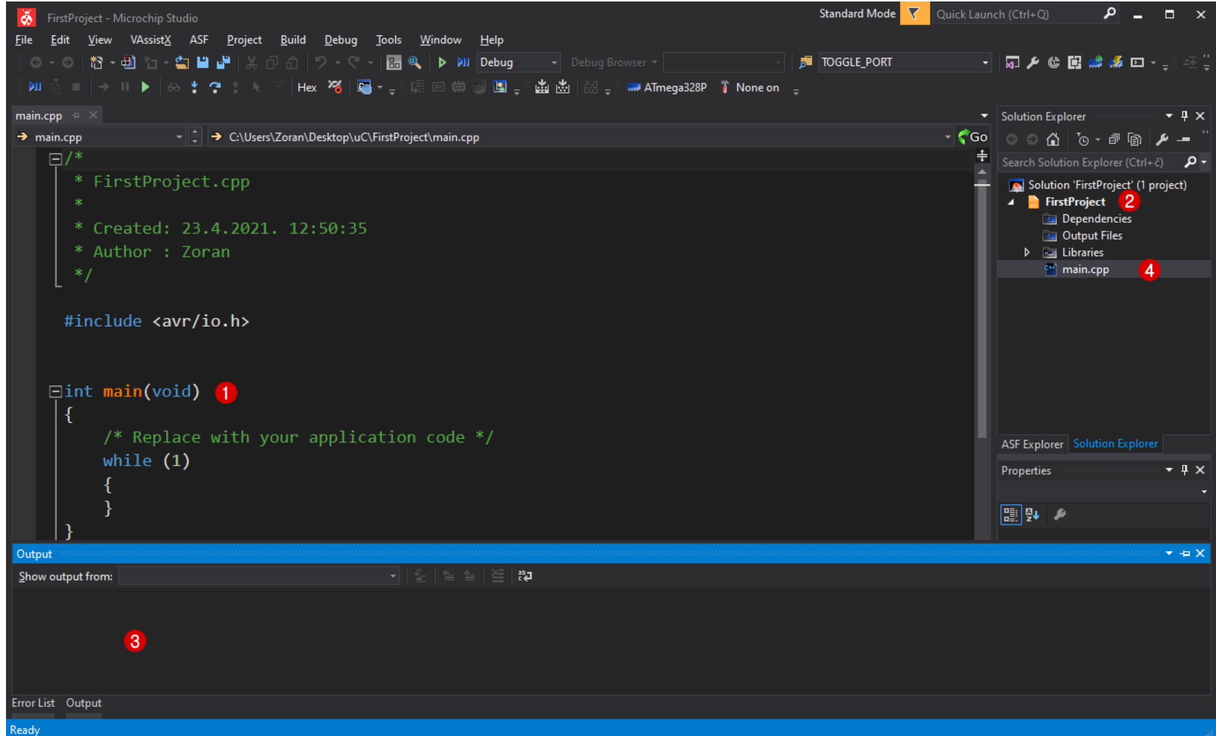


Slika 1.5: Programsko razvojno okruženje *Microchip Studio* - odabir mikroupravljača

Odabir mikroupravljača prikazan je na slici 1.5 koracima od 1 do 4 :

1. odaberite porodicu AVR mikroupravljača (npr. ATmega).
2. odaberite mikroupravljač iz odabrane AVR porodice (npr. ATmega328P).
3. odaberite OK za nastavak kreiranja projekta.

Korak 1 i 2 mogu se preskočiti ukoliko poznajete točno ime mikroupravljača. Ime mikroupravljača (npr. ATmega328P) možete upisati u tražilicu (*Search for device*) na slici 1.5. Nakon odabira mikroupravljača za koji ćete pisati programsko rješenje, otvorit će se uređivač programskog koda prikazan na slici 1.6.



Slika 1.6: Programsko razvojno okruženje *Microchip Studio* - uređivač programskog koda

Uređivač programskog koda sadrži sljedeće elemente označene brojevima na slici 1.6:

1. tekstualni uređivač programskog koda,
2. projektno stablo s nazivom projekta koji je stvoren,
3. pokaznik statusnih poruka i
4. programska datoteka koja uključuje `main()` funkcijom (naziv programske datoteke je main.cpp, a može se promijeniti).

U tekstualni uređivač koda piše se programski kod u programskom jeziku C ili C++. Projektno stablo s nazivom projekta koji je stvoren služi za strukturiranje programskog koda u zaglavlja. Statusne poruke prikazuju upozorenja i pogreške koje se javljaju prilikom prevođenja programskog jezika C ili C++ u strojni kod. Ove poruke pomažu nam u otklanjanju sintaksnih pogrešaka koje su napravljene prilikom pisanja programskog koda. Programska datoteka s `main()` funkcijom kreira se automatski pri kreiranju projekta. Ova programska datoteka sadrži `main()` funkciju, beskonačnu petlju u `main()` funkciji te uključuje zaglavlje `avr/io.h` koje sadrži osnovne definicije registara i konstanti za mikroupravljač koji se koristi u projektu.

Zadatak prvog projekta kojeg ćemo stvoriti u programskom razvojnom okruženju *Microchip Studio* jest:

- uključiti LED diode spojene na pinove PB1, PB2 i PB3 ako je pritisnuto tipkalo spojeno na pin PD2,
- isključiti LED diode spojene na pinove PB1, PB2 i PB3 ako nije pritisnuto tipkalo spojeno na pin PD2.

Riješenje zadanog zadatka prikazano je programskim kodom 1.1.

Programski kod 1.1: Sadržaj datoteke `FirstProject.cpp`

```
/*
 * FirstProject.cpp
 *
 * Created: 23.4.2021.
 * Authors: Zoran Vrhovski
 */

#include <avr/io.h>
// definiranje pozicija pinova PB1, PB2, PB3 i PD2
#define PB1 1
#define PB2 2
#define PB3 3
#define PD2 2

int main(void) {

    // Postavljanje pinova PB1, PB2 i PB3 kao izlazni
    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);
    // Postavljanje pina PD2 kao ulazni
    DDRD &= ~(1 << PD2);
    //Ukljucenje pull up otpornika na PD2
    PORTD |= (1 << PD2);

    while (1) {
        // ako je pritisnuto tipkalo PD2 (PD2 = 0)
        if ((PIND & (1 << PD2)) == false) {
            // ukljuci LED diode PB1, PB2 i PB3
            PORTB |= (1 << PB1) | (1 << PB2) | (1 << PB3);
        } else {
            // iskljuci LED diode PB1, PB2 i PB3
            PORTB &= ~((1 << PB1) | (1 << PB2) | (1 << PB3));
        }
    }
}
```

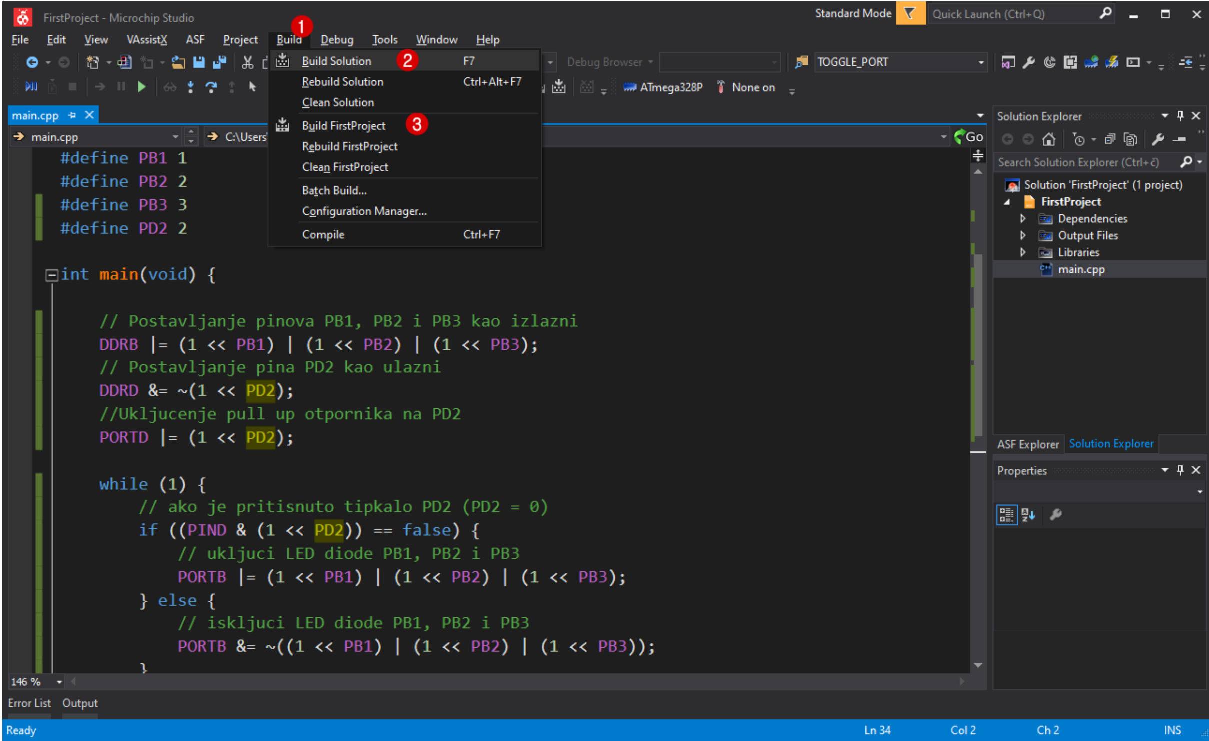
Cilj programskog koda 1.1 nije ići u detalje izvedbe problemskog zadatka koji uključuje tri LED diode pritiskom na tipkalo, već pokazati na koji način se programski kod prevodi u strojni kod pomoću programskog razvojnog okruženja *Microchip Studio*. Način kako se konfiguriraju digitalni ulazi i izlazi za potrebe korištenja LED dioda i tipkala na mikroupravljaču obradit ćemo detaljno u sljedećim poglavljima.

Prilikom testiranja programskog rješenja potrebno je napisani programski kod prevesti u strojni kod. Prevođenje napisanog programskog koda u strojni kod u programskom razvojnom okruženju *Microchip Studio* može se napraviti na jedan od tri načina:

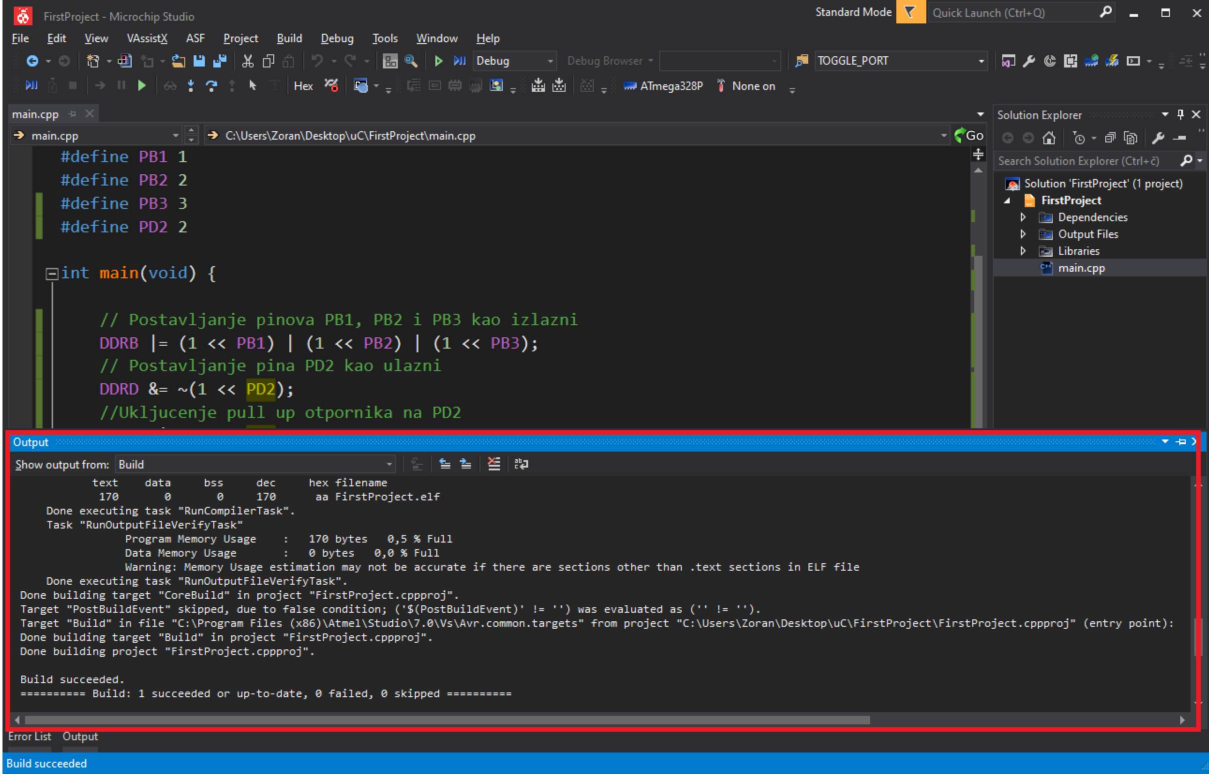
- pritisnite tipku *F7* ili
- u izborniku odaberite *Build* → *Build Solution* (slika 1.7 koraci 1 i 2) ili
- u izborniku odaberite *Build* → *Build FirstProject* (slika 1.7 koraci 1 i 3).

Treći način se u pravilu koristi ako u projektnom stablu unutar jednog rješenja (engl. *Solution*) imamo više od jednog projekta.

Napisani programski kod uspješno je preveden u strojni kod ako se u pokazniku statusnih poruka (slika 1.8) pojavi poruka „Uspješno prevođenje" (engl. *Build succeeded*). Kada prevođenje programskog koda u strojni kod nije uspješno, tada će u pokazniku statusnih poruka biti navedene pogreške koje je potrebno ispraviti, a sve u skladu sa sintaksom programskog jezika C ili C++.



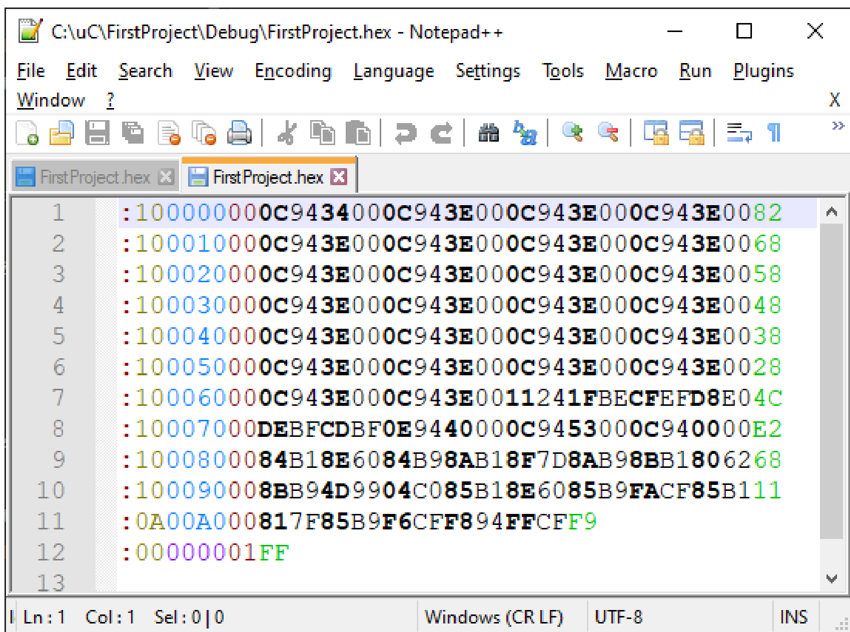
Slika 1.7: Programsko razvojno okruženje *Microchip Studio* - postupak prevođenja programskog koda u strojni jezik



Slika 1.8: Programsko razvojno okruženje *Microchip Studio* - ispis u pokazniku statusnih poruka nakon prevođenja programskog koda u strojni jezik

Strojni kod koji je nastao prevođenjem programskog koda prikazan je na slici 1.9. Datoteka

sa strojnim kodom nalazi se na lokaciji projekta (u našem slučaju `C:\uC\FirstProject\Debug`). Ekstenzija datoteke sa strojnim kodom jest `*.hex`. Strojni kod prikazan na slici 1.9 u heksadecimalnom je zapisu pri čemu prvih osam heksadecimalnih znamenaka u retku datoteke predstavljaju adresu memorijske lokacije programske memorije mikroupravljača na koju se sprema strojni kod. Programiranje mikroupravljača provodi se pomoću softvera za programiranje koji će strojni kod upisati u programsku memoriju mikroupravljača [1]. Način na koji se strojni kod "spušta" na mikroupravljač biti će prikazan u narednim poglavljima.

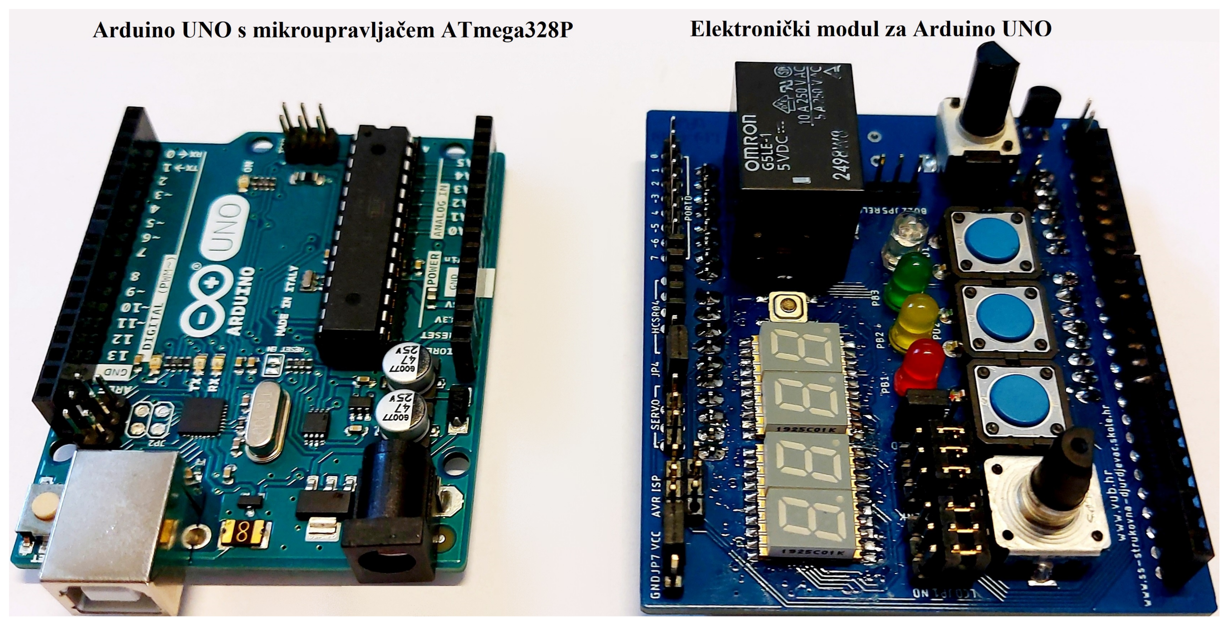


Slika 1.9: Strojni kod u datoteci `FirstProject.hex`

Poglavlje 2

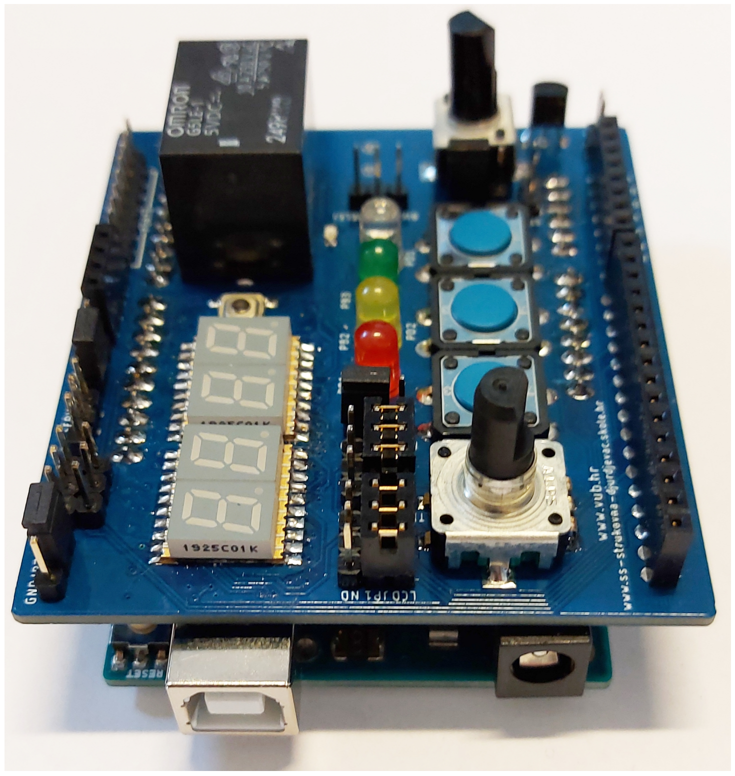
Razvojno okruženje s mikroupravljačem ATmega328P

U svrhu programiranja mikroupravljača ATmega328P koristit će se razvojno okruženje Arduino UNO (lijevo na slici 2.1) i razvijeni elektronički modul (desno na slici 2.1) za Arduino UNO razvojno okruženje. Razvojno okruženje Arduino UNO opremljeno je s mikroupravljačem ATmega328P. U ovom udžbeniku Arduino UNO razvojno okruženje nećemo koristi kao Arduino okruženje, već kao okruženje koje će nam omogućiti da programiramo mikroupravljač ATmega328P.

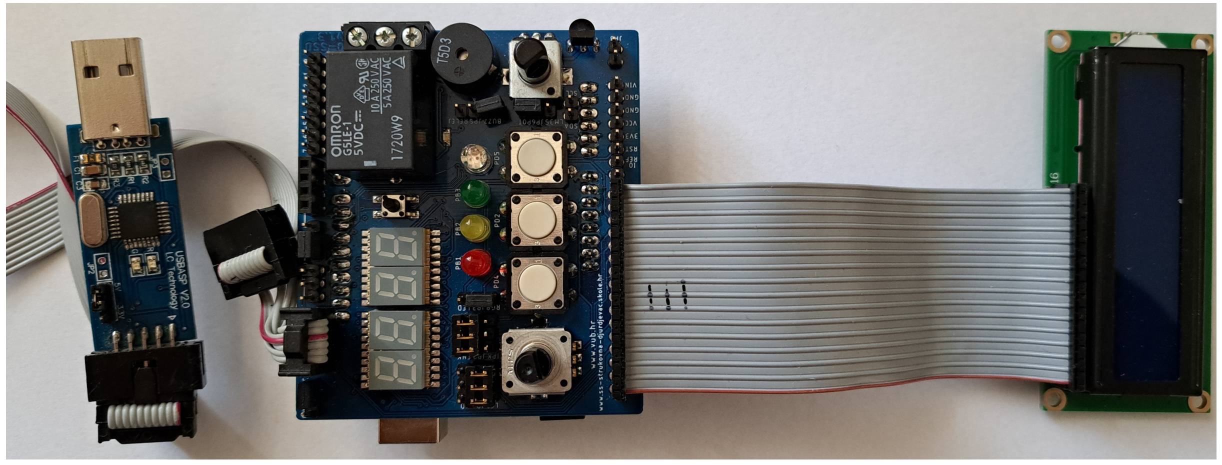


Slika 2.1: Razvojno okruženje Arduino UNO i razvijeni elektronički modul za Arduino UNO razvojno okruženje

Elektronički modul za Arduino UNO razvojno okruženje priključuje se na redne stezaljke razvojnog okruženja Arduino i na taj način ostvaruje povezanost raznih elektroničkih sklopova i komponenti na digitalne pinove mikroupravljača ATmega328P. Elektronički modul priključen na razvojno okruženje s mikroupravljačem ATmega328P prikazan je na slici 2.2.



Slika 2.2: Elektronički modul priključen na razvojno okruženje s mikroupravljačem ATmega328P



Slika 2.3: Elektronički modul za Arduino UNO razvojno okruženje

Razvijeni elektronički modul za Arduino UNO razvojno okruženje sa slike 2.3 opremljen je sljedećim elektroničkim komponentama i sklopovima:

- tri tipkala spojena na pinove PD2, PD4, i PD5 pomoću kratkospojnika JP2,
- tri LED diode (crvena, žuta i zelena) spojene na pinove PB1, PB2 i PB3 pomoću kratkospojnika JP3,
- RGB dioda spojena na pinove PB1, PB2 i PB3 pomoću kratkospojnika JP3,
- mehanički rotacijski enkoder s tipkalom spojen na pinove PD2, PD4, i PD5 pomoću

kratkospojnika JP2,

- potenciometar spojen na pin PC0 (ADC0) pomoći kratkospojnika JP6,
- temperaturni senzor LM35 spojen na pin PC0 (ADC0) pomoću kratkospojnika JP6,
- konektor za servo motor koji je spojen na pin PD3 pomoću kratkospojnika JP4,
- zujalica spojena na pin PD1 pomoću kratkospojnika JP5,
- relej spojen pomoću tranzistora kao sklopka na pin PD1 pomoću kratkospojnika JP5,
- četiri posmična registra spojena na pinove PC1, PC2 i PC3 pomoću kratkospojnika JP1,
- LCD displej spojena na pinove PC1, PC2 i PC3 pomoću kratkospojnika JP1 te izravno na pinove PB0, PB4 i PB6,
- RESET tipkalo spojeno na pin PC6,
- konektor za I2C komunikaciju spojen na pinove PC4 i PC5,
- konektor za ISP programiranje mikroupravljača ATmega328P spojen na pinove PB2, PB3, PB4 i PB5,
- serijska komunikacija putem USB priključka na Arduino UNO razvojnom okruženju, a spojena je na pinove PD0 i PD1.

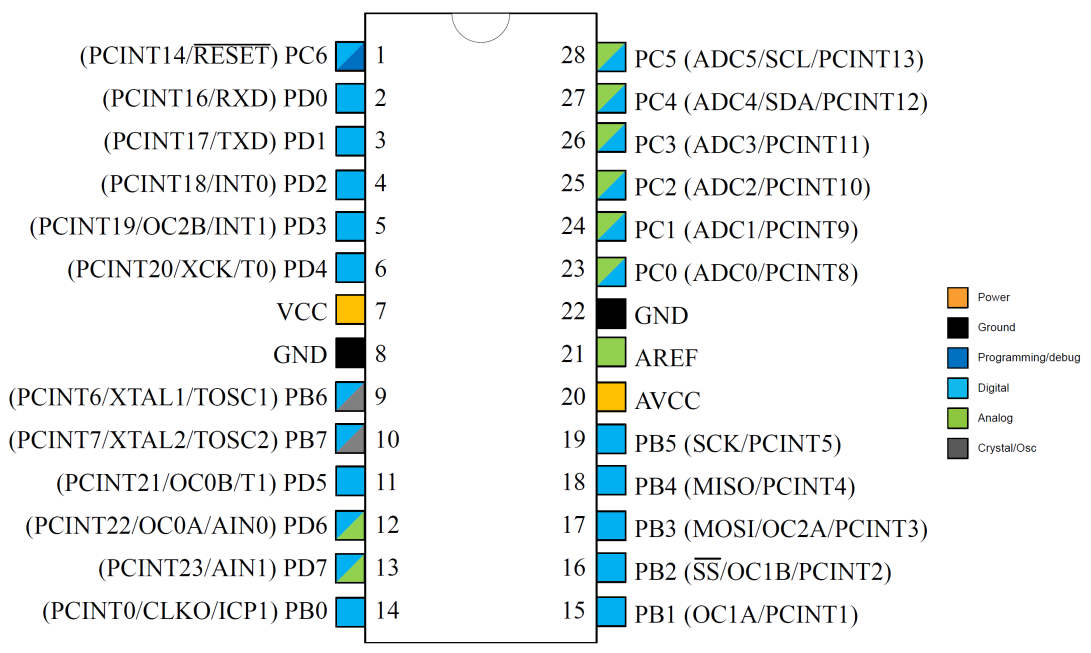
Na razvojnom okruženju Arduino UNO nalazi se mikroupravljač ATmega328P koji je jedan od često korištenih mikroupravljača opće namjene. Značajke mikroupravljača ATmega328P su [2]:

- visoke performanse, 8-bitni mikroupravljač male snage i AVR®porodice,
- RISC arhitektura:
  - 131 instrukcija,
  - 32 registra opće namjene,
  - do 20 MIPS (engl. *Million instructions per second*) na 20 MHz.
- 32 kB (engl. *In-System Self-programmable Flash*) programske memorije,
- 1 kB EEPROM podatkovne memorije,
- 2 kB SRAM podatkovne memorije,
- Ciklus pisanja/brisanja: *Flash* 10 000 puta/EEPROM 100 000 puta,
- dva 8-bitna tajmera *(Timers)*/brojača *(Counters)* s djeliteljima frekvencije,
- jedan 16-bitni tajmer *(Timer)*/brojač *(Counter)* s djeliteljima frekvencije,
- 6 PWM kanala,
- 6 kanala za analogno-digitalnu pretvorbu (razlučivost 10 bitova),

- *Master/Slave* SPI (engl. *Serial Peripheral Interface*) sučelja za komunikaciju,
- 2-*wire Serial Interface* sučelje za komunikaciju (I2C komunikacija),
- programirljivi USART (engl. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*),
- programirljivi tajmer za nadzor ispravnog rada (engl. *Watchdog Timer*),
- analogni komparator,
- *Power-on Reset*,
- programirljivi detektor pada napona napajanja (engl. *Brown Out Detection*),
- unutarnji i vanjski izvor prekida (engl. *Interrupts*),
- šest *Sleep* modova rada,
- interni oscilator (kalibrirani),
- 23 ulazno/izlazna pina koji su smješteni na tri porta,
- I/O podnožje: 28 pinova,
- nazivni napon: 1.8 - 5.5 V
- radni takt: 0 - 4 MHz na 1.8 - 5.5 V, 0 - 10 MHz na 2.7 - 5.5 V, 0 - 20 MHz na 4.5 - 5.5 V

Raspored pinova na mikroupravljaču ATmega328P prikazan je na slici 2.4. Mikroupravljač ATmega328P ima 23 ulazno/izlazna pina opće namjene (engl. *General-Purpose Input/Output* - GPIO) koji su smješteni na tri porta:

- **PORTB** (pinovi: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- **PORTC** (pinovi: PC0, PC1, PC2, PC3, PC4, PC5, PC6) i,
- **PORTD** (pinovi: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7).



Slika 2.4: Raspored pinova na mikroupravljaču ATmega328P [2]

Navedenih se 23 ulazno/izlazna pina mogu koristiti kao digitalni ulazi ili digitalni izlazi. Svi digitalni pinovi imaju svoju alternativnu namjenu. Kod spajanja elektroničkih komponenti ili sklopova na mikroupravljač važno je voditi računa o namjeni digitalnih pinova. Ukoliko imate potrebu za korištenje serijske komunikacije, tada ćete koristiti pinove PD0 (RXD) i PD1 (TXD). Analogno digitalna pretvorba može se provesti na pinovima PC0 - PC5 (ADC0 -ADC5). Ako u svom projektu koristite vanjske prekide (npr. za enkoder), tada ćete koristiti pinove PD2 (INT0) i PD3 (INT1). Vrlo korisna značajka mikroupravljača ATmega328P jest detekcija promjene signala na pinovima (engl. *Pin Change Interrupt*) koja je dostupna na svim digitalnim pinovima. Analogna komparacija dvaju analognih signala može se provesti na pinovima PD6 (AIN0) i PD7 (AIN1). Kada u projektu koristite I^2C komunikaciju, tada ćete uređaje koji koriste I^2C komunikaciju spojiti na pinove PC4 (SDA) i PC5 (SCL). Modulacija širine impulsa (PWM) može se ostvariti na ukupno šest pinova: PD6 (OC0A), PD5 (OC0B), PB1 (OC1A), PB2 (OC1B), PB3 (OC2A) i PD3 (OC2B). Primijetite da se na primjer pin PD3 može koristiti kao vanjski prekid ili kao pin za generiranje PWM signala. U tom slučaju potrebno je odlučiti da li ćete ovaj pin koristiti kao vanjski prekid ili kao pin za generiranje PWM, jer obje namjene pina u isto vrijeme nisu moguće. Pri donošenju odluke na koji ćete digitalni pin spojiti korištene elektroničke komponente i sklopove u projektu uvijek se prvo rezerviraju namjenski pinovi (serijska komunikacija, analogno digitalna pretvorba, PWM, ...), a nakon toga na preostale pinove možete spojiti komponente koje ne zahtijevaju namjenske pinove (LED diode, tipkala, LCD displej, ...).

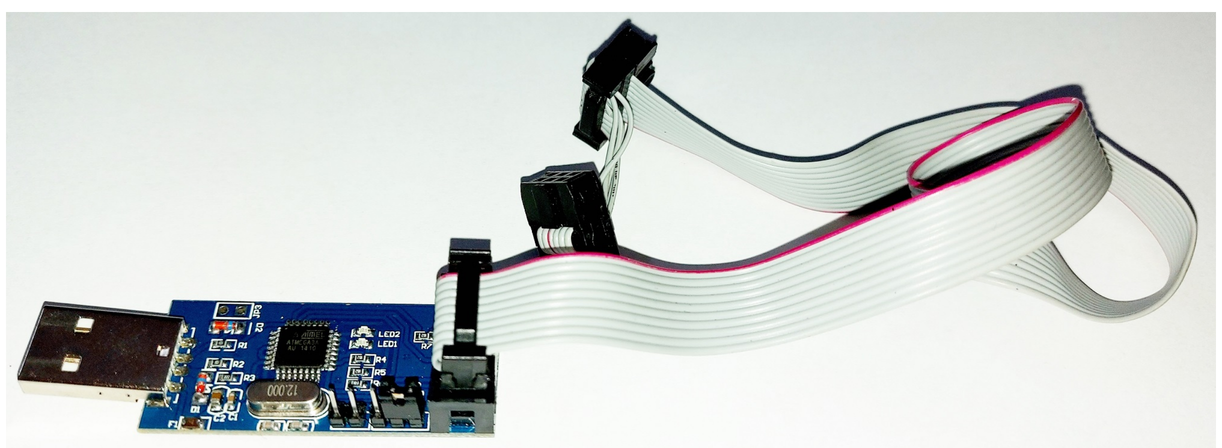
Poglavlje 3

Programiranje mikroupravljača ATmega328P

3.1 ISP programiranje mikroupravljača ATmega328P

Strojni kod koji je generiran prevođenjem programskog koda u programskom razvojnom okruženju *Microchip Studio* potrebno je "snimiti" na mikroupravljač ATmega328P. Postupak koji ćemo koristiti za programiranje mikroupravljača ATmega328P naziva se *In-System Programming* (ISP). Specifično za ovaj postupak programiranja jest taj što se mikroupravljač programira na razvojnom okruženju.

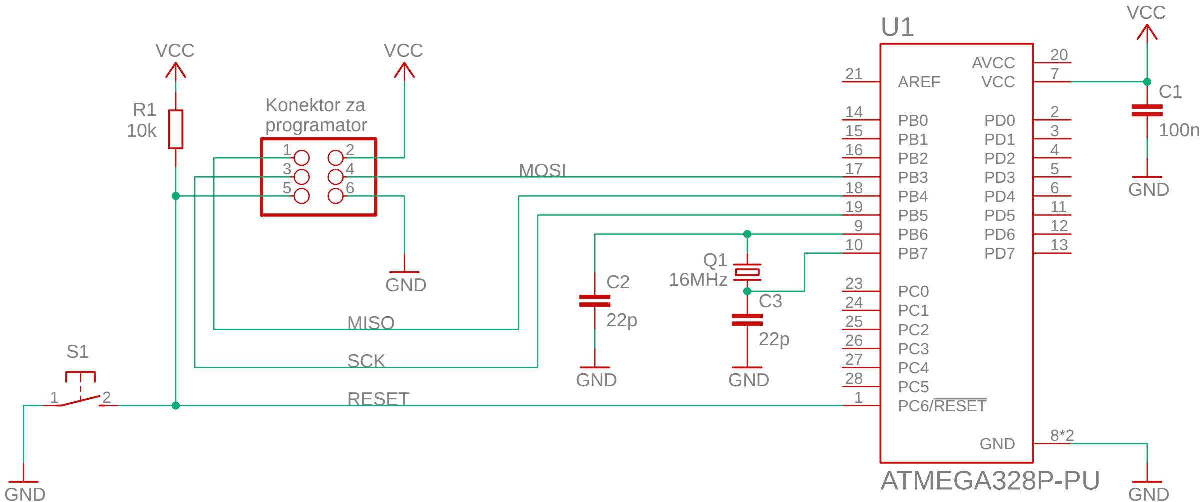
ISP programiranje mikroupravljača ATmega328P provodit ćemo pomoću programatora USBasp koji je prikazan na slici 3.1. Ovaj programator ima vrlo nisku cijenu, a omogućuje programiranje AVR mikroupravljača putem USB sučelja. USBasp programator potrebno je spojiti na ISP konektor elektroničkog modula za Arduino UNO razvojno okruženje (vidi sliku 2.3).



Slika 3.1: Programator USBasp za ISP programiranje mikroupravljača ATmega328P

Shema spajanja ISP programatora s mikroupravljačem ATmega328P prikazana je na slici 3.2. Pri programiranju mikroupravljača ATmega328P, ISP programator koristi RESET pin te SPI[1] komunikaciju koja koristi pinove SCK (PB5), MISO (PB4) i MOSI (PB3). Radni takt mikroupravljača iznosi 16 MHz, a generira se vanjskim oscilatorom (kristal kvarca) koji je na slici 3.2 prikazan oznakom Q1.

[1] *Serial Peripheral Interface.*



Slika 3.2: Shema spajanja USBasp programatora s mikroupravljačem ATmega328P

Uz programator USBasp, za programiranje mikroupravljača ATmega328P, koristit ćemo softver AVRDUDESS. Ovaj softver koristi se za programiranje AVR porodice mikroupravljača. Zadnju verziju softvera *AVRDUDESS* možete preuzeti na stranici `https://avrdudess.software.informer.com/`.

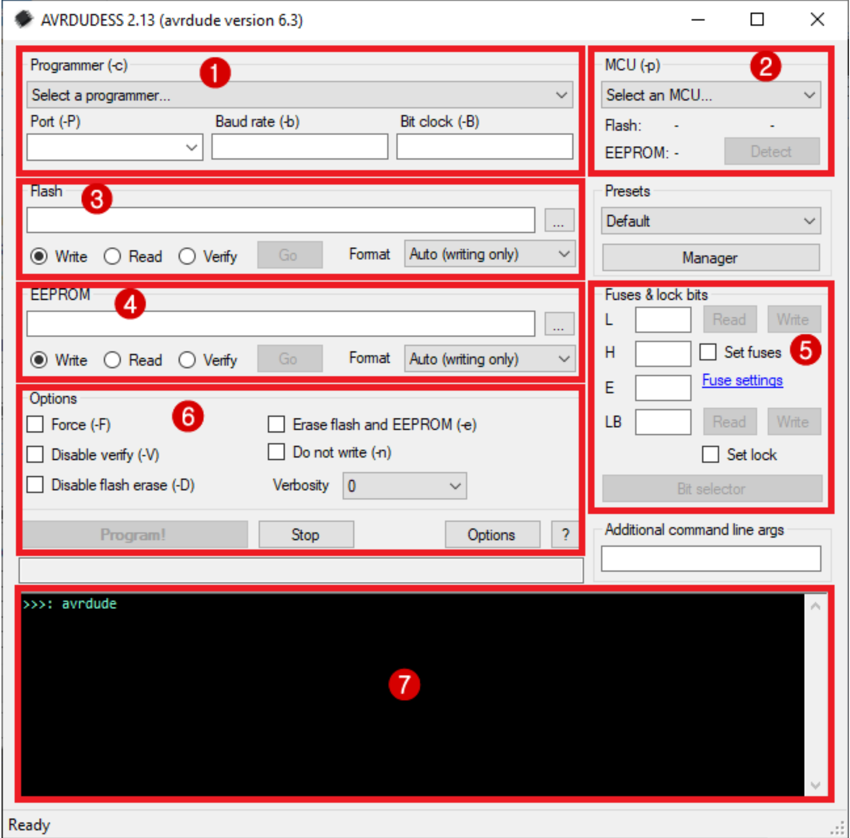
Softver *AVRDUDESS* pokreće se dvostrukim klikom na ikonu sa slike 3.3. Pokretanjem softvera *AVRDUDESS* pojavit će se prozor sa slike 3.4.

Slika 3.3: Softver *AVRDUDESS* - ikona

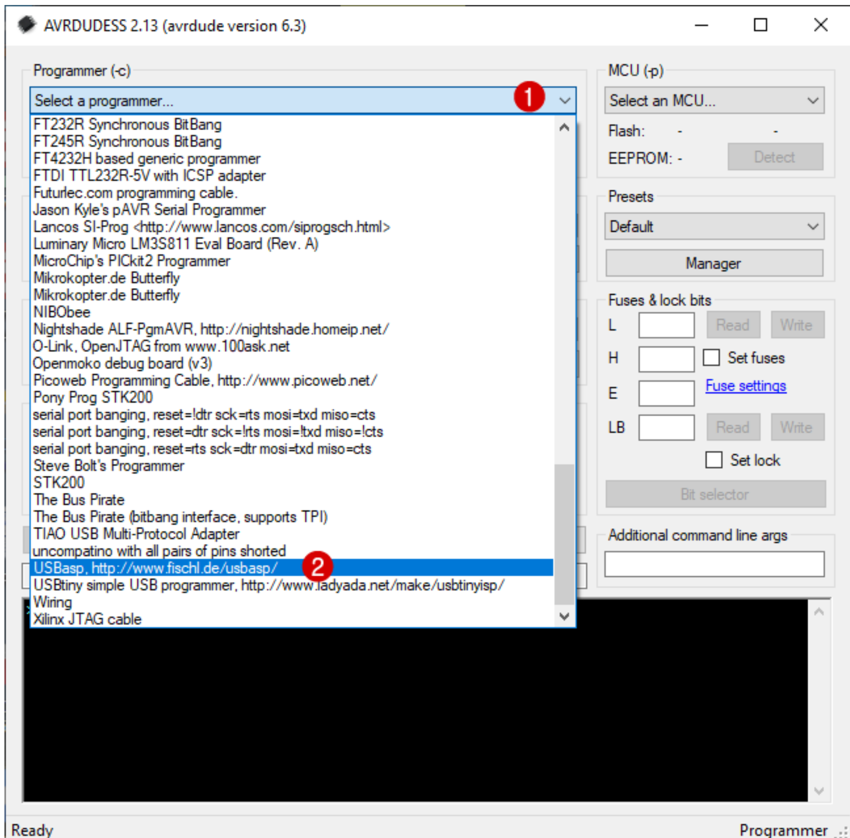
Na slici 3.4 označeno je sedam ključnih segmenata softvera *AVRDUDESS*:

1. odabir i konfiguriranje programatora,
2. odabir mikroupravljača koji će se programirati,
3. učitavanje strojnog koda i programiranje programske memorije mikroupravljača,
4. učitavanje sadržaja EEPROM memorije i programiranje EEPROM memorije mikroupravljača,
5. čitanje i pisanje *Fuse* i *Lock* bitova,
6. opcije programiranja mikroupravljača,
7. pokaznih statusnih poruka.

Prvo što je potrebno napraviti pri pokretanju softvera *AVRDUDESS* jest odabrati programator koji koristite za programiranje AVR mikroupravljača. U segmentu označenom brojem 1 na slici 3.4 u padajućem izborniku potrebno je odabrati programator USBasp (vidi sliku 3.5). Nakon odabira programatora, potrebno je odabrati mikroupravljač na koji ćemo snimati strojni kod. Postupak odabira mikroupravljača moguć je na dva načina.

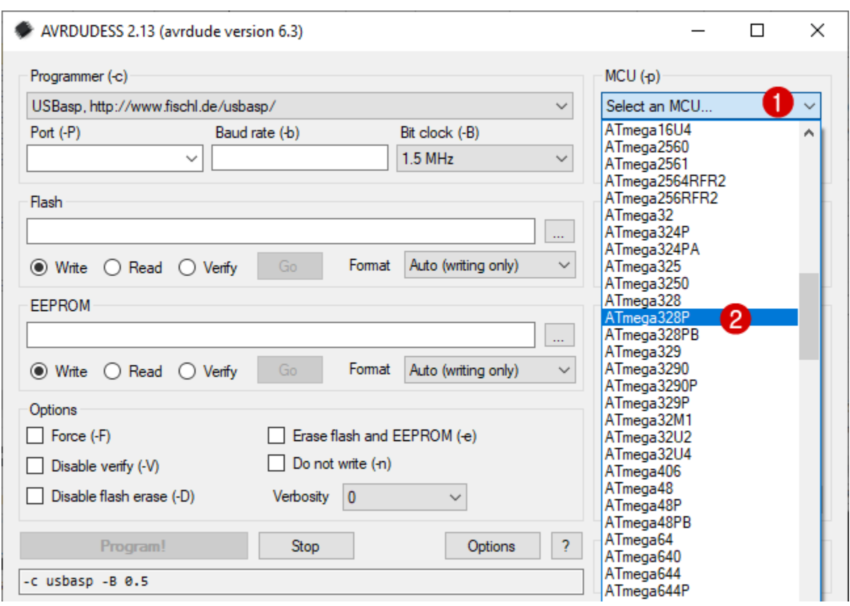


Slika 3.4: Softver *AVRDUDESS* - početni prozor

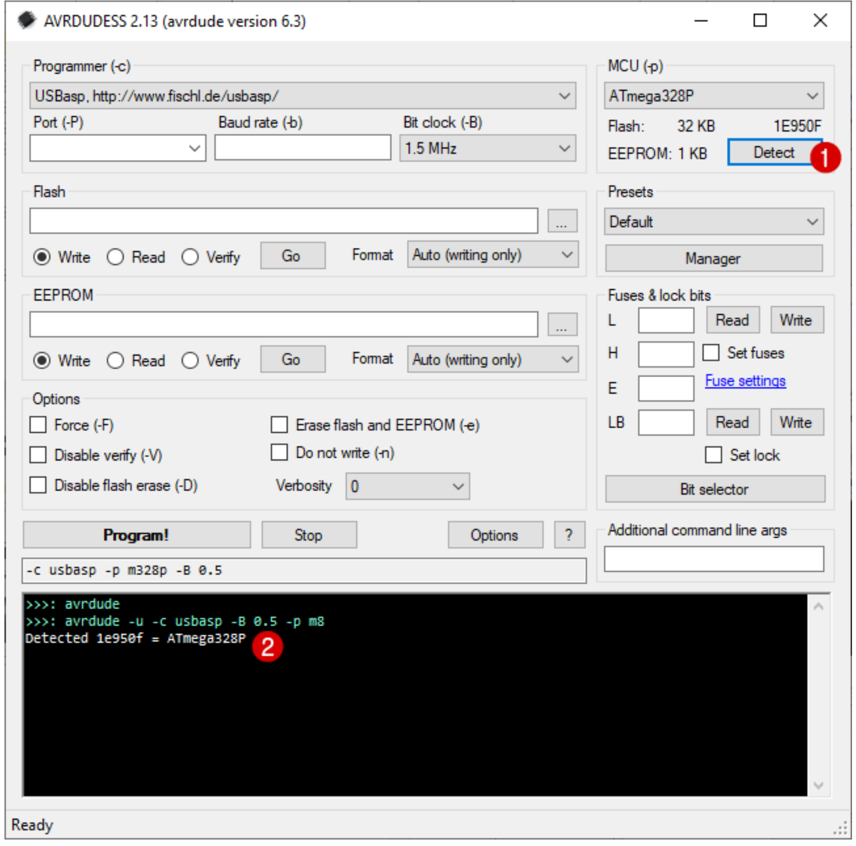


Slika 3.5: Softver *AVRDUDESS* - odabir programatora

Prvi način jest da u segmentu označenom brojem 2 na slici 3.4 u padajućem izborniku pronađemo mikroupravljač ATmega328P (vidi sliku 3.6). Drugi način jest da u segmentu označenom brojem 2 na slici 3.4 pritisnemo tipku *Detect* (broj 1 na slici 3.7). Uz preduvjet da ste USBasp programator priključili na USB port računala, softver *AVRDUDESS* će prepoznati mikroupravljač koji je spojen s programatorom USBasp. U pokazniku statusnih poruka bit će prikazan potpis i ime detektiranog mikorupravljača (broj 2 na slici 3.7).

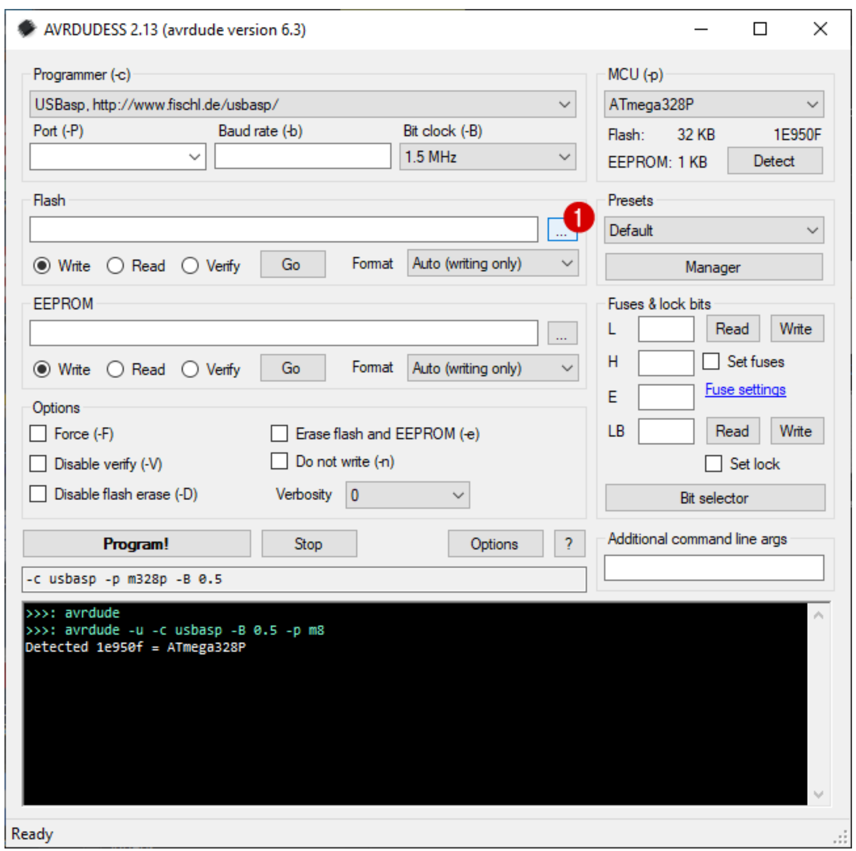


Slika 3.6: Softver *AVRDUDESS* - odabir mikroupravljača (1. način)

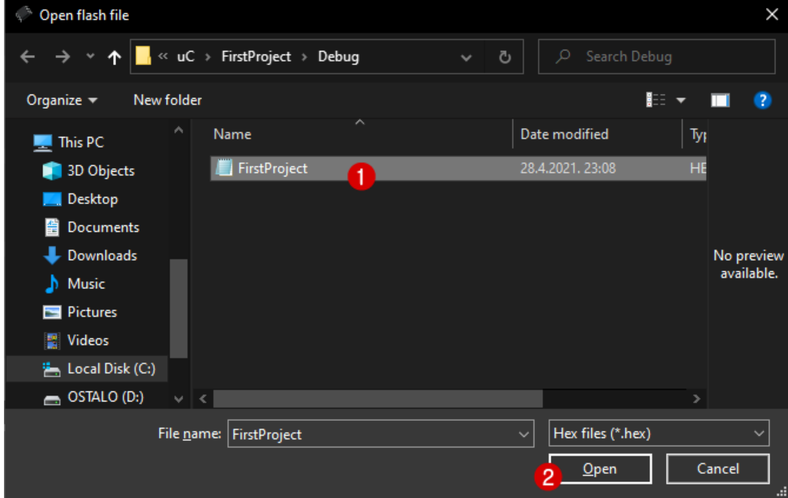


Slika 3.7: Softver *AVRDUDESS* - odabir mikroupravljača (2. način)

Programirati mikroupravljač znači snimiti strojni kod koji smo stvorili pomoću razvojnog programskog okruženja *Mirochip Studio* na mikroupravljač pomoću programatora. Da bismo snimili strojni kod na mikroupravljač, isti je potrebno učitati u softver *AVRDUDESS*. To možemo učiniti tako da u segmentu označenom brojem 3 na slici 3.4 pritisnemo tipku ... (broj 1 na slici 3.8). Nakon toga je u prozoru sa slike 3.9 potrebno odabrati lokaciju `C:\uC\FirstProject\Debug`. Na ovoj lokaciji nalazi se strojni kod u datoteci `FirstProject.hex` (broj 1 na slici 3.9) koju je potrebno odabrati i učitati u softver *AVRDUDESS* (broj 2 na slici 3.9).



Slika 3.8: Softver *AVRDUDESS* - učitavanje strojnog koda u softver

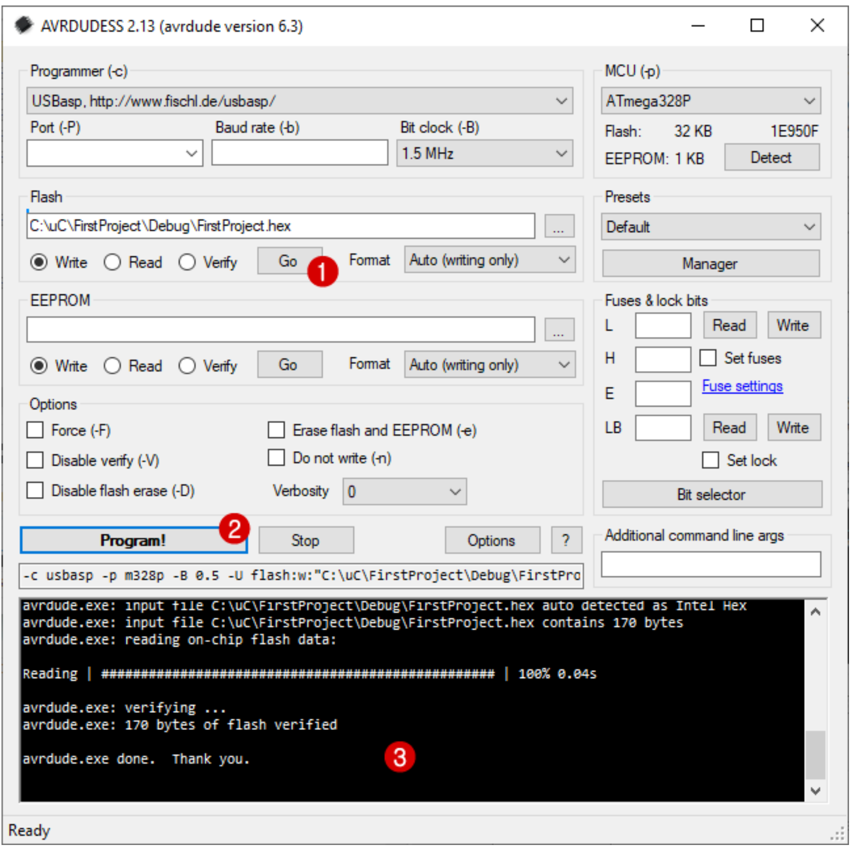


Slika 3.9: Softver *AVRDUDESS* - odabir lokacije strojnog koda

Softver *AVRDUDESS* u segmentu označenom brojem 3 na slici 3.4 omogućuje:

- zapisivanje strojnog koda na mikroupravljač (opcija *Write*),
- čitanje strojnog koda s mikroupravljača (opcija *Read*),
- provjeru zapisanog strojnog koda na mikroupravljač (opcija *Verify*).

Kako bismo snimili strojni kod na mikroupravljač, u segmentu označenom brojem 3 na slici 3.4 potrebno je odabrati opciju *Write* i zatim pritisnuti tipku *Go* (broj 1 na slici 3.10). Program se na mikroupravljač može snimiti tako da se u segmentu označenom brojem 6 na slici 3.4 pritisne tipka *Program* (broj 2 na slici 3.10). Uspješnost zapisivanje, čitanja i provjere strojnog koda mikroupravljača prikazuje se u pokazniku statusnih poruka (broj 3 na slici 3.10).



Slika 3.10: Softver *AVRDUDESS* - odabir lokacije strojnog koda

Ako prvi put programiramo mikroupravljač ili mu se mijenjaju osnovne postavke, potrebno je namjestiti tzv. *Fuse* bitove (konfiguracijske bitove). Ti bitovi konfiguriraju rad mikroupravljača, odnosno namještaju postavke koje se obično ne mijenjaju tijekom izvođenja programskog koda. *Fuse* bitovima može se mijenjati radni takt mikroupravljača, izvor radnog takta (unutarnji ili vanjski), način programiranja, detekcija propada napona napajanja i drugo. Tvorničkom konfiguracijom *Fuse* bitova mikroupravljača ATmega328P definirane su sljedeće postavke:

- izvor radnog takta unutarnji je oscilator frekvencije 8 MHz sa *Start-up* vremenom 6 CK + 65 ms,
- radni takt interno je podijeljen s 8,
- omogućeno je programiranje mikroupravljača pomoću SPI komunikacije (ISP

programiranje),

- detekcija propada napona napajanja (engl. *Brown-out detection*) je isključena,
- *Boot Flash* veličine je 2048 memorijskih riječi, odnosno 4 kB.

Ovim postavkama odgovaraju sljedeće vrijednosti *Fuse* bitova u heksadecimalnom zapisu:

- niži *Fuse* bitovi (engl. *Low Fuse*) jesu 0x62,
- viši *Fuse* bitovi (engl. *High Fuse*) jesu 0xD9,
- prošireni *Fuse* bitovi (engl. *Extended Fuse*) jesu 0xFF.

Fuse bitovi mikroupravljača AVR porodice mogu se izračunati korištenjem *Fuse* bit kalkulatora koji je dostupan na stranici `www.engbedded.com/fusecalc/`. Na mikroupravljač ATmega328P ćemo za potrebe vježbi u ovom udžbeniku snimiti sljedeće *Fuse* bitove:

- niži *Fuse* bitovi (engl. *Low Fuse*) jesu 0xFF,
- viši *Fuse* bitovi (engl. *High Fuse*) jesu 0xDF,
- prošireni *Fuse* bitovi (engl. *Extended Fuse*) jesu 0xFF.

Navedeni *Fuse* bitovi odgovaraju sljedećim postavkama mikroupravljača ATmega328P:

- izvor radnog takta vanjski je oscilator frekvencije veće i jednake od 8 MHz sa *Start-up* vremenom 16k CK + 65 ms,
- omogućeno je programiranje putem SPI sučelja,
- *Boot Flash* veličine je 256 memorijskih riječi (0.5 kB).

Nakon što smo pomoću kalkulatora izračunali *Fuse* bitove, potrebno ih je snimiti na mikroupravljač ATmega328P. *Fuse* bitove ćemo snimiti na mikroupravljač tako da u segmentu označenom brojem 5 na slici 3.4 provedemo sljedeće korake (vidi sliku 3.11):

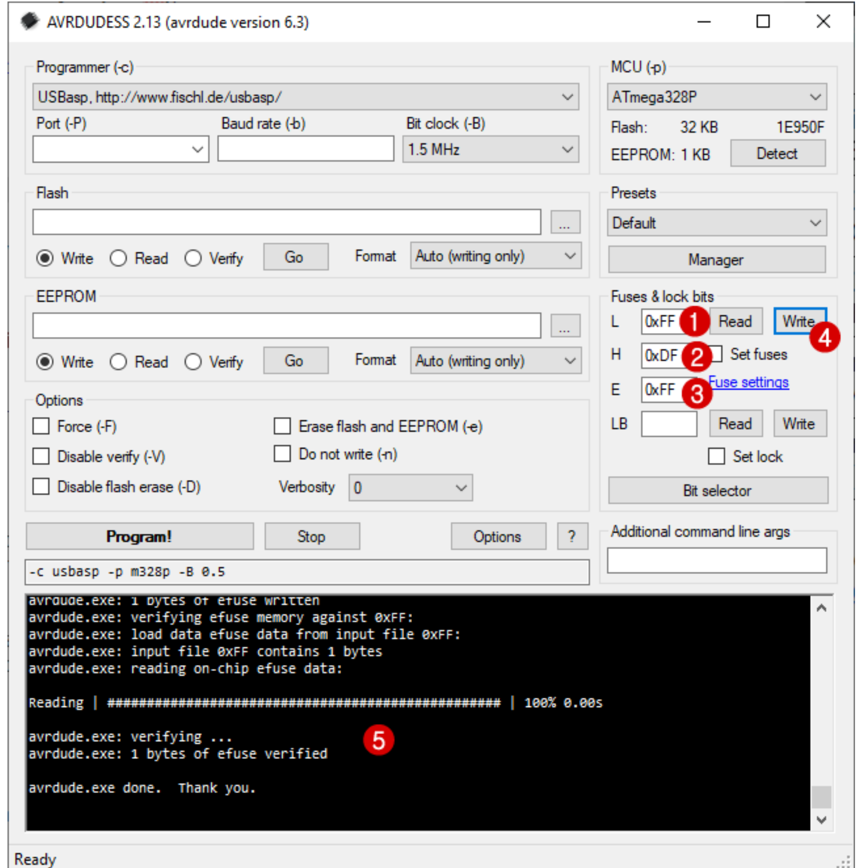
1. u polje `L` (broj 1 na slici 3.11) upišite 0xFF (*Low Fuse*),

2. u polje `H` (broj 2 na slici 3.11) upišite 0xDF (*High Fuse*),

3. u polje `E` (broj 3 na slici 3.11) upišite 0xFF (*Extended Fuse*),

4. pritisnite tipku *Write* (broj 4 na slici 3.11) .

Rezultat snimanja *Fuse* bitova na mikroupravljač prikazan je u pokazniku statusnik poruka (broj 5 na slici 3.11). *Fuse* bitovi mogu se i pročitati iz mikroupravljača ukoliko u segmentu označenom brojem 5 na slici 3.4 pritisnete tipku *Read*. Vrijednosti *Fuse* bitova prikazat će se u poljima označenima brojevima 1, 2 i 3 na slici 3.11. *Fuse* bitovi se u pravilu na mikroupravljač snimaju samo jednom i to pri prvom korištenju mikroupravljača.



Slika 3.11: Softver *AVRDUDESS* - snimanje *Fuse* bitova

Segment označen brojem 4 na slici 3.4 omogućuje zapisivanje, čitanje i provjeru EEPROM memorije. Postupak svih radnji isti je kao i za programiranje programske memorije. EEPROM memorija koristi se za snimanje podataka koji moraju biti dostupni i nakon gubitka napajanja mikroupravljača.

U segmentu označenom brojem 6 na slici 3.4 navedene su opcije kojima se konfigurira napredne opcije programiranja mikroupravljača:

- *Force (-F)* - opcija koja snima strojni kod na mikroupravljač iako potpis (engl. *Signature*) mikroupravljača nije ispravan. Ova opcija koristi se ako se iz nekog razloga izbrisao potpis mikroupravljača, a sigurni smo koji mikroupravljač programiramo.

- *Erase flash and EEPROM (-e)* - opcija koja pri programiranju mikroupravljača briše sadržaj programske memorije i EEPROM memorije.

- *Disable verify (-V)* - opcija koja isključuje provjeru strojnog koda koji smo snimili na mikroupravljač.

- *Do not write (-n)* - opcija koja onemogućuje snimanje svih podataka na mikroupravljač.

- *Disable flash erase (-D) - opcija koja isključuje automatsko brisanje programske memorije pri programiranju mikroupravljača.*

- *Verbosity* - padajući izbornik koji omogućuje ispis detalja u pokazniku statusnih poruka pri programiranju (raznina detalja je od 0 do 4).

Pokaznik statusnih poruka softvera *AVRDUDESS* spomenuli smo već nekoliko, a prikazan je

u segmentu označenom brojem 7 na slici 3.4. Kako smo već naveli, pokaznik statusnih poruka služi informiranju korisnika o aktivnostima provedenima pomoću softvera *AVRDUDESS*.

Kada na mikroupravljač snimite strojni kod koji je nastao prevođenjem programskog koda u programskom razvojnom okruženju *Microchip Studio*, potrebno je testirati rad mikroupravljača. Testiranje provodite tako da na razvojnom okruženju kojeg koristite provjerite funkcionalnosti koje ste definirali projektom. Na primjer: provjerite da li će se uključiti crvena LED dioda spojena na pin PB1 ako ste pritisnuli tipkalo koje je spojeno na pin PD2. Ako testiranjem utvrdite propuste u funkcionalnosti rada razvojnog okruženja, potrebno je izmijeniti programsko kod, prevesti ga u strojni kod i ponovno provesti postupak programiranja mikroupravljača. Ovi koraci se provode sve dok funkcionalnost razvojnog okruženja nije jednaka definiranoj funkcionalnosti.

Snimljeni strojni kod se može pročitati s mikroupravljača. Taj strojni kod može se snimiti na drugi mikroupravljač istog kataloškog broja, no ne može se prevesti u početni programski kod.

U slučaju da na razvojnom okruženju imate mikroupravljač porodice AVR koji je različit od mikroupravljača ATmega328P, postupak programiranja mikroupravljača softverom *AVRDUDESS* isti je i za ostale mikroupravljače AVR porodice. Pri programiranju mikroupravljača važno je u segmentu označenom brojem 2 na slici 3.4 izabrati ili detektirati ispravni mikroupravljač na razvojnom okruženju.

Prilikom učitavanja strojnog koda u softver *AVRDUDESS* vodite računa da nijedna datoteka na putanji prema `*.hex` datoteci nema dijakritičkih znakova. U suprotnom softver *AVRDUDESS* neće učitati strojni kod.

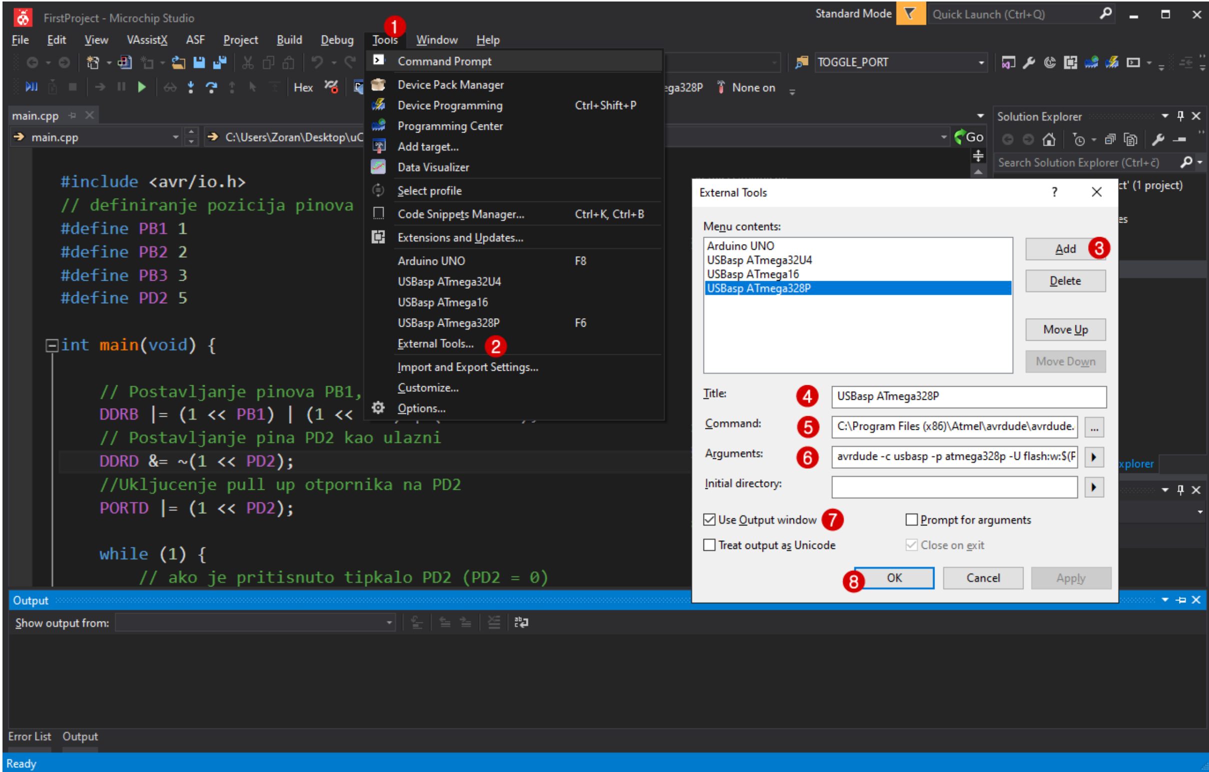
Snimanje strojnog koda na mikroupravljač porodice *AVR* moguće je i direktno iz programskog razvojnog okruženja *Microchip Studio*. U tom slučaju potrebno je u programsko razvojno okruženje *Microchip Studio* dodati vanjski programator USBasp s pripadajućim opcijama. Postupak dodavanja USBasp programatora za mikroupravljač ATmega328P kao vanjskog uređaja u programsko razvojno okruženje *Microchip Studio* prikazan je na slici 3.12.

Dodavanje USBasp programatora u programsko razvojno okruženje *Microchip Studio* za mikroupravljač ATmega328P može se provesti kroz korake prikazane na slici 3.12 [1]:

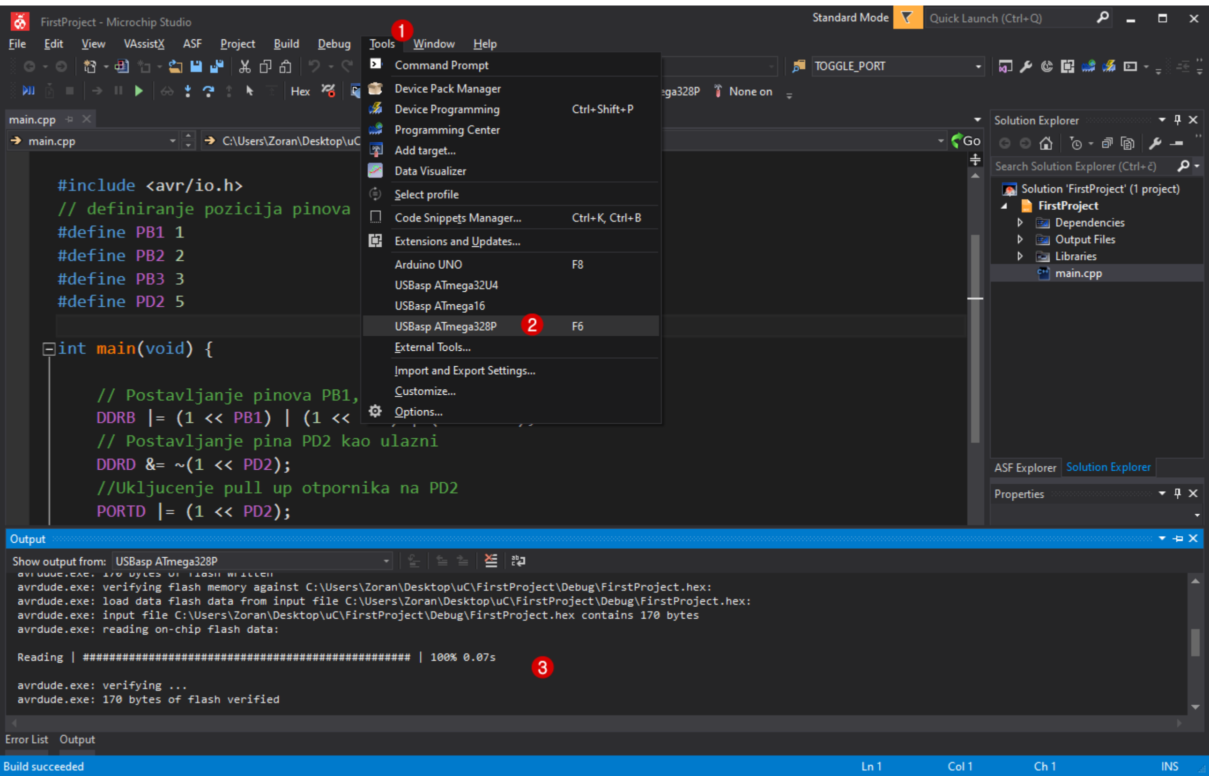
1. u programskom razvojnom okruženju *Microchip Studio* odaberite izbornik `Tools`,
2. u izborniku `Tools` odaberite `External Tools...`,
3. u novootvorenom prozoru pritisnite gumb `Add`,
4. u polju `Title` upišite: USBasp ATmega328P,
5. u polje `Command` upišite: C:\Program Files (x86)\Atmel\avrdude\avrdude.exe[2]
6. u polje `Arguments` upišite:
   avrdude -c usbasp -p atmega328P -U flash:w:$(ProjectDir)Debug\$(TargetName).hex:i
7. označite opciju `Use Output window`,
8. završite parametriranje pritiskom na gumb `OK`.

Nakon provedenog postupka parametriranja programatora USBasp, strojni kod za mikroupravljač ATmega328P možete snimiti na mikroupravljač ATmega328P odabirom `Tools -> USBasp ATmega328P` (slika 3.13 - brojevi 1 i 2). Uspješnost programiranja možete pratiti u pokazniku statusnih poruka (slika 3.13 - broj 3).

[2]Potrebno je skinuti *Avrdude* sa stranice `https://www.nongnu.org/avrdude/`.



Slika 3.12: Programiranje mikroupravljača ATmega328P iz programskog razvojnog okruženja *Microchip Studio* - parametriranje programatora USBasp



Slika 3.13: Programiranje mikroupravljača iz programskog razvojnog okruženja *Microchip Studio* pomoću programatora USBasp

Zanimljivo je vidjeti kako su statusne poruke jednake bez obzira korisite li softver *AVRDUDESS* za programiranje mikroupravljaća ATmega328P ili programsko razvojno okruženja *Microchip Studio*. Razlog tome jest taj što se za programiranje mikroupravljača u oba slučaja poziva softver AVRDUDE koji omogućuje programiranje mikroupravljača porodice AVR.

Fuse bitove mikroupravljača i dalje je potrebno snimiti pomoću softvera *AVRDUDESS*, no kada ih jednom podesite programirati možete pomoću programskog razvojnog okruženja *Microchip Studio*, što će vam omogućiti rad u samo jednom softveru.

Strojni kod koji smo snimili u programsk memoriju mikroupravljača izvodi se onog trenutka kada na mikroupravljač dovedemo napajanje uz uvjet da je stanje RESET pina visoko. Ako mikroupravljač izgubi napajanje ili na pin RESET dovedemo nisko stanje, mikroupravljač zaustavlja izvođenje programskog koda koji se nalazi zapisan u programsku memoriju. Vraćanjem stanja RESET pina u visoko stanje, programski kod mikroupravljača izvodi se ispočetka.

3.2 Programiranje mikroupravljača ATmega328P pomoću Arduino *Bootloader* programa

Jedna od prikladnih metoda programiranja mikroupravljača jest pomoću *Bootloader* programa. Kod ove metode nije potrebno koristiti programator, već se može koristiti postojeća serijska komunikacija (USART) na mikroupravljaču. *Bootloader* program je program mikroupravljača koji je zapisan na kraju programske memorije, a služi za programiranje mikroupravljača. Prilikom uključenja mikroupravljača ili ako na RESET pin dovedemo nisko stanje, mikroupravljač izvodi *Bootloader* program. Ako istovremeno serijskom komunikacijom s računala šaljemo strojni kod, *Bootloader* program će ga smjestiti u programsku memoriju.

Bootloader program se na mikroupravljač snima pomoću USBasp programatora. S obzirom na široku upotrebu Arduino razvojnih okruženja i s obzirom da naše razvojno okruženje koristi Arduino UNO razvojno okruženje koristit ćemo Arduino *Bootloader* program. Arduino UNO razvojno okruženje na sebi već ima Arduino *Bootloader* program. No, ukoliko ste mikroupravljač ATmega328P programirali pomoću programatora USBasp, tada ste Arduino *Bootloader* program izbrisali te ga je potrebno vratiti nazad. Za programiranje mikroupravljača ATmega328P na Arduino UNO razvojnom okruženju koristit ćemo USB priključak koji se nalazi na Arduino UNO razvojnom okruženju.

U nastavku ćemo ukratko opisati snimanje Arduino *Bootloader* programa na mikroupravljač ATmega328P. Za Arduino *Bootloader* program potrebno je koristiti sljedeće *Fuse* bitove:

1. u polje `L` (broj 1 na slici 3.11) upišite 0xFF (*Low Fuse*),
2. u polje `H` (broj 2 na slici 3.11) upišite 0xDE (*High Fuse*),
3. u polje `E` (broj 3 na slici 3.11) upišite 0xFD (*Extended Fuse*),
4. pritisnite tipku *Write* (broj 4 na slici 3.11) .

Navedeni *Fuse* bitovi odgovaraju sljedećim postavkama mikroupravljača ATmega328P:

- izvor radnog takta vanjski je oscilator frekvencije veće ili jednake od 8 MHz sa *Start-up* vremenom 16k CK + 65 ms,
- omogućeno je programiranje putem SPI sučelja,
- *Boot Flash* veličine je 256 memorijskih riječi (0.5 kB),

- *Boot Reset* vektor je omogućen,
- detekcija propada napona napajanja (engl. *Brown-out detection*) postavljena je na 2.7 V.

Fuse bitove za Arduino *Bootloader* program potrebno je snimiti na mikroupravljač ATmega328P prema uputama iz prethodnog potpoglavlja. *Fuse* bit *Boot Reset* omogućuje mikroupravljaču da pozove Arduino *Bootloader* program pri uključenju ili resetiranju.

Arduino *Bootloader* program prikazan je strojnim kodom 3.1.

Programski kod 3.1: Strojni kod Arduino *Bootloader* programa

```
:107E0000112484B714BE81FFF0D085E080938100F7
:107E100082E08093C00088E18093C10086E0809377
:107E2000C20080E18093C4008EE0C9D0259A86E02C
:107E300020E33CEF91E0309385002093840096BBD3
:107E4000B09BFECF1D9AA8958150A9F7CC24DD24C4
:107E500088248394B5E0AB2EA1E19A2EF3E0BF2EE7
:107E6000A2D0813461F49FD0082FAFD0023811F036
:107E7000013811F484E001C083E08DD089C08234E0
:107E800011F484E103C0853419F485E0A6D080C0E4
:107E9000853579F488D0E82EFF2485D0082F10E0AE
:107EA000102F00270E291F29000F111F8ED06801E7
:107EB0006FC0863521F484E090D080E0DECF843638
:107EC00009F040C070D06FD0082F6DD080E0C81688
:107ED00080E7D80618F4F601B7BEE895C0E0D1E017
:107EE00062D089930C17E1F7F0E0CF16F0E7DF06D8
:107EF00018F0F601B7BEE89568D007B600FCFDCFD4
:107F0000A601A0E0B1E02C9130E011968C91119780
:107F100090E0982F8827822B932B1296FA010C0160
:107F200087BEE89511244E5F5F4FF1E0A038BF0790
:107F300051F7F601A7BEE89507B600FCFDCF97BE46
:107F4000E89526C08437B1F42ED02DD0F82E2BD052
:107F50003CD0F601EF2C8F010F5F1F4F84911BD097
:107F6000EA94F801C1F70894C11CD11CFA94CF0C13
:107F7000D11C0EC0853739F428D08EE10CD085E9AC
:107F80000AD08FE07ACF813511F488E018D01DD067
:107F900080E101D065CF982F8091C00085FFFCCF94
:107FA0009093C60008958091C00087FFFCCF809118
:107FB000C00084FD01C0A8958091C6000895E0E648
:107FC000F0E098E1908380830895EDDF803219F02E
:107FD00088E0F5DFFFCF84E1DECF1F93182FE3DFCA
:107FE0001150E9F7F2DF1F91089580E0E8DFEE27F6
:047FF000FF270994CA
:027FFE00040479
:0400000300007E007B
:00000001FF
```

Strojni kod 3.1 potrebno je kopirati i snimiti u datoteku koju nazovite `Bootloader.hex`[3] (vidi sliku 3.14). Prema uputama iz prethodnog potpoglavlja, strojni kod iz datoteke `Bootloader.hex` potrebno je snimiti na mikroupravljač ATmega328P pomoću softvera *AVRDUDESS*.

Snimanje strojnog koda na mikroupravljač ATmega328P pomoću Arduino *Bootloader* programa moguće je i direktno iz programskog razvojnog okruženja *Microchip Studio*. U tom slučaju potrebno je u programsko razvojno okruženje *Microchip Studio* dodati mogućnost programiranja pomoću Arduino *Bootloader* programa.

[3] Stvorite Notepad datoteku i promijenite joj ekstenziju iz `*.txt` u `*.hex`

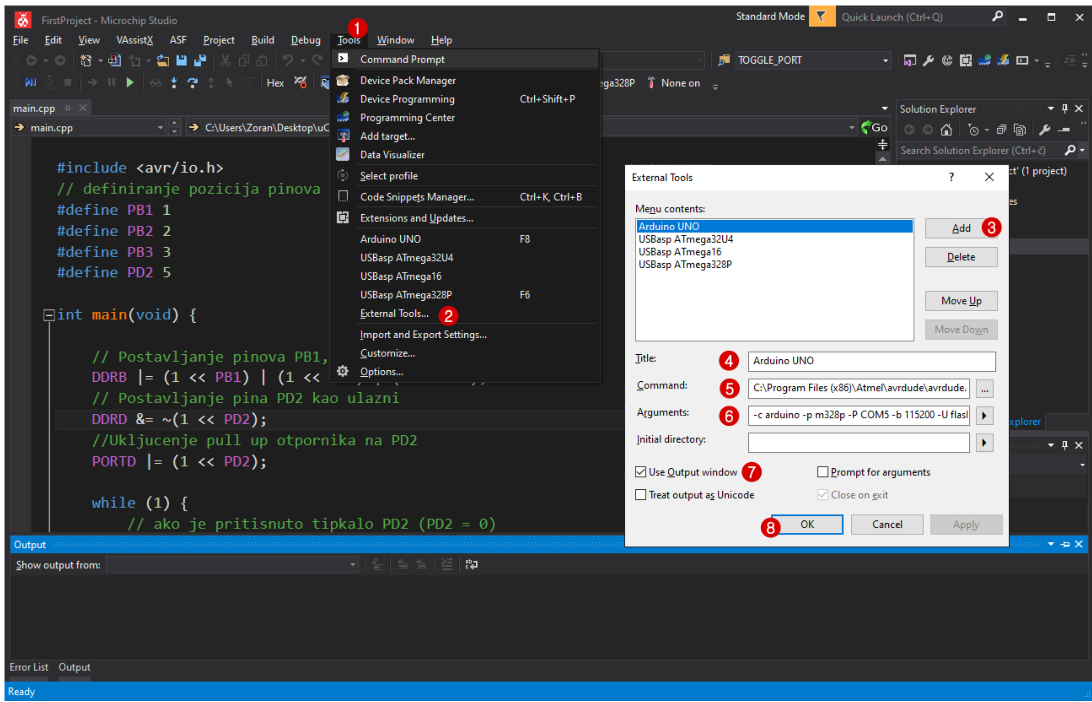
Slika 3.14: Strojni kod Arduino *Bootloader* programa u datoteci `Bootloader.hex`

Dodavanje mogućnosti programiranja pomoću Arduino *Bootloader* programa u programsko razvojno okruženje *Microchip Studio* za mikroupravljač ATmega328P može se provesti kroz korake prikazane na slici 3.15:

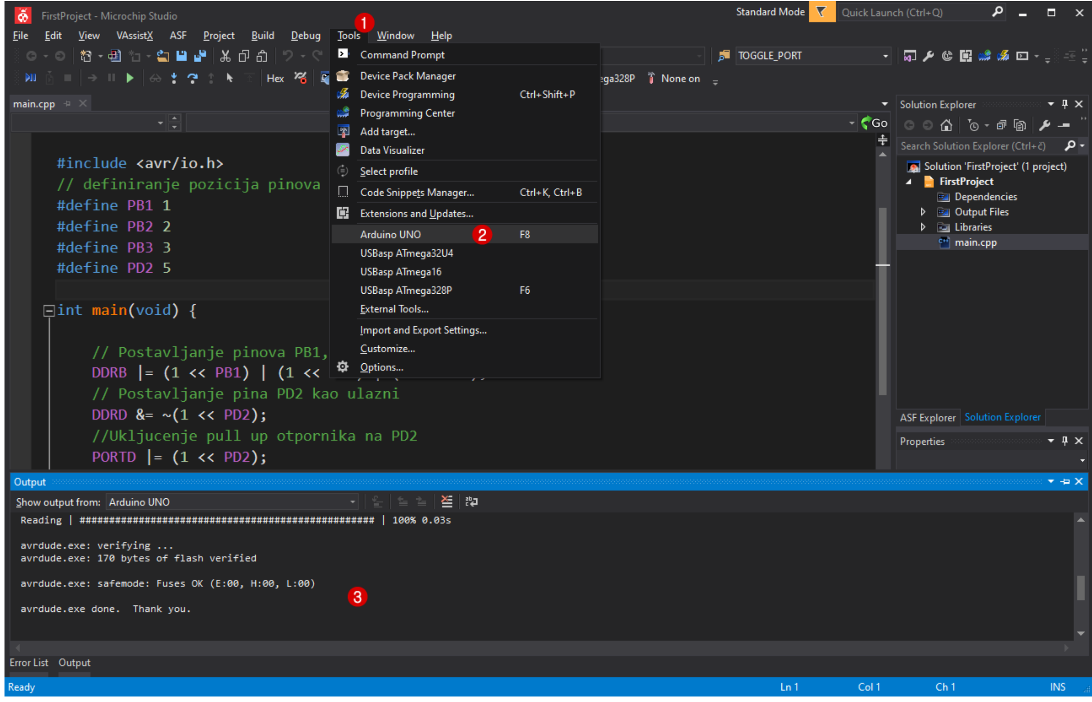
1. u programskom razvojnom okruženju *Microchip Studio* odaberite izbornik `Tools`,
2. u izborniku `Tools` odaberite `External Tools...`,
3. u novootvorenom prozoru pritisnite gumb `Add`,
4. u polju `Title` upišite: Arduino UNO,
5. u polje `Command` upišite: C:\Program Files (x86)\Atmel\avrdude\avrdude.exe[4]
6. u polje `Arguments` upišite:
   -c arduino -p m328p -P COM5 -b 115200 -U flash:w:$(ProjectDir)Debug\$(TargetName).hex:i
7. označite opciju `Use Output window`,
8. završite parametriranje pritiskom na tipku `OK`.

[4]Potrebno je skinuti *Avrdude* sa stranice `https://www.nongnu.org/avrdude/`.

Nakon provedenog postupka parametriranja programiranja pomoću Arduino *Bootloader* programa, strojni kod za mikroupravljač ATmega328P možete snimiti na mikroupravljač ATmega328P odabirom `Tools -> Arduino UNO` (slika 3.16 - brojevi 1 i 2). Uspješnost programiranja možete pratiti u pokazniku statusnih poruka (slika 3.16 - broj 3).



Slika 3.15: Programiranje mikroupravljača ATmega328P iz programskog razvojnog okruženja *Microchip Studio* - parametriranje programiranja pomoću Arduino *Bootloader* programa



Slika 3.16: Programiranje mikroupravljača iz programskog razvojnog okruženja *Microchip Studio* pomoću Arduino *Bootloader* programa

Poglavlje 4

Digitalni izlazi mikroupravljača ATmega328P

Digitalni pinovi mikroupravljača ATmega328P mogu se konfigurirati kao izlazni pinovi ili kao ulazni ponovi. Promjena konfiguracije pina moguća je i tijekom rada mikroupravljača. Na primjer, digitalni izlaz se u nekom trenutku može konfigurirati u digitalni ulaz. U ovom poglavlju bavit ćemo se digitalnim izlazima mikroupravljača.

Mikroupravljač ATmega328P ima 23 ulazno/izlazna pina opće namjene (engl. *General-Purpose Input/Output* - GPIO) koji su smješteni u tri grupe koje nazivamo portovima:

- **PORTB** (pinovi: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- **PORTC** (pinovi: PC0, PC1, PC2, PC3, PC4, PC5, PC6) i
- **PORTD** (pinovi: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7).

Digitalni pinovi mikroupravljača ATmega328P, osim što mogu biti digitalni izlazi ili ulazi, imaju i alternativnu namjenu. Tako se pinovi PC0 - PC5 mogu koristiti za analogno digitalnu pretvorbu. Konfiguracija digitalnih pinova na mikroupravljaču ATmega328P provodi se pomoću dva registra:

- **DDRx**, (x = B, C, D), (engl. *Data Direction Register*) - registar smjera podataka,
- **PORTx**, (x = B, C, D) - podatkovni registar.

Registri **DDRx** i **PORTx** širine su 8 bitova[1]. Registrom **DDRx** se određuje da li će neki pin na portu x biti ulazni ili izlazni (određuje se smjer toka podatka). Konfiguracija pojedinog pina na poziciji i ($i = 0, 1, ..., 7$) portova B, C i D određuje se postavljanjem bita u registru **DDRx** na zadanoj poziciji[2]. Ako želimo promijeniti konfiguraciju pina PC2, tada ćemo postavljati vrijednost u registru **DDRC** na poziciji bita $i = 2$.

Stanjem bita na poziciji i ($i = 0, 1, ..., 7$) u registru **DDRx**, (x = B, C, D), konfigurira se ulazni ili izlazni pin na poziciji i prema sljedećim pravilima:

- pin na poziciji i biti će konfiguriran kao izlazni pin ako je bit na poziciji i u registru **DDRx**, (x = B, C, D), jednak 1,
- pin na poziciji i biti će konfiguriran kao ulazni pin ako je bit na poziciji i u registru **DDRx**,

[1] Najveći broj registara u mikroupravljaču ATmega328P jest širine 8 bitova.
[2] Primijetite da **PORTC** ima šest pinova pa će pozicija bita biti i ($i = 0, 1, ..., 6$)

(x = B, C, D), jednak 0.

Ako u registar **DDRD** upišemo heksadecimalnu konstantu 0xAA (ili binarnu konstantu 0b10101010) pomoću naredbe `DDRD = 0xAA;`, tada će pinovi PD1, PD3, PD5 i PD7 biti konfigurirani kao izlazni pinovi, a PD0, PD2, PD4 i PD6 kao ulazni pinovi. Navedena konfiguracija prikazana je u tablici 4.1.

Tablica 4.1: Konfiguracija porta D sa sadržajem registra **DDRD** = 0xAA

DDRD registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **DDRD** registra	1	0	1	0	1	0	1	0
Port D	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
Konfiguracija pina	izlaz	ulaz	izlaz	ulaz	izlaz	ulaz	izlaz	ulaz

Ako je pin konfiguriran kao izlazni pin, tada on može biti postavljen u dva stanja:

- nisko stanje (logička nula, isključen pin) - stanje koje odgovara naponu 0 V,
- visoko stanje (logička jedinica, uključen pin) - stanje koje odgovara naponskom nivou `VCC`[3] V.

Stanje izlaznog pina određuje se pomoću registra **PORTx** prema sljedećim pravilima:

- ako je bit na poziciji i u registru **PORTx** jednak 0, tada će izlazni pin na poziciji i biti u niskom stanju,
- ako je bit na poziciji i u registru **PORTx** jednak 1, tada će izlazni pin na poziciji i biti u visokom stanju.

Za prethodnu konfiguraciju digitalnih pinova `DDRD = 0xAA`, pin PD1 može se uključiti tako da se na poziciju bita 1 u registru **PORTD** upiše vrijednost 1 (na primjer `PORTD |= 0x02`).

Tablica 4.2: Uključenje pina PD1 na portau D uz konfiguraciju porta u tablici 4.1

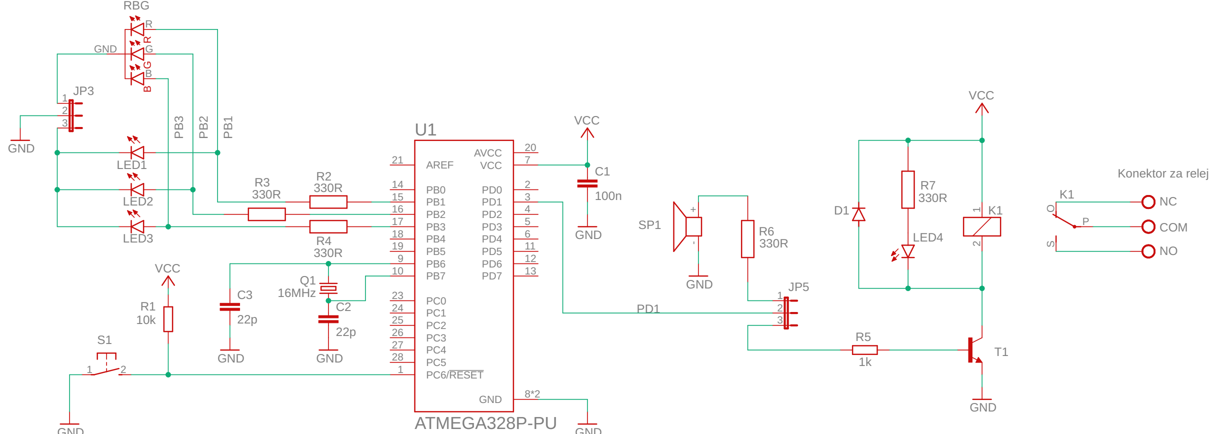
PORTD registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **PORTD** registra	0	0	0	0	0	0	1	0
Port D	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
Stanje pina	nisko	hi-Z	nisko	hi-Z	nisko	hi-Z	visoko	hi-Z*

*hi-Z (engl. *High impedance*) - visoka impendancija ili plivajuće (engl. *floating*) stanje pina.

4.1 Vježbe - digitalni izlazi mikroupravljača ATmega328P

Digitalne izlaze mikroupravljača testirat ćemo pomoću tri LED diode, zujalice i releja koji je na mikroupravljač ATmega328P spojen posredno preko tranzistorske sklopke. Shema spajanja tri LED diode, zujalice i releja na mikroupravljač ATmega328P prikazana je na slici 4.1.

[3] VCC može biti 5 V, 3.3 V ili neka druga razina što ovisi o napajanju mikroupravljača



Slika 4.1: Shema spajanja tri LED diode, zujalice i releja na mikroupravljač ATmega328P

LED diode, zujalica i relej su aktuatori. Aktuatori se mogu spojiti na bilo koji digitalni pin koji je konfiguriran kao izlaz pazeći pri tome na upravljačku struju aktuatora. Na primjer, digitalni pin u pravilu ne može biti protjecan strujom koja je dovoljna da uključi relej te se iz tog razloga koristi tranzistorska sklopka kao posrednik. U suprotnom, digitalni pin bi pregorio kada bi direktno bio spojen na upravljačke pinove releja[4]. Shema sa slike 4.1 usklađena je s razvojnim okruženjem prikazanim na slici 2.1.

Prema shemi na slici 4.1 crvena LED dioda spojena je na pin PB1, žuta LED dioda spojena je na pin PB2, a zelena LED dioda spojena je na pin PB3. Zujalica i relej spojeni su preko kratkospojnika JP5 na pin PD1. Kada je kratkospojnik JP5 spojen između trnova 1 i 2, tada je na mikroupravljač ATmega328P spojena zujalica. Ako je kratkospojnik JP5 spojen između trnova 2 i 3, tada je na mikroupravljač ATmega328P spojen relej. U ovoj vježbi koristit ćemo zujalicu, dok ćemo relej koristiti u posljednjem poglavlju ovog udžbenika. U seriju s LED diodama i zujalicom spojeni su otpornici otpora 330 Ω. Razlog tomu jest prekostrujna zaštita digitalnog pina. Struja digitalnog pina ne smije biti veća od 40 mA prema literaturi [2]. Općenito se na digitalne izlaze mikroupravljača mogu spojiti unipolarni tranzistori, releji s maksimalnom upravljačkom strujom iznosa 40 mA, optički sprežnici (engl. *optocoupler*), dijci, trijci i ostali digitalni aktuatori uz uvjet da se ne premaši maksimalna struja digitalnog pina.

Za detalje oko konfiguracije digitalnih izlaza mikroupravljača ATmega328P pogledajte tablicu 13-1 u literaturi [2] na stranici 60. Programsko razvojno okruženje *Microchip Studio*ima predefinirane konstante (makronaredbe) za imena registara **DDRx** i **PORTx** (x = B, C i D). Konfiguracija pinova svodi na dodjeljivanje vrijednosti u predefinirana imena registara. Na primjer, ako želimo da svi pinovi porta B budu izlazni pinovi u programskom razvojnom okruženju *Microchip Studio*, napisat ćemo `DDRB = 0xFF;` u heksadecimalnom zapisu ili `DDRB = 0b11111111;` u binarnom zapisu. Općenito vrijedi da su imena registara koja se koriste u literaturi [2] jednaka imenima definiranim u programskom razvojnom okruženju *Microchip Studio*. Isto vrijedi i za imena bitova unutar pojedinog registra.

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `Digitalni izlazi.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `Digitalni izlazi.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\Digitalni izlazi\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Digitalni`

[4]Vrlo rijetko koriste se releji koji imaju malu upravljačku struju (do 40 mA) pa bi se u tom slučaju takav relej s porednom diodom mogao spojiti na digitalni pin.

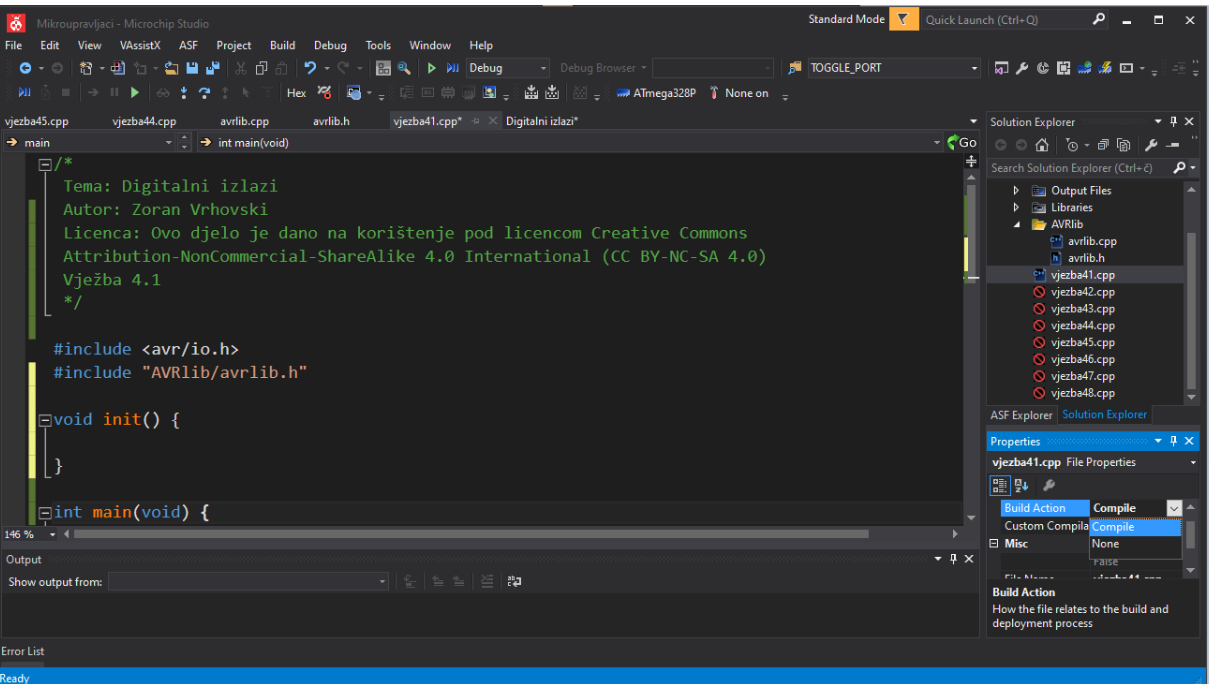
`izlazi mikroupravljača ATmega328P`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. Budući da će svaka datoteka u koju ćemo pisati vježbe sadržavati funkciju `main()`[5], za svaku vježbu potrebno je napraviti sljedeće korake (slika 4.2):

1. u projektnom stablu projekta `Digitalni izlazi` odaberite datoteku s ekstenzijom `*.cpp` (npr. `vjezba41.cpp`) i na njoj pritisnite desnu tipku miša te odaberite `Properties`,

2. ispod prozora projektnog stabla otvorit će se prozor sa svojstvima datoteke koju ste odabrali. Za datoteku koju želite prevesti u strojni kod potrebno je u polju *Build Action* odabrati opciju *Compile*, a za sve ostale datoteke u projektnom stablu koje sadrže funkciju `main()` u polju *Build Action* odabrati opciju *None*.

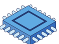
Provedbom prethodnih koraka osigurali smo da se uvijek samo jedna vježba prevodi u strojni kod (`*.hex` datoteka). Ukoliko je na više vježbi u polju *Build Action* odabrana opcija *Compile*, prevoditelj će javiti grešku da u programskom kodu postoji višestruka definicija glavne funkcije `main()`, što je prema pravilima programskog jezika C/C++ neispravno.

U budućim vježbama nećemo više objašnjavati ovaj korak, već ćemo ga podrazumijevati. U datoteci s vježbama koju ste preuzeli s mrežne stranice `https://vub.hr/atmega328p` nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.



Slika 4.2: Odabir datoteke koja će se prevoditi u strojni kod

[5] U programskom jeziku C/C++ poznato je da program može imati samo jednu funkciju `main()`.

 Vježba 4.1

Napišite program koji će uključiti crvenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni izlaz PB1 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba41.cpp`. Omogućite prevođenje datoteke `vjezba41.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba41.cpp` prikazan je programskim kodom 4.1.

Programski kod 4.1: Početni sadržaj datoteke `vjezba41.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

}

int main(void) {

    init();

    return 0;
}
```

U nastavku ćemo objasniti linije programskog koda 4.1:

- `#include "AVR/avrlib.h"` - zaglavlje `avrlib.h` sadrži:
  - definirane makronaredbe za konfiguraciju, promjenu stanja i ispitivanje stanja digitalnih pinova,
  - definirane konstante za funkcijski pristup konfiguraciji, promjeni stanja i ispitivanju stanja digitalnih pinova,
  - deklarirane funkcije za konfiguraciju, promjenu stanja i ispitivanje stanja digitalnih pinova.

  Zaglavlje `avrlib.h` napisano je od strane korisnika te se u programski kod uključuje pretprocesorskom naredbom `#include ""`.

- `#include <avr/io.h>` - zaglavlje `io.h` sadrži definirane makronaredbe i funkcije za manipulaciju s digitalnim ulazima i izlazima. Ovo zaglavlje u obzir uzima mikroupravljač za koji je stvoren projekt u programskom razvojnom okruženju *Microchip Studio*. Zaglavlje `io.h` razvijeno je u tvrtki *Microchip* te se u programski kod uključuje pretprocesorskom naredbom `#include <>`.

- `void init()` - funkcija koju ćemo koristiti za inicijalizaciju mikroupravljača. Ova funkcija koristi se za konfiguraciju rada mikroupravljača. Naziv funkcije za inicijalizaciju mikroupravljača proizvoljan je.

- `main(void)` - glavna funkcija u programskom jeziku C/C++ koju mikroupravljač prvu počinje izvoditi.

Mikroupravljač programski kod izvodi počevši od glavne funkcije `main(void)`. Prva funkcija koja se poziva i izvodi na mikroupravljaču jest inicijalizacijska funkcija `init()`. Nakon što se

izvede tijelo funkcije `init()`, programski kod se nastavlja izvoditi. Sljedeća linija programskog koda koja se izvodi jest `return 0;`. Izvođenjem ove linije koda, završava izvođenje glavne funkcije `main(void)`, a time i mikroupravljač završava s izvođenjem programskog koda koji smo u obliku strojnog koda upisali na njega.

Problem koji moramo riješiti jest napisati program koji će uključiti crvenu LED diodu koja je spojena na digitalni pin PB1 mikroupravljača ATmega328P. LED dioda je elektronička komponenta kojoj je između elektroda anode i katode potrebno dovesti napon praga (koljena) kako bi emitirala svjetlost. Pin na koji je spojena LED dioda se, prema tome, mora konfigurirati kao izlazni pin. Kako smo naveli u uvodu ovog poglavlja, izlazni pin mikroupravljača konfigura se tako da se u `DDRB` registar bit na poziciji pina PB1 postavi u 1, a svi ostali bitovi u 0. Prikaz konfiguracije porta B za izlazni pin PB1 prikazan je u tablici 4.3. Vrijednost konstante koju je potrebno upisati u registar `DDRB` jest `0x02` (binarno `0b00000010`[6]).

Tablica 4.3: Konfiguracija porta B za izlazni pin PB1

DDRB registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **DDRB** registra	0	0	0	0	0	0	1	0
Port B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Konfiguracija pina	ulaz	ulaz	ulaz	ulaz	ulaz	ulaz	izlaz	ulaz

Sada kada smo digitalni pin PB1 definirali kao izlaz, potrebno ga je postaviti u visoko stanje. Kako smo već spomenuli, stanje izlaznog pina mikroupravljača postavlja se pomoću `PORTB` registara tako da za visoko stanje bit na poziciji pina PB1 postavimo u 1, a sve ostale bitove u 0. Postavljanje izlaznog pina PB1 u visoko stanje prikazano je u tablici 4.4. Prema tome, sadržaj registra `PORTB` bit će `0x02`.

Tablica 4.4: Postavljanje izlaznog pina PB1 u visoko stanje

PORTB registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **PORTB** registra	0	0	0	0	0	0	1	0
Port B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Stanje pina	hi-Z	hi-Z	hi-Z	hi-Z	hi-Z	hi-Z	visoko	hi-Z

Da bismo ostvarili zadani zadatak, programski kod 4.1 potrebno je modificirati na sljedeći način:

- upišite u funkciju `init()` konstantu `DDRB = 0x02;`,
- u novi redak funkcije `init()` upišite konstantu `PORTB = 0x02;`.

Programski kod 4.2: Sadržaj funkcije `init()` mikroupravljača ATmega328P - prvi način

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    DDRB = 0x02;
```

[6] Iako konstanta 0b00000010 nije podržana C standardom, Microchip Studio podržava zapis binarne konstante u varijablu. Programeri mikroupravljača češće koriste heksadecimalne konstante.

```
    PORTB = 0x02;
}

int main(void) {

    init();

    return 0;
}
```

Tijelo inicijalizacijske funkcije `init()` mora odgovarati programskom kodu 4.2. Ako je Vaš programski kod u razvojnom okruženju *Microchip Studio* istovjetan programskom kodu 4.2, tada je datoteku `vjezba41.cpp` potrebno prevesti u strojni kod sukladno uputama iz poglavlja `Programsko razvojno okruženje Microchip Studio`. Strojni kod pomoću softvera *AVRDUDESS* snimite na mikroupravljač ATmega328P prema uputama iz poglavlja `Programiranje mikroupravljača ATmega328P`. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P trebala bi svijetliti crvena LED dioda.

Naveden način konfiguracije mikroupravljača i postavljanje digitalnog pina u visoko stanje jedan je od načina kako se to može provesti kroz inicijalizacijsku funkciju `init()`. Ovaj pristup inicijalizacije mikroupravljača može se koristiti samo ako su sve funkcije digitalnih pinova na portu B poznate ili ako se koristi samo digitalni pin PB1. U suprotnom, ovaj način inicijalizacije može dovesti do krivog rada mikroupravljača te ga stoga ne preporučamo kao prvi izbor. U praktičnoj primjeni mikroupravljača često je potrebno konfigurirati i postaviti samo ciljani digitalni pin, a da se konfiguracija i stanje ostalih digitalnih pinova ne mijenja. U tu svrhu koriste se bitovni operatori i operator posmaka.

Modificirajte programski kod 4.2 na sljedeći način:

- umjesto naredbe `DDRB = 0x02;` upišite naredbu `DDRB |= (1 << PB1);`,
- umjesto naredbe `PORTB = 0x02;` upišite naredbu `PORTB |= (1 << PB1);`.

Programski kod 4.3: Sadržaj funkcije `init()` mikroupravljača ATmega328P - drugi način

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    DDRB |= (1 << PB1);
    PORTB |= (1 << PB1);
}

int main(void) {

    init();

    return 0;
}
```

Tijelo inicijalizacijske funkcije `init()` sada mora odgovarati programskom kodu 4.3. U programskom kodu 4.3 koriste se bitovne operacije ILI i posmak. Kod bitovne operacije posmaka (`<<`) koristi se definirana konstanta `PB1` čija je vrijednost 1, odnosno odgovara poziciji digitalnog pina na portu B. Ova konstanta (kao i brojne druge) definirane su u zaglavlju `io.h` i produkt su tvrtke *Microchip*. Definirane konstante možete pregledati tako da u programskom okruženju *Microchip Studio*iznad konstante `PB1` pritisnete desnu tipku miša te

odaberete `Goto Implementation`. Otvorit će se zaglavlje u kojem su definirane konstante za sve registre korištenog mikroupravljača. Proučite ostali sadržaj otvorenog zaglavlja. Opcija `Goto Implementation` vrlo je korisna i preporučuje se njezino često korištenje za sve definirane konstante i makronaredbe kako bi se bolje razumio programski kod. Definirane konstante i makronaredbe prepoznat ćete po ljubičastoj boji, a nazivi su isti nazivima korištenima u literaturi [2].

U tablici 4.5 prikazani su bitovni operator ILI i bitovni operator posmaka ulijevo korišteni za konfiguriranje izlaznog pina te za postavljanje izlaznog digitalnog pina u visoko stanje. Konstanta 1 bitovno se posmiče ulijevo za jedno mjesto naredbom `1 << PB1`. Na taj smo način broj 1 pozicionirali ispod bita broj 1 (pozicija pina PB1). Pretpostavimo da su stanja bitova registra `DDRB` nepoznata i ta stanja označit ćemo s x. Ako želimo da pin PB1 bude izlazni pin, tada na mjestu bita broj 1 u registru `DDRB` moramo postaviti 1. To ćemo ostvariti tako da koristimo bitovni ILI operator. Stanje registra `DDRB` i konstantu koja je dobivena naredbom `1 << PB1` podvrgnemo bitovnom ILI operatoru, a rezultat spremimo nazad u registar `DDRB`. Na taj smo način samo bit na poziciji 1 postavili u 1, a ostali bitovi ostali su nepromijenjeni. Isto ćemo postupiti i za postavljanje pina PB1 u visoko stanje (tablica 4.5).

Tablica 4.5: Bitovni operator ILI i bitovni operator posmaka korišteni za konfiguriranje izlaznog pina

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	0	0	0	0	0	0	0	1
1 « **PB1**	0	0	0	0	0	0	1	0
Stanje registra **DDRB**	x	x	x	x	x	x	x	x
DDRB |= (1 « **PB1**)	x	x	x	x	x	x	1	x
Stanje registra **PORTB**	x	x	x	x	x	x	x	x
PORTB |= (1 « **PB1**)	x	x	x	x	x	x	1	x

Ponovno prevedite datoteku `vjezba41.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Navedeni način konfiguracije digitalnih pinova i postavljanje digitalnih pinova je najsloženiji, ali se najčešće koristi u praksi.

S obzirom na složenost prethodne inicijalizacije mikroupravljača, izradili smo vlastite makronaredbe kojima se mogu konfigurirati digitalni pinovi i makronaredbe kojima se mogu postavljati stanja pinova. Vlastite makronaredbe nalaze se u zaglavlju `avrlib.h`.

Za konfiguraciju izlaznih pinova i za postavljanje stanja digitalnih pinova koristit ćemo sljedeće makronaredbe:

- `config_output(DDRx, Pxi)` - makronaredba koja kao argumente prima registar `DDRx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) kojeg želimo postaviti kao izlazni pin,

- `set_pin_on(PORTx, Pxi)` - makronaredba koja kao argumente prima registar `PORTx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) kojeg želimo postaviti u visoko stanje,

- `set_pin_off(PORTx, Pxi)` - makronaredba koja kao argumente prima registar `PORTx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) kojeg želimo postaviti u

nisko stanje.

Primjenu navedenih makronaredbi prikazat ćemo na nekoliko sljedećih primjera:

- `config_output(DDRB, PB1)` - pin PB1 konfiguriran je kao izlazni pin,
- `config_output(DDRB, PB4)` - pin PB4 konfiguriran je kao izlazni pin,
- `config_output(DDRC, PC0)` - pin PC0 konfiguriran je kao izlazni pin,
- `config_output(DDRD, PD6)` - pin PD6 konfiguriran je kao izlazni pin,
- `set_pin_on(PORTB, PB1)` - visoko stanje postavljeno na pinu PB1,
- `set_pin_off(PORTB, PB4)` - nisko stanje postavljeno na pinu PB4,
- `set_pin_on(PORTC, PC0)` - visoko stanje postavljeno na pinu PC0,
- `set_pin_off(PORTD, PD6)` - nisko stanje postavljeno na pinu PD6.

Slijedom navedenih primjera, modificirajte programski kod 4.3 na sljedeći način:

- umjesto naredbe `DDRB |= (1 << PB1);` upišite naredbu `config_output(DDRB, PB1)`,
- umjesto naredbe `PORTB |= (1 << PB1);` upišite naredbu `set_pin_on(PORTB, PB1)`.

Tijelo inicijalizacijske funkcije `init()` mora odgovarati programskom kodu 4.4. Ovaj programski kod, kao i prethodna dva, konfigurira pin PB1 kao izlazni te uključuje crvenu LED diodu koja je spojena na njega.

Programski kod 4.4: Sadržaj funkcije `init()` mikroupravljača ATmega328P - treći način

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    config_output(DDRB, PB1);
    set_pin_on(PORTB, PB1);
}

int main(void) {

    init();

    return 0;
}
```

Prednost pristupa konfiguraciji digitalnih pinova i postavljanja stanja pinova pomoću makronaredbi je u tome što ne morate voditi računa o poziciji pina, o bitovnim operacijama i slično. Što se brzine izvedbe programa tiče, programski kod 4.3 i programski kod 4.4 se jednako brzo izvode na mikroupravljaču. Znatiželjni čitatelj koji prouči makronaredbe `config_output()` i `set_pin_on()` primijetiti će da su programski kodovi 4.3 i 4.4 potpuno jednaki. Naime, prilikom pretprocesiranja programskog koda 4.4, on se zamjenjuje kodom koji je istovjetan programskom kodu 4.4.

Prevedite datoteku `vjezba41.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Pretpostavljamo da su mnogi od Vas svoje prve korake u programiranju mikroupravljača

napravili s *Arduino* razvojnim okruženjima. U *Arduino* IDE programskom razvojnom okruženju koristi se funkcijski pristup za konfiguriranje pinova i postavljanje stanja na digitalne izlaze. Autori ovog udžbenika napisali su funkcije koje se koriste na identičan način kao i *Arduino* funkcije za digitalne pinove. Funkcije su deklarirane u zaglavlju `avrlib.h`.

Pri funkcijskom pristupu za konfiguraciju izlaznih pinova i za postavljanje stanja digitalnih pinova koristit ćemo sljedeće funkcije:

- `pinMode(xi, OUTPUT);` - funkcija koja prima dva argumenta. Prvi argument je konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) koja predstavlja ime porta i poziciju pina koji želimo konfigurirati. Drugi argument je konstanta koja određuje da li će pin biti ulazni ili izlazni. Za izlazni pin koristi se konstanta `OUTPUT`.

- `digitalWrite(xi, HIGH);` - funkcija koja prima dva argumenta. Prvi argument je konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) koja predstavlja ime porta i poziciju pina koji želimo konfigurirati. Drugi argument je stanje koje želimo postaviti na digitalni izlaz. Za visoko stanje izlaznog pina koristi se konstanta `HIGH`, a za nisko stanje `LOW`.

Važno je naglasiti da konstante `B1` i `PB1` nisu iste konstante, ali se odnose na isti digitalni pin (pin PB1). Konstanta `B1` koristi se pri funkcijskom pristupu konfiguracije digitalnih pinova, a konstanta `PB1` se koristi pri korištenju makronaredbi za konfiguriranje digitalnih pinova.

Primjenu navedenih funkcija prikazat ćemo na nekoliko sljedećih primjera:

- `pinMode(B1, OUTPUT)` - pin PB1 konfiguriran je kao izlazni pin,
- `pinMode(B4, OUTPUT)` - pin PB4 konfiguriran je kao izlazni pin,
- `pinMode(C0, OUTPUT)` - pin PC0 konfiguriran je kao izlazni pin,
- `pinMode(D6, OUTPUT)` - pin PD6 konfiguriran je kao izlazni pin,
- `digitalWrite(B1, HIGH)` - visoko stanje postavljeno na pinu PB1,
- `digitalWrite(B4, LOW)` - nisko stanje postavljeno na pinu PB4,
- `digitalWrite(C0, HIGH)` - visoko stanje postavljeno na pinu PC0,
- `digitalWrite(D6, LOW)` - nisko stanje postavljeno na pinu PD6.

Neispravno bi bilo napisati `pinMode(PB1, OUTPUT)`, kao i `config_output(DDRB, B1)`. Slijedom navedenih primjera, modificirajte programski kod 4.4 na sljedeći način:

- umjesto naredbe `config_output(DDRB, PB1)` upišite naredbu `pinMode(B1, OUTPUT);`,
- umjesto naredbe `set_pin_on(PORTB, PB1)` upišite naredbu `digitalWrite(B1, HIGH)`.

Tijelo inicijalizacijske funkcije `init()` mora odgovarati programskom kodu 4.5. Prevedite datoteku `vjezba41.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Četvrti način konfiguracije digitalnih pinova (programski kod 4.5) je najintuitivniji i najlakši za korištenje. U praksi se najviše koristi drugi način (programski kod 4.3). Analizom zauzeća programske memorije svih navedenih programskih kodova koji obavljaju isti zadatak dolazimo do sljedećih rezultata:

- programski kod 4.2 zauzima 240 B programske memorije,

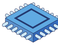
- programski kod 4.3 zauzima 238 B programske memorije,
- programski kod 4.4 zauzima 238 B programske memorije,
- programski kod 4.5 zauzima 540 B programske memorije.

Funkcijski pristup konfiguraciji pinova (programski kod 4.5) zauzima najviše programske memorije što je cijena jednostavnosti korištenja. Najmanje programske memorije zauzimaju programski kodovi 4.3 i 4.4. Primijetite da ova dva koda zauzimaju istu količinu programske memorije što smo prethodno i spomenuli jer su to istovjetni programski kodovi. U mikroupravljačima, zauzeće manje programske memorije znači brže izvođenje programa jer najčešće jedan bajt programske memorije predstavlja jednu instrukciju strojnog koda koja se izvede na mikroupravljaču.

Programski kod 4.5: Sadržaj funkcije `init()` mikroupravljača ATmega328P - četvrti način

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    pinMode(B1, OUTPUT);
    digitalWrite(B1, HIGH);
}

int main(void) {

    init();

    return 0;
}
```

Zatvorite datoteku `vjezba41.cpp` i onemogućite prevođenje ove datoteke.



Vježba 4.2

Napišite program koji će uključiti žutu i zelenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P. Prema shemi na slici 4.1, žuta LED dioda spojena je na digitalni izlaz PB2 mikroupravljača ATmega328P, a zelena LED dioda na digitalni izlaz PB3.

U projektnom stablu otvorite datoteku `vjezba42.cpp`. Omogućite prevođenje datoteke `vjezba42.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba42.cpp` prikazan je programskim kodom 4.6.

Programski kod 4.6: Početni sadržaj datoteke `vjezba42.cpp`

```
int main(void) {

    return 0;
}
```

U programskom kodu 4.6 nedostaju brojne naredbe koje su potrebne za realizaciju zadanog zadatka. S obzirom da ćemo koristiti registre i konstante koje su povezane s mikroupravljačem ATmega328P, u programski kod je potrebno uključiti zaglavlja `io.h` i `avrlib.h`. U programski kod 4.6 dodajte sljedeće dvije naredbe iznad glavne funkcije:

- `#include <avr/io.h>`
- `#include "AVR/avrlib.h"`

Iako nije nužno, dobra praksa je u programiranju mikroupravljača imati inicijalizacijsku funkciju. U programski kod 4.6 iza prethodno uključenih zaglavlja dodajte definiciju inicijalizacijske funkcije: `void init(){ }`. Funkciju `init()` pozovite na početku glavne funkcije. Vaš program sada izgleda kao programski kod 4.1.

Problem koji moramo riješiti jest napisati program koji će uključiti zelenu i žutu LED diodu koje su spojene na digitalne pinove PB2 i PB3. Koristit ćemo pristup s bitovnim ILI operatorom i bitovnim operatorom posmaka. Modificirajte programski kod 4.6 na sljedeći način:

- upišite u funkciju `init()` naredbu `DDRB |= (1 << PB2)| (1 << PB3);`,
- u novi redak funkcije `init()` upišite naredbu `PORTB |= (1 << PB2)| (1 << PB3);`.

Konačan program je prikazan programskim kodom 4.7. Kada je potrebno uključiti dva ili više ciljana bita u registru, koristimo niz bitovnih posmaka koji su povezani međusobno s bitovnim ILI. Na primjer, dio naredbe `(1 << PB2)| (1 << PB3)` omogućit će postavljanje bitova na poziciji 2 i 3 u registru `DDRB`.

Programski kod 4.7: Program koji uključuje zelenu i žutu LED diodu - prvi način

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    // PB2 i PB3 konfigurirani kao izlazni pinovi
    DDRB |= (1 << PB2) | (1 << PB3);
    // visoko stanje postavljeno na pinove PB2 i PB3
    PORTB |= (1 << PB2) | (1 << PB3);
}

int main(void) {

    init();

    return 0;
}
```

Prevedite datoteku `vjezba42.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P trebale bi svijetliti žuta i zelena LED dioda.

Napravite zadani zadatak tako da napravite konfiguraciju pomoću makronaredaba i funkcija koje smo naučili u prethodnoj vježbi. Rješenja su prikazana programskim kodovima 4.8 i 4.9. Testirajte oba programa na razvojnom okruženju s mikroupravljačem ATmega328P.

Programski kod 4.8: Program koji uključuje zelenu i žutu LED diodu - primjena makronaredaba

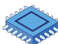
```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {
```

```
    config_output(DDRB, PB2); // PB2 konfiguriran kao izlazni pinovi
    config_output(DDRB, PB3); // PB3 konfiguriran kao izlazni pinovi
    set_pin_on(PORTB, PB2); // pin PB2 postavljen u visoko stanje
    set_pin_on(PORTB, PB3); // pin PB3 postavljen u visoko stanje
}

int main(void) {

    init();

    return 0;
}
```

Programski kod 4.9: Program koji uključuje zelenu i žutu LED diodu - primjena funkcija

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pinovi
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pinovi
    digitalWrite(B2, HIGH); // pin PB2 postavljen u visoko stanje
    digitalWrite(B3, HIGH); // pin PB3 postavljen u visoko stanje
}

int main(void) {

    init();

    return 0;
}
```

Zatvorite datoteku `vjezba42.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 4.3

Napišite program koji će četiri puta uključiti i isključiti zelenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P. Vremenski razmak između uključivanja i isključivanja zelene LED diode neka je jedna sekunda. Prema shemi na slici 4.1 zelena LED dioda spojena je na digitalni izlaz PB3 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba43.cpp`. Omogućite prevođenje datoteke `vjezba43.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba43.cpp` prikazan je programskim kodom 4.10.

Programski kod 4.10: Početni sadržaj datoteke `vjezba43.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pinovi

}
```

```
int main(void) {

    init(); // inicijalizacija mikroupravljaca

    return 0;
}
```

Funkcionalnost koju moramo ostvariti jest ta da zelena LED dioda mora četiri puta izmijeniti stanje uključenosti i isključenosti s vremenskim razmakom između stanja od jedne sekunde. Novi problem koji se pojavio u ovom zadatku jest ostvarivanje promjene stanja digitalnog izlaza u točno određenom vremenu. Postoji barem dva načina kako se ovaj problem može riješiti, a s obzirom da smo još na početku udžbenika, krenut ćemo s lakšim načinom. Koristit ćemo funkcije za vremensko kašnjenje izvođenja programskog koda. Funkcije za vremensko kašnjenje nalaze se u zaglavlju `delay.h`. U programski kod 4.10 upišite naredbu `#include <util/delay.h>` kako bi omogućili korištenje funkcija za kašnjenje. U zaglavlju `delay.h` nalaze se sljedeće funkcije za vremensko kašnjenje [1]:

- `_delay_ms(double)` - funkcija koja kao argument prima realan broj dvostruke preciznosti koji predstavlja kašnjenje u ms,
- `_delay_us(double)` - funkcija koja kao argument prima realan broj dvostruke preciznosti koji predstavlja kašnjenje u μs.

Funkcije za vremensko kašnjenje `_delay_ms(double)` i `_delay_us(double)` zaustavit će provođenje programskog koda na vrijeme koje je zadano argumentom funkcije[7]. Ovo je u ovom trenutku najlakši način na koji možemo osigurati promjenu stanja digitalnog izlaza u zadanim vremenima. U praksi je potrebno, koliko je god to moguće, izbjegavati korištenje funkcija za vremensko kašnjenje `_delay_ms(double)` i `_delay_us(double)`. Naravno, ponekad je nemoguće izbjeći njihovo korištenje.

Kako bi funkcije vremenskog kašnjenja `_delay_ms(double)` i `_delay_us(double)` mogle ostvariti ispravna vremenska kašnjenja, potrebno je pravilno namjestiti *Fuse* bitove te u programskom okruženju *Microchip Studio*definirati konstantu `F_CPU`. *Fuse* bitovi su na našem razvojnom okruženju namješteni na način da se podrazumijeva korištenje vanjskog oscilatora visoke frekvencije. Na razvojnom okruženju s mikroupravljačem ATmega328P koristi se oscilator kojim se generira radni takt frekvencije 16 MHz. Konstantu `F_CPU` je moguće redefinirati u zaglavlju `avrlib.h`. Naredbom `#define F_CPU 16000000ul` koja se nalazi u zaglavlju `avrlib.h` ukazujemo prevoditelju da frekvencija radnog takta mikroupravljača ATmega328P iznosi 16 MHz. Provjerite konstantu `F_CPU` koja se nalazi u zaglavlju `avrlib.h`. Ako se konstanta `F_CPU` ne redefinira, tada će njen iznos biti 1 MHz. Treba imati na umu da se konstantom `F_CPU` ne namješta frekvencija mikroupravljača na hardverskoj razini, već se korištenim bibliotekama u programskom razvojnom okruženju *Microchip Studio*daje informacija da je frekvencija mikroupravljača jednaka konstanti `F_CPU`. Već nakon prvog korištenja funkcija vremenskog kašnjenja `_delay_ms(double)` i `_delay_us(double)` utvrdit ćete da li su vam usklađene frekvencije na razini hardvera i softvera. Brojne biblioteke koriste konstantu `F_CPU`, stoga je vrlo važno da se ona ispravno definira. Također, važno je da se konstanta `F_CPU` redefinira prije uključivanja onih zaglavlja koja tu konstantu koriste.

Prema svemu navedenom, program koji moramo napraviti će uključiti zelenu LED diodu, omogućiti vremensko kašnjenje od jedne sekunde, isključiti zelenu LED diodu, omogućiti vremensko kašnjenje od jedne sekunde i tako četiri puta.

U programskom kodu 4.10 u inicijalizacijskoj funkciji `init()` pin PB3 je konfiguriran kao izlazni pin. U glavnu funkciju, nakon poziva funkcije `init()`, potrebno je unijeti sljedeće naredbe:

[7] Ove funkcije u programskoj petlji izvode instrukciju NOP (engl. *No operation*) koja traje jedan radni takt.

- `PORTB |= (1 << PB3);` - postavljanje pina PB3 u visoko stanje. Ovo je prvi korak jer je na početku zelena LED dioda ugašena.
- `_delay_ms(1000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000 ms = 1 s.
- `PORTB &= ~(1 << PB3);` - postavljanje pina PB3 u nisko stanje. Da bi se isključio ciljani bit koristi se I bitovni operator, bitovni operator komplementa i bitovni operator posmaka. Isključivanje ciljanog bita prikazano je u tablici 4.6.
- `_delay_ms(1000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000 ms = 1 s.
- `set_pin_on(PORTB, PB3);` - postavljanje pina PB3 u visoko stanje.
- `_delay_us(1000000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000000 μs = 1 s.
- `set_pin_off(PORTB, PB3);` - postavljanje pina PB3 u nisko stanje pomoću makronaredbe.
- `_delay_us(1000000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000000 μs = 1 s.
- `digitalWrite(B3, HIGH);` - postavljanje pina PB3 u visoko stanje pomoću funkcije.
- `_delay_ms(1000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000 ms = 1 s.
- `digitalWrite(B3, LOW);` - postavljanje pina PB3 u nisko stanje pomoću funkcije.
- `_delay_ms(1000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000 ms = 1 s.
- `PORTB |= (1 << PB3);` - postavljanje pina PB3 u visoko stanje.
- `_delay_ms(1000);` - funkcija koja omogućuje vremensko kašnjenje iznosa 1000 ms = 1 s.
- `PORTB &= ~(1 << PB3);` - postavljanje pina PB3 u nisko stanje što je ujedno i zadnji korak.

Konačan program je prikazan programskim kodom 4.11.

Tablica 4.6: Bitovni operator I i bitovni operator posmaka korišteni za isključivanje ciljanog bita

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	0	0	0	0	0	0	0	1
1 « **PB3**	0	0	0	0	1	0	0	0
∼ (1 « **PB3**)	1	1	1	1	0	1	1	1
Stanje registra **PORTB**	x	x	x	x	1	x	x	x
PORTB &= ∼(1 « **PB3**)	x	x	x	x	0	x	x	x

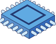
Programski kod 4.11: Program kojim zelena LED dioda četiri puta izmjeni stanje uključenosti i isključenosti u vremenskim razmacima od jedne sekunde

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>
```

```
void init() {

    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pinovi

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    PORTB |= (1 << PB3); // pin PB3 postavljen u visoko stanje
    _delay_ms(1000); // zakasni 1000 ms = 1 s
    PORTB &= ~(1 << PB3); // pin PB3 postavljen u nisko stanje
    _delay_ms(1000); // zakasni 1000 ms = 1 s
    set_pin_on(PORTB, PB3); // pin PB3 postavljen u visoko stanje
    _delay_us(1000000); // zakasni 1000000 us = 1 s
    set_pin_off(PORTB, PB3); // pin PB3 postavljen u nisko stanje
    _delay_us(1000000); // zakasni 1000000 us = 1 s
    digitalWrite(B3, HIGH); // pin PB3 postavljen u visoko stanje
    _delay_ms(1000); // zakasni 1000 ms = 1 s
    digitalWrite(B3, LOW); // pin PB3 postavljen u nisko stanje
    _delay_ms(1000); // zakasni 1000 ms = 1 s
    PORTB |= (1 << PB3); // pin PB3 postavljen u visoko stanje
    _delay_ms(1000); // zakasni 1000 ms = 1 s
    PORTB &= ~(1 << PB3); // pin PB3 postavljen u visoko stanje

    return 0;
}s
```

Prevedite datoteku `vjezba43.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P zelena LED dioda trebala bi četiri puta izmijeniti stanje s vremenskim razmacima od jedne sekunde. Postavlja se pitanje kako osigurati trajnu izmjenu stanja zelene LED diode? Odgovor na ovo pitanje dat ćemo u sljedećoj vježbi.

Zatvorite datoteku `vjezba43.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 4.4

Napišite program koji će na razvojnom okruženju s mikroupravljačem ATmega328P osigurati beskonačno "trčanje" svih LED dioda svakih 250 ms. Redoslijed trčanja LED dioda neka je crvena → žuta → zelena → crvena → žuta → ... Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba44.cpp`. Omogućite prevođenje datoteke `vjezba44.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba44.cpp` prikazan je programskim kodom 4.12.

Programski kod 4.12: Početni sadržaj datoteke `vjezba44.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

}
```

```
int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

    }

    return 0;
}
```

Funkcionalnost programskog koda koju moramo ostvariti jest „trčanje" LED dioda na način da trči uključena LED dioda. Trčanje LED dioda ostvaruje se na sljedeći način:

- uključite crvenu LED diodu, isključite zelenu LED diodu te omogućite vremensko kašnjenje od 250 ms,
- uključite žutu LED diodu, isključite crvenu LED diodu te omogućite vremensko kašnjenje od 250 ms,
- uključite zelenu LED diodu, isključite žutu LED diodu te omogućite vremensko kašnjenje od 250 ms,
- beskonačno ponavljajte prethodna tri koraka.

Prema prethodnom opisu, pretpostavlja se beskonačno trčanje LED dioda, odnosno dok god mikroupravljač ima napajanje. Da bismo to ostvarili, potrebno je koristiti programske petlje. U našem slučaju ta programska petlja mora biti beskonačna. Primjeri programskih petlji dani su programskim kodovima 4.13 i 4.14. Svaki koristan programski kod napisan za mikroupravljač mora imati beskonačnu programsku petlju, jer u suprotnom će mikroupravljač odraditi niz instrukcija i zaustaviti se (kao u svim dosadašnjim primjerima kada se pozvala naredba `return 0;`). Koju od beskonačnih programskih petlji ćete koristiti, svejedno je. Mi ćemo se u nastavku udžbenika opredijeliti za beskonačnu `while` petlju (programski kod 4.13).

Programski kod 4.13: Beskonačna `while` petlja

```
    while (1) { // beskonačna petlja

        // kod koji se izvršava u petlji
    }
```

Programski kod 4.14: Beskonačna `for` petlja

```
    for (;;) { // beskonačna petlja

        // kod koji se izvršava u petlji
    }
```

Početni sadržaj datoteke `vjezba44.cpp` prikazan programskim kodom 4.12 ne sadrži konfiguraciju digitalnih izlaznih pinova u inicijalizacijskoj funkciji `init()`. Modificirajmo programski kod 4.12 na sljedeći način:

- upišite u funkciju `init()` naredbu `DDRB |= (1 << PB1)| (1 << PB2)| (1 << PB3);`.

U zaglavlju `avrlib.h` nalazi se makronaredba `toggle_pin(PORTx, Pxi)` koja služi za promjenu stanja digitalnog izlaza. Ova makronaredba prima iste argumente kao i dosad korištena makronaredba `set_pin_on(PORTx, Pxi)`. Dakle, makronaredba `toggle_pin(PORTx, Pxi)` kao argumente prima registar `PORTx`

(x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) kojem želimo promijeniti stanje. Makronaredbu `toggle_pin` ćemo koristiti za promjenu stanja trčećih LED dioda.

U beskonačnu `while` petlju programskog koda 4.12 upišite sljedeće naredbe:

- `toggle_pin(PORTB, PB1);` - početno su sve LED diode isključene pa će ova makronaredba uključiti crvenu LED diodu.
- `_delay_ms(250);` - funkcija koja omogućuje vremensko kašnjenje iznosa 250 ms.
- `toggle_pin(PORTB, PB1);` - mijenja se stanje crvene LED diode (isključuje se).
- `toggle_pin(PORTB, PB2);` - mijenja se stanje žute LED diode (uključuje se).
- `_delay_ms(250);` - funkcija koja omogućuje vremensko kašnjenje iznosa 250 ms.
- `toggle_pin(PORTB, PB2);` - mijenja se stanje žute LED diode (isključuje se).
- `toggle_pin(PORTB, PB3);` - mijenja se stanje zelene LED diode (uključuje se).
- `_delay_ms(250);` - funkcija koja omogućuje vremensko kašnjenje iznosa 250 ms.
- `toggle_pin(PORTB, PB3);` - mijenja se stanje zelene LED diode (isključuje se).

Nakon što se sa zadnjom naredbom u `while` petlji isključi zelena LED dioda, petlja se izvodi ponovno i uključuje se crvena LED dioda. Ovaj niz se izvodi dok god mikroupravljač ima napajanje. Konačan program je prikazan programskim kodom 4.15.

Programski kod 4.15: Program za trčanje crvene, žute i zelene LED diode s vremenskim razmakom od 250 ms - korištenje makronaredbe `toggle_pin`

```c
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

    // PB1, PB2 i PB3 konfigurirani kao izlazni pinovi
    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

        toggle_pin(PORTB, PB1); // promijeni stanje pina PB1
        _delay_ms(250); // zakasni 250 ms
        toggle_pin(PORTB, PB1); // promijeni stanje pina PB1
        toggle_pin(PORTB, PB2); // promijeni stanje pina PB2
        _delay_ms(250); // zakasni 250 ms
        toggle_pin(PORTB, PB2); // promijeni stanje pina PB2
        toggle_pin(PORTB, PB3); // promijeni stanje pina PB3
        _delay_ms(250); // zakasni 250 ms
        toggle_pin(PORTB, PB3); // promijeni stanje pina PB3
    }

    return 0;
}
```

Prevedite datoteku `vjezba44.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P crvena, žuta i zelena LED dioda će trčati s vremenskim razmakom od 250 ms.

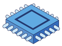
Prethodni program može se napisati na različite načine. U programskom kodu 4.16 dan je primjer u kojem se umjesto makronaredbe `toggle_pin` koriste makronaredbe `set_pin_on` i `set_pin_off`. Ovaj primjer je intuitivniji, jer u svakom koraku vidimo koja je LED dioda isključena, a koja je uključena. Prepravite programski kod 4.15 u skladu s programskim kodom 4.16.

Programski kod 4.16: Program za trčanje crvene, žute i zelene LED diode s vremenskim razmakom od 250 ms - korištenje makronaredbi `set_pin_on` i `set_pin_off`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

    // PB1, PB2 i PB3 konfigurirani kao izlazni pinovi
    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

        set_pin_off(PORTB, PB3); // pin PB3 postavljen u nisko stanje
        set_pin_on(PORTB, PB1); // pin PB1 postavljen u visoko stanje
        _delay_ms(250); // zakasni 250 ms
        set_pin_off(PORTB, PB1); // pin PB1 postavljen u nisko stanje
        set_pin_on(PORTB, PB2); // pin PB2 postavljen u visoko stanje
        _delay_ms(250); // zakasni 250 ms
        set_pin_off(PORTB, PB2); // pin PB2 postavljen u nisko stanje
        set_pin_on(PORTB, PB3); // pin PB3 postavljen u visoko stanje
        _delay_ms(250); // zakasni 250 ms
    }

    return 0;
}
```

Pokušajte samostalno zamijeniti makronaredbe `set_pin_on` i `set_pin_off` u programskom kodu 4.16 s funkcijom `digitalWrite()`. Prevedite datoteku `vjezba44.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba44.cpp` i onemogućite prevođenje ove datoteke.

 **Vježba 4.5**

Napišite program koji će na razvojnom okruženju s mikroupravljačem ATmega328P osigurati beskonačno "trčanje" svih LED dioda desno-lijevo svakih 100 ms. Redoslijed trčanja LED dioda neka je crvena → žuta → zelena → žuta → crvena → ... Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba45.cpp`. Omogućite prevođenje datoteke `vjezba45.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba45.cpp` prikazan je programskim kodom 4.17.

Programski kod 4.17: Početni sadržaj datoteke `vjezba45.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pinovi
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pinovi
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pinovi
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    return 0;
}
```

Funkcionalnost programskog koda koju moramo ostvariti jest „trčanje" LED dioda na način da trči uključena LED dioda desno-lijevo. Ovaj način trčanja LED dioda ostvaruje se na sljedeći način:

- uključite crvenu LED diodu, isključite žutu LED diodu te omogućite vremensko kašnjenje od 100 ms,
- uključite žutu LED diodu, isključite crvenu LED diodu te omogućite vremensko kašnjenje od 100 ms,
- uključite zelenu LED diodu, isključite žutu LED diodu te omogućite vremensko kašnjenje od 100 ms,
- uključite žutu LED diodu, isključite zelenu LED diodu te omogućite vremensko kašnjenje od 100 ms,
- beskonačno ponavljajte prethodnih četiri koraka.

Početni sadržaj datoteke `vjezba45.cpp` prikazan programskim kodom 4.17 sadrži konfiguraciju digitalnih izlaznih pinova u inicijalizacijskoj funkciji `init()`. Korištena je funkcija `pinMode()`. Da bismo ostvarili zadanu funkcionalnost, u programski kod 4.17 upišite sljedeće naredbe:

- `while(1){ }` - nakon funkcije `init()` potrebno je dodati beskonačnu `while` petlju s pripadajućim vitičastim zagradama kako bi se ostvarilo beskonačno trčanje LED dioda

desno-lijevo. Sve sljedeće naredbe upisat ćete unutar `while` petlje.

- `digitalWrite(B2, LOW);` - mijenja se stanje žute LED diode (isključuje se) jer je na kraju svakog ciklusa žuta LED dioda zadnja koja je uključena.
- `digitalWrite(B1, HIGH);` - početno su sve LED diode isključene pa će ova funkcija uključiti crvenu LED diodu.
- `_delay_ms(100);` - funkcija koja omogućuje vremensko kašnjenje iznosa 100 ms.
- `digitalWrite(B1, LOW);` - mijenja se stanje crvene LED diode (isključuje se).
- `digitalWrite(B2, HIGH);` - mijenja se stanje žute LED diode (uključuje se).
- `_delay_ms(100);` - funkcija koja omogućuje vremensko kašnjenje iznosa 100 ms.
- `digitalWrite(B2, LOW);` - mijenja se stanje žute LED diode (isključuje se).
- `digitalWrite(B3, HIGH);` - mijenja se stanje zelene LED diode (uključuje se).
- `_delay_ms(100);` - funkcija koja omogućuje vremensko kašnjenje iznosa 100 ms.
- `digitalWrite(B3, LOW);` - mijenja se stanje zelene LED diode (isključuje se).
- `digitalWrite(B2, HIGH);` - mijenja se stanje žute LED diode (uključuje se).
- `_delay_ms(100);` - funkcija koja omogućuje vremensko kašnjenje iznosa 100 ms.

Konačan program je prikazan programskim kodom 4.18.

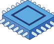
Programski kod 4.18: Program za trčanje crvene, žute i zelene LED diode desno-lijevo s vremenskim razmakom od 100 ms

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pin
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        digitalWrite(B2, LOW); // pin PB2 postavljen u nisko stanje
        digitalWrite(B1, HIGH); // pin PB1 postavljen u visoko stanje
        _delay_ms(100); // zakasni 100 ms
        digitalWrite(B1, LOW); // pin PB1 postavljen u nisko stanje
        digitalWrite(B2, HIGH); // pin PB2 postavljen u visoko stanje
        _delay_ms(100); // zakasni 100 ms
        digitalWrite(B2, LOW); // pin PB2 postavljen u nisko stanje
        digitalWrite(B3, HIGH); // pin PB3 postavljen u visoko stanje
        _delay_ms(100); // zakasni 100 ms
        digitalWrite(B3, LOW); // pin PB3 postavljen u nisko stanje
```

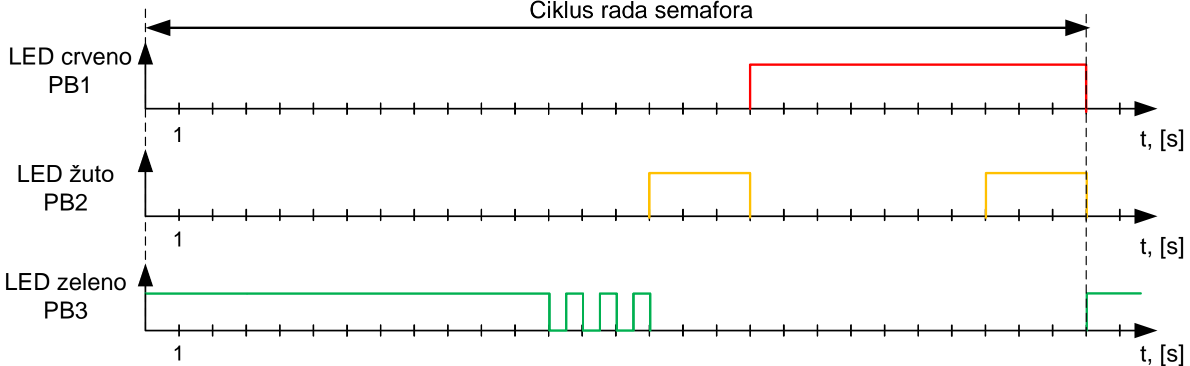
```
        digitalWrite(B2, HIGH); // pin PB2 postavljen u visoko stanje
        _delay_ms(100); // zakasni 100 ms
    }

    return 0;
}
```

Prevedite datoteku `vjezba45.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P crvena, žuta i zelena LED dioda će trčati desno-lijevo s vremenskim razmakom od 100 ms.

Zatvorite datoteku `vjezba45.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 4.6

Napišite program koji će na razvojnom okruženju s mikroupravljačem ATmega328P simulirati rad semafora prema vremenskom dijagramu na slici 4.3.



Slika 4.3: Vremenski dijagram rada zelenog, žutog i crvenog svjetla na semaforu

Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba46.cpp`. Omogućite prevođenje datoteke `vjezba46.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba46.cpp` prikazan je programskim kodom 4.19.

Programski kod 4.19: Početni sadržaj datoteke `vjezba46.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

#define CRVENO B1 // alias za crveno svjetlo
#define ZUTO B2 // alias za žuto svjetlo
#define ZELENO B3 // alias za zeleno svjetlo

void init() {

    pinMode(CRVENO, OUTPUT); // PB1 konfiguriran kao izlazni pinovi
    pinMode(ZUTO, OUTPUT); // PB2 konfiguriran kao izlazni pinovi
    pinMode(ZELENO, OUTPUT); // PB3 konfiguriran kao izlazni pinovi
```

```
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

    }

    return 0;
}
```

Pokazat ćemo nekoliko pristupa pri realizaciji semafora sa zadanim vremenskim dijagramom na slici 4.3. Prema vremenskom dijagramu na slici 4.3, jedan ciklus rada semafora traje 28 s i on se periodički ponavlja. Dakle, naš zadatak je napraviti program za jedan ciklus rada semafora koji će se ponavljati u beskonačnoj petlji. Analizom vremenskog dijagrama rada semafora dolazimo do sljedećeg:

- zeleno svijetlo radi 12 s od početka ciklusa,
- nakon 12 s zeleno svjetlo tijekom 3 s mijenja stanja svakih 500 ms,
- nakon 15 s žuto svjetlo radi 3 s,
- nakon 18 s crveno svjetlo radi 10 s,
- nakon 25 s žuto svijetlo radi 3 s (u tih 3 s uključeno je i crveno svjetlo).

Početni sadržaj datoteke `vjezba46.cpp` sadrži aliase (simbolička imena, definirane konstante) za pinove na kojima su spojene crvena, žuta i zelena LED dioda. Na primjer, naredba `#define CRVENO B1` omogućuje da se konfiguracija pina PB1 i postavljanje pina PB1 u visoko ili nisko stanja obavlja pomoću konstante `CRVENO` umjesto dosada korištene `B1`. Na taj način možete stvoriti simbolička imena za različite senzore i aktuatore i pristupati im na intuitivan način. U programskom kodu 4.19 konfigurirani su izlazni pinovi s definiranim konstantama `CRVENO`, `ZUTO` i `ZELENO`.

Ciklus rada semafora potrebno je realizirati u beskonačnoj petlji programskog koda 4.19. Da bi zeleno svijetlo bilo uključeno 12 s od početka ciklusa, u `while` petlju upišite sljedeće naredbe:

- `digitalWrite(ZELENO, HIGH);` - na početku ciklusa uključuje se zelena LED dioda.
- `_delay_ms(12000);` - funkcija koja omogućuje vremensko kašnjenje od 12000 ms kako bi zelena LED dioda bila uključena 12 s.

Nakon 12 s ciklusa rada semafora, zeleno svjetlo mora 3 sekunde isprekidano svijetliti tako da je 500 ms zeleno svjetlo isključeno, a 500 ms uključeno. Nakon prethodno unesenih naredbi unesite sljedeće naredbe:

- `digitalWrite(ZELENO, LOW);` - zelena LED je isključena 500 ms.
- `_delay_ms(500);` - funkcija koja omogućuje vremensko kašnjenje od 500 ms kako bi zelena LED dioda bila isključena 500 ms.
- `digitalWrite(ZELENO, HIGH);` - zelena LED je uključena 500 ms.
- `_delay_ms(500);` - funkcija koja omogućuje vremensko kašnjenje od 500 ms kako bi zelena LED dioda bila uključena 500 ms.

- ponovite prethode naredbe još dva puta (kopirajte ih i dva puta zalijepite jednu ispod druge).

Nakon 15 s ciklusa rada semafora, uključuje se crveno svjetlo na 10 s, a prethodno uključeno zeleno svjetlo je potrebno isključiti. Nakon 7 s rada crvenog svjetla, uključuje se žuto svjetlo koje radi 3 s. Za ovu funkcionalnost unesite sljedeće naredbe u beskonačnu petlju:

- `digitalWrite(ZELENO, LOW);` - zelena LED dioda je isključena do početka ciklusa.
- `digitalWrite(CRVENO, HIGH);` - uključena je crvena LED dioda koja mora raditi 10 s.
- `_delay_ms(7000);` - funkcija koja omogućuje vremensko kašnjenje od 7000 ms kako bi se nakon 7 s rada crvene LED diode mogla uključiti žuta na 3 s.
- `digitalWrite(ZUTO, HIGH);` - uključena je žuta LED dioda koja mora raditi 3 s.
- `_delay_ms(3000);` - funkcija koja omogućuje vremensko kašnjenje od 3000 ms kako bi crvena LED dioda radila ukupno 10 s, a žuta 3 s.

Nakon što prođe jedan ciklus rada semafora koji traje 28 s, na kraju je potrebno isključiti crveno i žuto svjetlo, a uključiti zeleno. Crveno i žuto svjetlo uključit ćemo sljedećim naredbama:

- `digitalWrite(ZUTO, LOW);` - isključena je žuta LED dioda.
- `digitalWrite(CRVENO, LOW);` - isključena je crvena LED dioda.

Zeleno svjetlo na semaforu uključit će se na početku novog ciklusa što smo ostvarili na samom početku beskonačne petlje. Konačan program je prikazan programskim kodom 4.20.

Programski kod 4.20: Upravljanje semaforom prema vremenskom dijagramu na slici 4.3 - prvi način

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

#define CRVENO B1 // alias za crveno svjetlo
#define ZUTO B2 // alias za žuto svjetlo
#define ZELENO B3 // alias za zeleno svjetlo

void init() {

    pinMode(CRVENO, OUTPUT); // PB1 konfiguriran kao izlazni pino
    pinMode(ZUTO, OUTPUT); // PB2 konfiguriran kao izlazni pino
    pinMode(ZELENO, OUTPUT); // PB3 konfiguriran kao izlazni pino
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        digitalWrite(ZELENO, HIGH); // uključeno zeleno svjetlo na 12 s
        _delay_ms(12000); // čekanje 12 s

        // izmjenjivanje stanja zelenog svjetla svakih 500 ms
        digitalWrite(ZELENO, LOW);
        _delay_ms(500); // čekanje 500 ms
        digitalWrite(ZELENO, HIGH);
        _delay_ms(500); // čekanje 500 ms
```

```
        digitalWrite(ZELENO, LOW);
        _delay_ms(500); // čekanje 500 ms
        digitalWrite(ZELENO, HIGH);
        _delay_ms(500); // čekanje 500 ms
        digitalWrite(ZELENO, LOW);
        _delay_ms(500); // čekanje 500 ms
        digitalWrite(ZELENO, HIGH);
        _delay_ms(500); // čekanje 500 ms

        digitalWrite(ZELENO, LOW); // isključeno zeleno svjetlo
        digitalWrite(ZUTO, HIGH); // uključeno žuto svjetlo na 3 s
        _delay_ms(3000); // čekanje 3 s

        digitalWrite(ZUTO, LOW); // isključeno žuto svjetlo
        digitalWrite(CRVENO, HIGH); // uključeno crveno svjetlo na 10 s
        _delay_ms(7000); // čekanje 7 s
        digitalWrite(ZUTO, HIGH); // uključeno žuto svjetlo na 3 s
        _delay_ms(3000); // čekanje 3 s

        digitalWrite(ZUTO, LOW); // isključeno žuto svjetlo
        digitalWrite(CRVENO, LOW); // isključeno crveno svjetlo
    }

    return 0;
}
```

Prevedite datoteku `vjezba46.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P crvena, žuta i zelena LED simulirat će rad semafora prema vremenskom dijagramu na slici 4.3.

Napisani programski kod radit će ispravno ako je to jedina funkcionalnost koju je potrebno realizirati mikroupravljačem. No, ako je paralelno potrebno obrađivati dodatne senzore i upravljati dodatnim aktuatorima, tada će nastati problem jer će to biti moguće samo na kraju ciklusa, odnosno svakih 28 s.

Ovaj problem može se riješiti na način da se uvede varijabla kojom ćemo mjeriti broj prolazaka kroz beskonačnu petlju, a same prolaske kroz petlju ćemo ubrzati (umjesto da imamo jedan prolaz kroz petlju trajanja 28 s, imat ćemo 280 prolaza kroz petlju trajanja 100 ms). U ovom pristupu ćemo definirati vremenske intervale u kojima svijetli pojedino svjetlo na sljedeći način:

- zeleno svjetlo svijetli u intervalima [0, 12] s, [12.5, 13] s, [13.5, 14] s i [14.5, 15] s,
- žuto svjetlo svijetli u intervalima [15, 18] s i [25, 28] s,
- crveno svjetlo svijetli u intervalu [18, 28] s.

Jedan od načina kako možemo upravljati svjetlima u zadanim intervalima jest da svjetlo uključimo na početku intervala, a isključimo na kraju intervala. Sukladno navedeno, definirat ćemo uvjete za uključenje i isključenje svjetala:

- zelena LED dioda uključuje se kada je vrijeme ciklusa jednako 0 s, 12.5 s, 13.5 s i 14.5 s,
- zelena LED dioda isključuje se kada je vrijeme ciklusa jednako 12 s, 13 s, 14 s i 15 s,
- žuta LED dioda uključuje se kada je vrijeme ciklusa jednako 15 s, 25 s
- žuta LED dioda isključuje se kada je vrijeme ciklusa jednako 18 s, 28 s = 0 s,
- crvena LED dioda uključuje se kada je vrijeme ciklusa jednako 18 s,

- crvena LED dioda isključuje se kada je vrijeme ciklusa jednako 28 s = 0 s.

Uvjeti su ostvareni pomoću bloka za uvjetovanje `if`. Modificirajte programski kod u datoteci `vjezba46.cpp` sukladno programskom kodu 4.21.

Programski kod 4.21: Upravljanje semaforom prema vremenskom dijagramu na slici 4.3 - drugi način

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

#define CRVENO B1 // alias za crveno svjetlo
#define ZUTO B2 // alias za žuto svjetlo
#define ZELENO B3 // alias za zeleno svjetlo

void init() {

    pinMode(CRVENO, OUTPUT); // PB1 konfiguriran kao izlazni pin
    pinMode(ZUTO, OUTPUT); // PB2 konfiguriran kao izlazni pin
    pinMode(ZELENO, OUTPUT); // PB3 konfiguriran kao izlazni pin
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    int brojacCiklusa = 0;

    while (1) { // beskonačna petlja

        // zelena LED dioda se uključuje u: 0 s, 12.5 s, 13.5 s i 14.5 s
        if (brojacCiklusa == 0 || brojacCiklusa == 125
        || brojacCiklusa == 135 || brojacCiklusa == 145) {
            digitalWrite(ZELENO, HIGH);
        }
        // zelena LED dioda se isključuje u: 12 s, 13 s, 14 s i 15 s
        if (brojacCiklusa == 120 || brojacCiklusa == 130
        || brojacCiklusa == 140 || brojacCiklusa == 150) {
            digitalWrite(ZELENO, LOW);
        }
        // zuta LED dioda se uključuje u: 15 s, 25 s
        if (brojacCiklusa == 150 || brojacCiklusa == 250) {
            digitalWrite(ZUTO, HIGH);
        }
        // zuta LED dioda se isključuje u: 18 s, 28 s = 0 s
        if (brojacCiklusa == 180 || brojacCiklusa == 0) {
            digitalWrite(ZUTO, LOW);
        }
        // crvena LED dioda se uključuje u: 18 s
        if (brojacCiklusa == 180) {
            digitalWrite(CRVENO, HIGH);
        }
        // crvena LED dioda se isključuje u: 28 s = 0 s
        if (brojacCiklusa == 0) {
            digitalWrite(CRVENO, LOW);
        }
        _delay_ms(100); // vrijeme jednog prolaza kroz petlju
        brojacCiklusa++;

        // 280 x 100 ms = 28 s
        // kada brojac dode do 28 s, ciklus se resetira na 0 s
        if (brojacCiklusa == 280) {
```

```
                brojacCiklusa = 0;
            }
        }

    return 0;
}
```

Prevedite datoteku `vjezba46.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

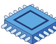
Bitna razlika između programskih kodova 4.20 i 4.21 je ta da prolaz kroz beskonačnu petlju u programskom kodu 4.20 traje 28 s, a u programskom kodu 4.21 samo 100 ms. Na ovaj način nismo napravili zaglavljenje programa na duže vrijeme i moguće je uz semafor upravljati i drugim uređajima.

Osvrnimo se još jednom na simboličke adrese. U programskim kodovima 4.20 i 4.21 koristili smo funkcije za konfiguraciju digitalnih izlaza i za postavljanje stanja na digitalne izlaze. Kod funkcijskog pristupa, koji zauzima više programske memorije, potrebno je stvoriti samo jedan alias po digitalnom pinu. Da smo koristili makronaredbe za konfiguraciju digitalnih izlaza i postavljanje stanja digitalnih izlaza ili da samo konfiguraciju i postavljanje stanja digitalnih izlaza radili direktno pomoću registara, za svaki digitalni pin trebali bismo napraviti tri aliasa. U programskom kodu 4.22 prikazano je kreiranje aliasa za crveno svjetlo pri korištenju makronaredbi za konfiguraciju digitalnih izlaza i postavljanje stanja digitalnih izlaza. Makronaredbe `config_output()` i `set_pin_on()` kao argumente koriste `DDRx` registar, `PORTx` registar i poziciju pina `Pxi`. Iz tog razloga je potrebno raditi alias za svaki od navedenih argumenata makronaredbi. Jednom kada se stvore aliasi, konfiguracija digitalnog pina i postavljanje izlaznog pina u visoko stanje jednostavni su (vidi `init()` funkciju u programskom kodu 4.22).

Programski kod 4.22: Kreiranje aliasa za crveno svjetlo pri korištenju makronaredbi za konfiguraciju digitalnih izlaza i postavljanje stanja digitalnih izlaza

```
// aliasi za crveno svjetlo
#define CRVENO_DDR DDRB
#define CRVENO_PORT PORTB
#define CRVENO_PIN PB1


void init() {

    config_output(CRVENO_DDR, CRVENO_PIN);
    set_pin_on(CRVENO_PORT, CRVENO_PIN);

}
```

Zatvorite datoteku `vjezba46.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 4.7

Napišite program koji će na razvojnom okruženju s mikroupravljačem ATmega328P izmjenjivati stanja crvene LED diode i zujalice svakih 500 ms. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, a zujalica je spojena na digitalni pin PD1 mikroupravljača ATmega328P. Kratkospojnik JP5 potrebno je spojiti između trnova 1 i 2.

U projektnom stablu otvorite datoteku `vjezba47.cpp`. Omogućite prevođenje datoteke `vjezba47.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba47.cpp` prikazan je programskim kodom 4.23.

Programski kod 4.23: Početni sadržaj datoteke vjezba47.cpp

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

// aliasi za crveno svjetlo
#define CRVENO_DDR DDRB
#define CRVENO_PORT PORTB
#define CRVENO_PIN PB1

void init() {
    //PB1 konfiguriran kao izlazni pin
    config_output(CRVENO_DDR, CRVENO_PIN);

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

    }

    return 0;
}
```

U ovoj vježbi koristit ćemo zujalicu u svrhu zvučne signalizacije. Zujalica je spojena na digitalni izlaz PD1. Na shemi sa slike 4.1 primijetite da je zujalica na pin PD1 mikroupravljača ATmega328P spojena preko kratkospojnika (engl. *jumper*) `JP5` između trnova 1 i 2. Ako na razvojnom okruženju s mikroupravljačem ATmega328P nema kratkospojnika `JP5`, zujalica neće raditi.

Funkcionalnost programskog koda koju moramo ostvariti jest izmjena stanja crvene LED diode i zujalice svakih 500 ms. Ovaj način rada ostvaruje se na sljedeći način:

- uključite crvenu LED diodu, isključite zujalicu te omogućite vremensko kašnjenje od 500 ms,
- isključite crvenu LED diodu, uključite zujalicu te omogućite vremensko kašnjenje od 500 ms,
- beskonačno ponavljajte prethodnih dva koraka.

U programskom kodu 4.23 stvoreni su sljedeći aliasi:

- `#define CRVENO_DDR DDRB` - registar smjera podataka koji omogućuje da se digitalni pin PB1 definira kao izlazni pin,
- `#define CRVENO_PORT PORTB` - podatkovni registar koji omogućuje da se digitalni pin PB1 postavi u visoko ili nisko stanje,
- `#define CRVENO_PIN PB1` - pozicija digitalnog pina PB1.

Zujalice se u praksi koriste kako bi se ukazalo na neki događaj. Za ostvarenje zvučne signalizacije, koristit ćemo funkciju `buzz(uint64_t trajanje, int frekvencija)` koja prima dva argumenta:

1. `trajanje` - cijeli broj koji predstavlja trajanje zvučnog signala u milisekundama,

2. `frekvencija` - cijeli broj koji predstavlja frekvenciju zvučnog signala u Hz.

Deklaracija funkcije `buzz()` nalazi se u datoteci `avrlib.h`. Iznad deklaracije funkcije `buzz()` u datoteci `avrlib.h` nalaze se aliasi za zujalicu:

- `#define BUZZER_DDR DDRD` - makronaredba kojom se definira registar smjera podataka za zujalicu koja je spojena na pin PD1,
- `#define BUZZER_PORT PORTD` - makronaredba kojom se definira podatkovni registar za zujalicu koja je spojena na pin PD1,
- `#define BUZZER_PIN PD1` - makronaredba kojom se definira pozicija digitalnog pina PD1.

Ako je zujalica spojena na pin PC4 tada biste modificirali aliase za zujalicu na sljedeći način:

- `#define BUZZER_DDR DDRC` - makronaredba kojom se definira registar smjera podataka za zujalicu koja je spojena na pin PC4,
- `#define BUZZER_PORT PORTC` - makronaredba kojom se definira podatkovni registar za zujalicu koja je spojena na pin PC4,
- `#define BUZZER_PIN PC4` - makronaredba kojom se definira pozicija digitalnog pina PC4.

Sada ćemo u programskom kodu umjesto registara koristiti aliase (na primjer: umjesto registra `DDRD` za konfiguraciju izlaza koristiti ćemo alias `BUZZER_DDR`). Inicijalizacijska funkcija `init()` u programskom kodu 4.23 sadrži konfiguraciju izlaznog pina PB1 na kojem je spojena crvena LED dioda. Sada ćemo napraviti konfiguraciju izlaznog pina na kojem je spojena zujalica. U programski kod 4.23 u inicijalizacijsku funkciju `init()` upišite makronaredbu: `config_output(BUZZER_DDR, BUZZER_PIN);`.

Izmjenu stanja crvenog svijetla i zujalice svakih 500 ms potrebno je realizirati u beskonačnoj petlji programskog koda 4.23. Da bismo to ostvarili, u `while` petlju upišite sljedeće naredbe:

- `set_pin_on(CRVENO_PORT, CRVENO_PIN);` - na početku ciklusa uključuje se crvena LED dioda.
- `_delay_ms(500);` - funkcija koja omogućuje vremensko kašnjenje od 500 ms.
- `set_pin_off(CRVENO_PORT, CRVENO_PIN);` - nakon 500 ms isključuje se crvena LED dioda.
- `buzz(500, 660);` - zujalica proizvodi zvučni signal frkvencije 660 Hz, trajanja 500 ms. Ova funkcija će zadržati izvršenje programskog koda 500 ms pa nije potrebno dodatno kašnjenje.

Konačan program je prikazan programskim kodom 4.24.

Programski kod 4.24: Izmjena stanja crvene LED diode i zujalice svakih 500 ms

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include <util/delay.h>

// aliasi za crveno svjetlo
#define CRVENO_DDR DDRB
#define CRVENO_PORT PORTB
#define CRVENO_PIN PB1

void init() {
```

```
    //PB1 konfiguriran kao izlazni pin
    config_output(CRVENO_DDR, CRVENO_PIN);
    //PD1 konfiguriran kao izlazni pin
    config_output(BUZZER_DDR, BUZZER_PIN);
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) { // beskonacna petlja

        set_pin_on(CRVENO_PORT, CRVENO_PIN); // ukljuceno crveno svijetlo
        _delay_ms(500); // vremensko kašnjenje od 500 ms

        set_pin_off(CRVENO_PORT, CRVENO_PIN); // iskljuceno crveno svijetlo
        buzz(500, 660); // zujalica radi 500 ms

    }

    return 0;
}
```

Prevedite datoteku `vjezba47.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste sve korake napravili ispravno, na razvojnom okruženju s mikroupravljačem ATmega328P crvena LED dioda i zujalica izmjenjivat će stanja svakih 500 ms. Pokušajte testirati različite frekvencije zvučnog signala na zujalici, kao i različita trajanja signala.

Zatvorite datoteku `vjezba47.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 5

Digitalni ulazi mikroupravljača ATmega328P

Kako smo naveli u prethodnom poglavlju, digitalni pinovi mikroupravljača ATmega328P mogu se konfigurirati kao izlazni pinovi ili kao ulazni pinovi. U ovom poglavlju naglasak će biti na digitalne ulaze na koje uobičajeno spajamo senzore poput tipkala i senzora prisutnosti, ali i druge uređaje koji informacije šalju putem vlastitih digitalnih izlaza.

Već smo spomenuli da mikroupravljač ATmega328P ima 23 ulazno/izlazna pina opće namjene koji su smješteni u tri grupe koje nazivamo portovima:

- **PORTB** (pinovi: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- **PORTC** (pinovi: PC0, PC1, PC2, PC3, PC4, PC5, PC6) i,
- **PORTD** (pinovi: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7).

Konfiguracija digitalnog ulaza ili izlaza provodi se pomoću registra `DDRx`. Ako je bit na poziciji i u registru `DDRx` jednak 0, tada će pin na poziciji i biti konfiguriran kao ulazni pin. Konfiguraciju porta D sa sadržajem registra **DDRD** = 0xAA prikazali smo u tablici 4.1 u prethodnom poglavlju. Ulazni pin može biti u dva stanja:

- stanje koje odgovara naponskom nivou 0 V ili stanje logičke nule (nisko stanje),
- stanje koje odgovara naponskom nivou `VCC`[1] V ili stanje logičke jedinice (visoko stanje).

Navedena stanja digitalnog ulaza mogu se pročitati u `PINx` registru prema pravilima:

- ako je na digitalnom ulaznom pinu na poziciji i nisko stanje (stanje logičke nule), tada će bit na poziciji i u registru `PINx` biti jednak 0,
- ako je na digitalnom ulaznom pinu na poziciji i visoko stanje (stanje logičke jedinice), tada će bit na poziciji i u registru `PINx` biti jednak 1.

Stanje ulaznih pinova na portu B ako su su svi pinovi konfigurirani kao ulazni može se pročitati u registru `PINB` kako je prikazano tablicom 5.1.

[1]VCC može biti 5 V, 3.3 V ili neka druga razina što ovisi o napajanju mikroupravljača

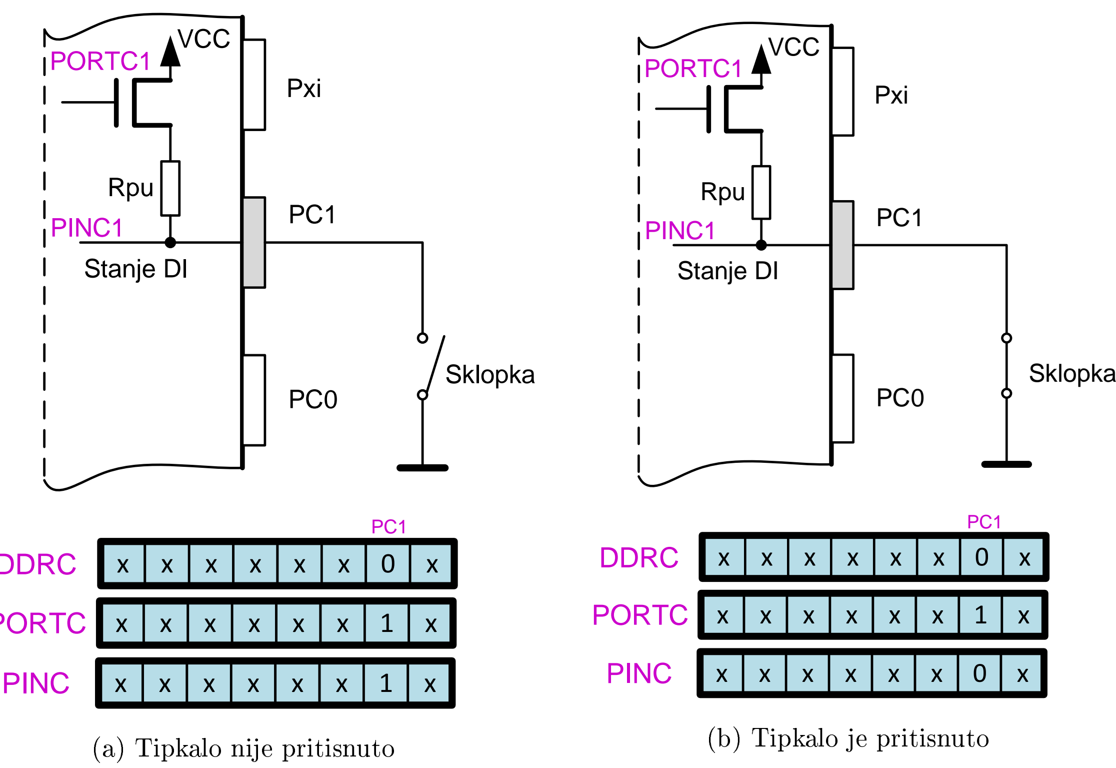
Tablica 5.1: Stanje ulaznih pinova na portu B ako su su svi pinovi konfigurirani kao ulazni (`DDRB = 0x00`)

PINB registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **PINB** registra	1	0	1	1	0	0	0	1
Port B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Stanje ulaznog pina	visoko	nisko	visoko	visoko	nisko	nisko	nisko	visoko

Ako je digitalni pin konfiguriran kao ulazni pin, tada se pomoću registra `PORTx` može isključiti ili uključiti pritezni otpornik (engl. *pull-up resistor*) prema pravilima:

- pritezni otpornik na poziciji pina i isključen je ako je bit na poziciji i u registru `PORTx` jednak 0,
- pritezni otpornik na poziciji pina i uključen je ako je bit na poziciji i u registru `PORTx` jednak 1.

Svrhu uključenja priteznog otpornika prikazat ćemo pomoću slike 5.1 na digitalnom pinu `PC1`.



(a) Tipkalo nije pritisnuto

(b) Tipkalo je pritisnuto

Slika 5.1: Tipkalo spojeno na ulazni pin `PC1` mikroupravljača ATmega328P

Pretpostavimo u prvom slučaju da pritezni otpornik na pinu `PC1` nije uključen. Kada je sklopka uključena (slika 5.1b), potencijal na pinu `PC1` je 0 V (stanje je nisko). Što će se desiti kada se sklopka isključi (slika 5.1a)? Koliki je sada potencijal pina `PC1`? Odgovor na ovo pitanje nije jednoznačan jer je pin u stanju visoke impendancije. Bilo kakva vanjska smetnja mogla bi pin `PC1` dovesti ili u visoko ili u nisko stanje. Uključenjem priteznog otpornika, digitalni pin će poprimiti stabilno stanje i kod isključene sklopke (slika 5.1a). U ovom slučaju, s uključenim priteznim otpornikom i isključenom sklopkom, potencijal pina `PC1` biti će `VCC` V (visoko stanje).

Primjer isključenih i uključenih priteznih otpornika na portu B kada su svi pinovi konfigurirani

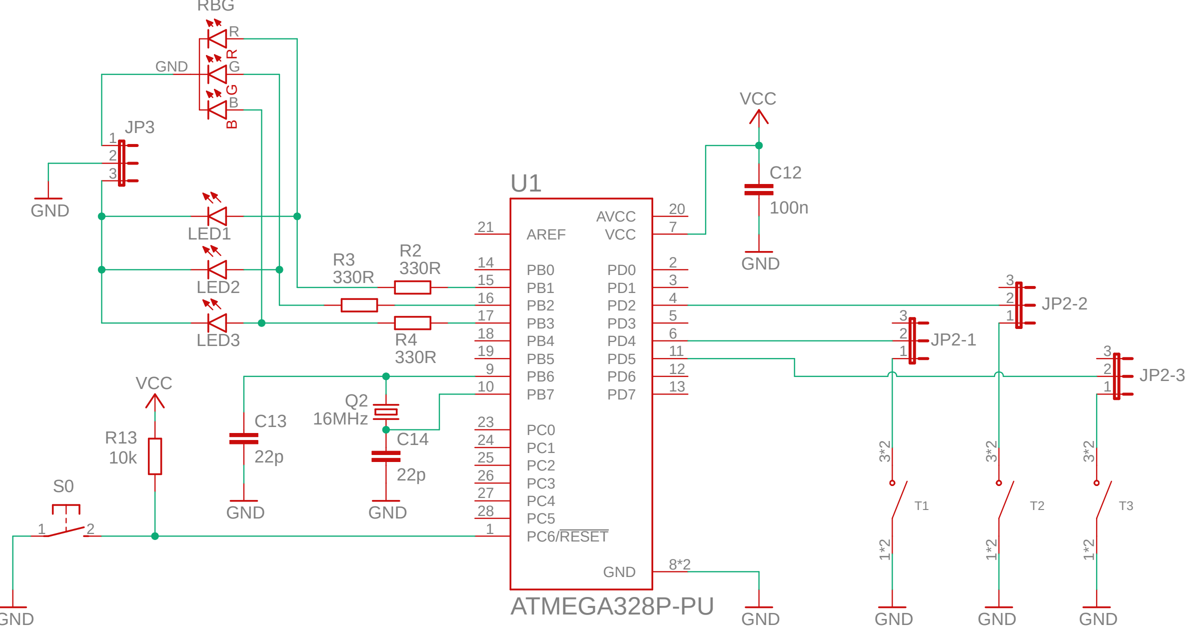
kao ulazni prikazan je tablicom 5.2.

Tablica 5.2: Primjer isključenih i uključenih priteznih otpornika na portu B kada su svi pinovi konfigurirani kao ulazni (`DDRB = 0x00`). U tablici stanje ON predstavlja uključen, a stanje OFF isključen pritezni otpornik.

PORTB registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **PORTB** registra	1	1	1	1	0	0	0	0
Port B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Pritezni otpornik	ON	ON	ON	ON	OFF	OFF	OFF	OFF

Pritezni otpornici moraju se uključiti ako se koriste senzori koji imaju samo jedno stabilno izlazno stanje (tipkalo, *open collector* senzori i drugi). U slučaju korištenja senzora koji imaju dva stabilna izlazna stanja (nisko i visoko), pritezni otpornik na mikroupravljaču nije potrebno uključivati. Pritezni otpornik može se na mikroupravljač spojiti izvana, a njegova je vrijednost otpora najčešće 10 kΩ.

5.1 Vježbe - digitalni ulazi mikroupravljača ATmega328P



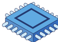
Slika 5.2: Shema spajanja tri tipkala na mikroupravljač ATmega328P

Digitalne ulaze mikroupravljača ATmega328P testirat ćemo pomoću tri tipkala spojenih na pinove PD2, PD4 i PD5. Shema tri spojena tipkala na digitalne ulaze mikroupravljača ATmega328P prikazana je na slici 5.2. Na istoj shemi prikazane su i LED diode koje smo koristili u prošloj vježbi. Shema sa slike 5.2 usklađena je s razvojnim okruženjem prikazanim na slici 2.1.

Prema shemi na slici 5.2 tipkalo T1 spojeno je na pin PD4, tipkalo T2 spojeno je na pin PD2, a tipkalo T3 spojeno je na pin PD5. Za korištenje tipkala T1, T2 i T3 trostruki kratkospojnik JP2 potrebno je spojiti između trnova 1 (`TIPK`) i 2. Navedeni ulazni pinovi koriste se i za rotacijski enkoder koji ćemo obraditi u budućim vježbama (tada se kratkospojnik JP2

spaja između trnova 2 i 3 (ENK)). Za detalje oko konfiguracije digitalnih ulaza mikroupravljača ATmega328P pogledajte tablicu 13-1 u literaturi [2] na stranici 60.

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `Digitalni ulazi.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `Digitalni ulazi.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\Digitalni ulazi\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Digitalni ulazi mikroupravljača ATmega328P`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.



Vježba 5.1

Napišite program koji će uključiti crvenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P ako je pritisnuto tipkalo T1. Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, a tipkalo T1 na digitalni ulaz PD4 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba51.cpp`. Omogućite prevođenje datoteke `vjezba51.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba51.cpp` prikazan je programskim kodom 5.1.

Programski kod 5.1: Početni sadržaj datoteke `vjezba51.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Problem koji moramo riješiti jest napisati program koji će uključiti crvenu LED diodu koja je spojena na digitalni pin PB1 mikroupravljača ATmega328P ako je pritisnuto tipkalo T1 koje je spojeno na digitalni pin PD4. Kako smo već naučili u prethodnom poglavlju, digitalni pin na koji je spojena LED dioda konfigurirat ćemo kao izlazni pin. Stanje tipkala moguće je pročitati ako digitalni pin konfiguriramo kao ulazni pin. Kako smo naveli u uvodu ovog poglavlja, ulazni pin mikroupravljača konfigura se tako da se u DDRD registru bit na poziciji pina PD4 postavi u 0. Inače, svi registri mikroupravljača su početno u stanju 0x00 pa su početno svi digitalni pinovi konfigurirani kao ulazni. Bez obzira na to, mi ćemo provesti postupak konfiguracije digitalnih ulaza kako bi iz programskog koda bilo jasno koja je svrha pojedinog digitalnog pina i koji se digitalni pinovi koriste.

Konfiguracija porta D za ulazni pin PD4, uz pretpostavku da drugi pinovi imaju nepoznatu

namjenu, prikazan je tablicom 5.3. Ako želimo neki bit postaviti u vrijednost 0, tada je potrebno napraviti bitovnu I operaciju s konstantom koja na poziciji bita koji konfigurira ulazni pin ima vrijednost 0, a na svim ostalim mjestima vrijednost 1. To ćemo postići tako da najprije kreiramo konstantu `(1 << PD4)` koja će imati vrijednost 1 na poziciji bita koji konfigurira ulaz, a na svim ostalim mjestima vrijednost 0. Kada takvu konstantu komplementiramo (`~(1 << PD4)`), dobit ćemo konstantu koja ima vrijednost 0 na poziciji bita koji konfigurira ulaz, a na svim ostalim mjestima vrijednost 1. Korištenjem bitovne I operacije, tada se samo bit na željenoj poziciji postavlja u vrijednost 0, a ostali bitovi ne mijenjaju prethodnu vrijednost. Prema tablica 5.3 u vrijednost 0 postavlja se bit 4 koji je povezan s digitalnim pinom PD4. Za ulazne bitove potrebno je uključiti pritezni otpornik kako smo opisali u uvodu ovog poglavlja. To ćemo napraviti tako da u registru `PORTx` bit na poziciji pina i postavimo u vrijednost 1. Pritezni otpornik na digitalnom ulazu PD4 možemo uključiti naredbom `PORTB |= 1 << PD4;`.

Tablica 5.3: Konfiguracija porta D za ulazni pin PD4 uz pretpostavku da drugi pinovi imaju nepoznatu namjenu

DDRD registar	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Sadržaj **DDRD** registra	x	x	x	x	x	x	x	x
1 « **PD4**	0	0	0	1	0	0	0	0
~(1 « **PD4**)	1	1	1	0	1	1	1	1
DDRD &= ~(1 « **PD4**)	x	x	x	0	x	x	x	x
Port D	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
Konfiguracija pina	x	x	x	ulaz	x	x	x	x

Konfiguraciju digitalnih pinova PB1 i PD4 provest ćemo tako da programski kod 5.1 modificiramo na sljedeći način:

- upišite u funkciju `init()` naredbu `DDRB |= (1 << PB1);` (ova naredba će pin PB1 konfigurirati kao izlazni pin),
- upišite u funkciju `init()` naredbu `DDRD &= ~(1 << PD4);` (ova naredba će pin PD4 konfigurirati kao ulazni pin),
- upišite u funkciju `init()` naredbu `PORTD |= (1 << PD4);` (ova naredba će uključiti pritezni otpornik na pinu PD4).

Tijelo inicijalizacijske funkcije `init()` mora odgovarati programskom kodu 5.2.

Programski kod 5.2: Sadržaj datoteke `vjezba51.cpp` nakon konfiguracije digitalnih pinova PB1 i PD4

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    DDRB |= (1 << PB1); // PB1 konfiguriran kao izlaz
    DDRD &= ~(1 << PD4); // PD4 konfiguriran kao ulaz

    PORTD |= (1 << PD4); // ukljucen pull up otpornik na ulazu PD4
}

int main(void) {
```

```
    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

U bloku naredbi `while` petlje potrebno je implementirati programski kod koji uključuje ili isključuje crvenu LED diodu ovisno o stanju digitalnog ulaza PD4. Stanje digitalnog ulaza čita se pomoću `PINx` registra.

U tablici 5.4 prikazan je slučaj u kojem je pritisnuto tipkalo spojeno na pin PD4. Ako tipkalo nije pritisnuto, na pinu PD4 nisko je stanje, a vrijednost bita 4 u registru `PIND` jest 0. Konstanta `1 << PD4` koja se koristi za ispitivanje stanja bita 4 formira se tako da se na poziciju bita 4 postavi 1, a na poziciju ostalih bitova postavi se 0 (tablica 5.4). Kada tipkalo nije pritisnuto, na pinu PD4 visoko je stanje, a vrijednost bita 4 u registru `PIND` jest 1. Ovaj slučaj prikazan je u tablici 5.5.

Stanje pina PD4 u programskom jeziku C ili C++ ispituje se naredbom `if((PIND & (1 << PD4))== 0x00)`. Rezultat bitovne operacije `PIND & (1 << PD4)` bit će jednak 0x00 ako je tipkalo pritisnuto (tablica 5.4). Ako tipkalo nije pritisnuto, rezultat bitovne operacije `PIND & (1 << PD4)` bit će jednak 0x10 (tablica 5.5).

Tablica 5.4: Ispitivanje stanja digitalnog ulaza PD4 - slučaj u kojem je pritisnuto tipkalo

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
PIND	x	x	x	0	x	x	x	x
1 « PD4	0	0	0	1	0	0	0	0
PIND & (1 « PD4)	0	0	0	0	0	0	0	0

Tablica 5.5: Ispitivanje stanja digitalnog ulaza PD4 - slučaj u kojem nije pritisnuto tipkalo

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
PIND	x	x	x	1	x	x	x	x
1 « PD4	0	0	0	1	0	0	0	0
PIND & (1 « PD4)	0	0	0	1	0	0	0	0

Općenito, stanje se ulaznog digitalnog pina na poziciji i te na portu D ispituje naredbom `if((PIND & (1 << PDi))== 0x00)`. Izraz `(1 << PDi)` formira konstantu u kojoj se na poziciju bita i nalazi vrijednost 1, a na poziciju ostalih bitova 0. Isti je princip i za port B ili C gdje je registar `PIND` potrebno zamijeniti registrom `PINB` ili `PINC`.

Programski kod 5.3 prikazuje prošireni programski kod 5.2 s uvjetovanim blokom naredbi za uključenje i isključenje crvene LED diode u ovisnosti o pritisnutom tipkalu. Ako je tipkalo pritisnuto (`if((PIND & (1 << PD4))== 0x00)`), tada će se uključiti crvena LED dioda (`PORTB |= (1 << PB1);`). U suprotnom će crvena LED dioda biti isključena (`PORTB &= ~(1 << PB1);`).

Programski kod 5.3: Sadržaj datoteke `vjezba51.cpp` - crvena LED dioda uključena je ako je pritisnuto tipkalo T1 (prvi način)

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    DDRB |= (1 << PB1); // PB1 konfiguriran kao izlaz
    DDRD &= ~(1 << PD4); // PD4 konfiguriran kao ulaz

    PORTD |= (1 << PD4); // ukljucen pull up otpornik na ulazu PD4
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja digitalnog ulaza PD4
        if((PIND & (1 << PD4)) == 0x00) {
            PORTB |= (1 << PB1); // uključi crvenu LED diodu
        } else {
            PORTB &= ~(1 << PB1); // isključi crvenu LED diodu
        }
    }

    return 0;
}
```

Prevedite datoteku `vjezba51.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, pritiskom na tipkalo T1 (spojeno na digitalni ulaz PD4), uključit će se crvena LED dioda (spojena na digitalni izlaz PB1) na razvojnom okruženju sa slike 2.1.

Pokušajte sada izbrisati naredbu `PORTD |= (1 << PD4);` u funkciji `init()` kojom se uključuje pritezni otpornik. Ponovno prevedite datoteku `vjezba51.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Na razvojnom okruženju sa slike 2.1 prstima dotaknite trnove s oznakom `PORTD` (trnovi se nalaze u gornjem desnom uglu razvojnog okruženja). Primijetit ćete da se crvena LED dioda sada uključuje nekontrolirano. Razlog tomu je što je pin PD4 u stanju visoke impedancije i ima samo jedno stabilno stanje (kada je tipkalo pritisnuto). Kada tipkalo nije pritisnuto, tada će svaka smetnja potencijalno uključiti crvenu LED diodu jer će stanje na pinu biti nisko. To će posebno biti izraženo ako prst druge ruke stavite na mjesto gdje je niski potencijal (`GND`) na razvojnom okruženju sa slike 2.1.

Prethodni način konfiguracije digitalnih pinova te ispitivanja stanja digitalnih ulaza je najsloženiji, ali se najčešće koristi u praksi. S obzirom na složenost prethodnog primjera, izradili smo vlastite makronaredbe kojima se mogu konfigurirati digitalni ulazi, makronaredbe kojima se mogu čitati stanja pinova te makronaredbe kojima se mogu uključivati i isključivati pritezni otpornici. Navedene makronaredbe nalaze se u zaglavlju `avrlib.h`.

U prethodnom poglavlju opisali smo dio makronaredbi koje se nalaze u zaglavlju `avrlib.h`. Za konfiguraciju ulaznih pinova, za čitanje stanja digitalnih pinova te za uključenje i isključenje priteznih otpornika koristit ćemo sljedeće makronaredbe:

- `config_input(DDRx, Pxi)` - makronaredba koja kao argumente prima registar `DDRx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) kojeg želimo postaviti kao ulazni pin,
- `get_pin(PINx, Pxi)` - makronaredba koja kao argumente prima registar `PINx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) na kojem želimo

pročitati stanje, a vraća stanje digitalnog ulaza (0 - nisko stanje, 1 - visoko stanje),

- `pull_up_on(PORTx, Pxi)` - makronaredba koja kao argumente prima registar `PORTx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) na kojem želimo uključiti pritezni otpornik,
- `pull_up_off(PORTx, Pxi)` - makronaredba koja kao argumente prima registar `PORTx` (x = B, C, D) i poziciju pina `Pxi` (x = B, C, D, i = 0, 1, 2, ..., 7) na kojem želimo isključiti pritezni otpornik.

Primjenu navedenih makronaredbi prikazat ćemo na nekoliko sljedećih primjera:

- `config_input(DDRB, PB5)` - pin PB5 konfiguriran je kao ulazni pin,
- `config_input(DDRB, PB7)` - pin PB7 konfiguriran je kao ulazni pin,
- `config_input(DDRC, PC5)` - pin PC5 konfiguriran je kao ulazni pin,
- `config_input(DDRD, PD2)` - pin PD2 konfiguriran je kao ulazni pin,
- `get_pin(PINB, PB5)` - čitanje stanja digitalnog ulaza PB5,
- `get_pin(PIND, PD2)` - čitanje stanja digitalnog ulaza PD2,
- `pull_up_on(PORTB, PB5)` - uključen je pritezni otpornik na digitalnom ulazu PB5,
- `pull_up_off(PORTB, PB7)` - isključen je pritezni otpornik na digitalnom ulazu PB7,
- `pull_up_on(PORTC, PC5)` - uključen je pritezni otpornik na digitalnom ulazu PC5,
- `pull_up_off(PORTD, PD2)` - isključen je pritezni otpornik na digitalnom ulazu PD2.

Slijedom navedenih primjera, modificirajte programski kod 5.3 na sljedeći način:

- umjesto naredbe `DDRB |= (1 << PB1);` upišite naredbu `config_output(DDRB, PB1);`,
- umjesto naredbe `DDRD &= ~(1 << PD4);` upišite naredbu `config_input(DDRD, PD4);`,
- umjesto naredbe `PORTD |= (1 << PD4);` upišite naredbu `pull_up_on(PORTD, PD4);`,
- uvjet `(PIND & (1 << PD4))== 0x00` u `if` uvjetovanom bloku zamijenite uvjetom `get_pin(PIND, PD4)== 0`,
- umjesto naredbe `PORTB |= (1 << PB1);` upišite naredbu `set_pin_on(PORTB, PB1);`,
- umjesto naredbe `PORTB &= ~(1 << PB1);` upišite naredbu `set_pin_off(PORTB, PB1);`.

Modificirani programski kod mora odgovarati programskom kodu 5.4. Ovaj programski kod, kao i prethodni, konfigurira pin PB1 kao digitalni izlaz, pin PD4 kao digitalni ulaz, uključuje pritezni otpornik na pinu PD4 te uključuje crvenu LED diodu ako je pritisnuto tipkalo T1. Prednost pristupa konfiguraciji digitalnih pinova. čitanja stanja pinova pomoću makronaredbi te uključivanju i isključivanju priteznih otpornika je u tome što ne morate voditi računa o poziciji pina, o bitovnim operacijama i slično. Što se brzine izvedbe programa tiče, programski kod 5.3 i programski kod 5.4 se jednako brzo izvode na mikroupravljaču.

Programski kod 5.4: Sadržaj datoteke `vjezba51.cpp` - crvena LED dioda uključena je ako je pritisnuto tipkalo T1 (drugi način)

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    config_output(DDRB, PB1); // PB1 konfiguriran kao izlaz (crvena LED dioda)
    config_input(DDRD, PD4); // PD4 konfiguriran kao ulaz (tipkalo T1)

    pull_up_on(PORTD, PD4); // ukljucen pull up otpornik na ulazu PD4
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
         // provjera stanja digitalnog ulaza PD4
        if (get_pin(PIND, PD4) == 0) {
            set_pin_on(PORTB, PB1); // uključi crvenu LED diodu
            } else {
            set_pin_off(PORTB, PB1); // isključi crvenu LED diodu
        }
    }

    return 0;
}
```

Ponovno prevedite datoteku `vjezba51.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

U *Arduino* IDE programskom razvojnom okruženju, kako smo već spomenuli, koristi se funkcijski pristup za konfiguriranje pinova i postavljanje stanja na digitalne izlaze. Autori ovog udžbenika napisali su funkcije koje se koriste na identičan način kao i *Arduino* funkcije za digitalne pinove. Funkcije su deklarirane u zaglavlju `avrlib.h`.

Pri funkcijskom pristupu za konfiguraciju ulaznih pinova, za čitanje stanja digitalnih ulaza te za uključivanje i isključivanje priteznih otpornika koristit ćemo sljedeće funkcije:

- `pinMode(xi, INPUT);` - funkcija koja prima dva argumenta. Prvi argument je konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) koja predstavlja ime porta i poziciju pina koji želimo konfigurirati. Drugi argument je konstanta koja određuje da li će pin biti ulazni ili izlazni. Za ulazni pin koristi se konstanta `INPUT`.

- `pinMode(xi, INPUT_PULLUP);` - prethodna funkcija kao drugi argument može primiti konstantu `INPUT_PULLUP`. Na ovaj način se konfigurira ulazni pin koji istovremeno ima uključen pritezni otpornik.

- `digitalRead(xi);` - funkcija koja prima jedan argument. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina čije stanje želimo pročitati. Funkcija vraća `bool` vrijednost: `false` za stanje pina koje je jednako 0 (nisko stanje) te `true` za stanje pina koje je jednako 1 (visoko stanje).

- `pullUpOn(xi);` - funkcija koja prima jedan argument. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina na kojem želimo uključiti pritezni otpornik.

- `pullUpOff(xi);` - funkcija koja prima jedan argument. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina na kojem želimo isključiti pritezni otpornik.

Ponovit ćemo da konstante `D4` i `PD4` nisu iste konstante, ali se odnose na isti digitalni pin

(pin PD4). Konstanta D4 koristi se pri funkcijskom pristupu konfiguracije digitalnih pinova, a konstanta PD4 se koristi pri korištenju makronaredbi za konfiguriranje digitalnih pinova.

Primjenu navedenih funkcija prikazat ćemo na nekoliko sljedećih primjera:

- `pinMode(B5, INPUT)` - pin PB5 konfiguriran je kao ulazni pin,
- `pinMode(B7, INPUT_PULLUP)` - pin PB7 konfiguriran je kao ulazni pin te je uključen pritezni otpornik na njemu,
- `pinMode(C5, INPUT_PULLUP)` - pin PC5 konfiguriran je kao ulazni pin te je uključen pritezni otpornik na njemu,
- `pinMode(D2, INPUT)` - pin PD2 konfiguriran je kao ulazni pin,
- `digitalRead(B5)` - čitanje stanja digitalnog ulaza PB5,
- `digitalRead(D2)` - čitanje stanja digitalnog ulaza PD2,
- `pullUpOn(B5)` - uključen je pritezni otpornik na digitalnom ulazu PB5,
- `pullUpOff(B7)` - isključen je pritezni otpornik na digitalnom ulazu PB7,
- `pullUpOff(C5)` - isključen je pritezni otpornik na digitalnom ulazu PC5,
- `pullUpOn(D2)` - uključen je pritezni otpornik na digitalnom ulazu PD2.

Slijedom navedenih primjera, modificirajte programski kod 5.4 na sljedeći način:

- umjesto naredbe `config_output(DDRB, PB1);` upišite naredbu `pinMode(B1, OUTPUT);`,
- umjesto naredbe `config_input(DDRD, PD4);` upišite naredbu `pinMode(D4, INPUT);`,
- umjesto naredbe `pull_up_on(PORTD, PD4);` upišite naredbu `pullUpOn(D4);`,
- uvjet `get_pin(PIND, PD4)== 0` u `if` uvjetovanom bloku zamijenite uvjetom `digitalRead(D4)== false`,
- umjesto naredbe `set_pin_on(PORTB, PB1);` upišite naredbu `digitalWrite(B1, HIGH);`,
- umjesto naredbe `set_pin_off(PORTB, PB1);` upišite naredbu `digitalWrite(B1, LOW);`.

Modificirani programski kod mora odgovarati programskom kodu 5.5. Ovaj programski kod, kao i prethodna dva, konfigurira pin PB1 kao digitalni izlaz, pin PD4 kao digitalni ulaz, uključuje pritezni otpornik na pinu PD4 te uključuje crvenu LED diodu ako je pritisnuto tipkalo T1. Ovaj način konfiguracije digitalnih pinova (programski kod 5.5) je najintuitivniji i najlakši za korištenje.

Programski kod 5.5: Sadržaj datoteke `vjezba51.cpp` - crvena LED dioda uključena je ako je pritisnuto tipkalo T1 (treći način)

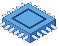
```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlaz (crvena LED dioda)
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulaz (tipkalo T1)
```

```
    pullUpOn(D4); // uključen pull up otpornik na ulazu PD4
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
         // provjera stanja digitalnog ulaza PD4
        if (digitalRead(D4) == false) {
            digitalWrite(B1, HIGH); // uključi crvenu LED diodu
            } else {
            digitalWrite(B1, LOW); // isključi crvenu LED diodu
        }
    }

    return 0;
}
```

Ponovno prevedite datoteku `vjezba51.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Prikazali smo tri načina kako pristupiti problemu definiranom ovom vježbom. Prva dva načina su programski najefikasnija, zauzimaju najmanje programske memorije i najčešće se koriste u praksi. Treći način je najjednostavniji, ali ta jednostavnost nas "košta" (engl. *computing cost*) programske memorije.

Zatvorite datoteku `vjezba51.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 5.2

Napišite program koji će uključiti crvenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P ako je pritisnuto tipkalo T1, žutu LED diodu ako je pritisnuto tipkalo T2 te zelenu LED diodu ako je pritisnuto tipkalo T3. Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, žuta LED dioda na digitalni izlaz PB2, zelena LED dioda na digitalni izlaz PB3, tipkalo T1 na digitalni ulaz PD4, tipkalo T2 na digitalni ulaz PD2, a tipkalo T3 na digitalni ulaz PD5 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba52.cpp`. Omogućite prevođenje datoteke `vjezba52.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba52.cpp` prikazan je programskim kodom 5.6.

Programski kod 5.6: Početni sadržaj datoteke `vjezba52.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }
```

```
    return 0;
}
```

Problem koji moramo riješiti jest napisati program koji će:

- uključiti crvenu LED diodu koja je spojena na digitalni pin PB1 mikroupravljača ATmega328P ako je pritisnuto tipkalo T1 koje je spojeno na digitalni pin PD4,
- uključiti žutu LED diodu koja je spojena na digitalni pin PB2 mikroupravljača ATmega328P ako je pritisnuto tipkalo T2 koje je spojeno na digitalni pin PD2,
- uključiti zelenu LED diodu koja je spojena na digitalni pin PB3 mikroupravljača ATmega328P ako je pritisnuto tipkalo T3 koje je spojeno na digitalni pin PD5.

Najprije je potrebno digitalne pinove PB1, PB2 i PB3 konfigurirati kao izlazne, digitalne pinove PD4, PD2 i PD5 konfigurirati kao ulazne te uključiti pritezne otpornike na pinovima PD4, PD2 i PD5. Funkciju `init()` u programskom kodu 5.6 potrebno je modificirati kako bi se provela navedena konfiguracija. U funkciju `init()` upišite sljedeće linije programskog koda (koristit ćemo makronaredbe):

- upišite naredbu `config_output(DDRB, PB1);` kojom pin PB1 konfiguriramo kao izlazni pin,
- upišite naredbu `config_output(DDRB, PB2);` kojom pin PB2 konfiguriramo kao izlazni pin,
- upišite naredbu `config_output(DDRB, PB3);` kojom pin PB3 konfiguriramo kao izlazni pin,
- upišite naredbu `config_input(DDRD, PD4);` kojom pin PD4 konfiguriramo kao ulazni pin,
- upišite naredbu `pull_up_on(PORTD, PD4);` kojom se uključuje pritezni otpornik na pinu PD4,
- upišite naredbu `config_input(DDRD, PD2);` kojom pin PD2 konfiguriramo kao ulazni pin,
- upišite naredbu `pull_up_on(PORTD, PD2);` kojom se uključuje pritezni otpornik na pinu PD2,
- upišite naredbu `config_input(DDRD, PD5);` kojom pin PD5 konfiguriramo kao ulazni pin,
- upišite naredbu `pull_up_on(PORTD, PD5);` kojom se uključuje pritezni otpornik na pinu PD5.

Za provjeru stanja pina koristit ćemo makronaredbu `get_pin()` (vidi programski kod 5.4). U `while` petlju programskog koda 5.6 dodajte uvjetovane blokove kojima se uključuju LED diode (kopirajte primjer iz programskog koda 5.4 za crvenu LED diodu i tipkalo T1). Kopirani primjer umnožite dva puta te promijenite digitalne ulaze u makronaredbama `get_pin()` i digitalne izlaze u makronaredbama `set_pin_on()` i `set_pin_off()`. Rješenje problema pomoću makronaredaba kojeg smo definirali na početku vježbe prikazano je u programskom kodu 5.7.

Programski kod 5.7: Sadržaj datoteke `vjezba52.cpp` - program koji uključuje crvenu, žutu i zelenu LED diodu ako su pritisnuta tipkala T1, T2 ili T3 (korištenje makronaredbi)

```
#include <avr/io.h>
#include "AVR/avrlib.h"
```

```
void init() {

    config_output(DDRB, PB1); // PB1 konfiguriran kao izlaz (crvena LED dioda)
    config_output(DDRB, PB2); // PB2 konfiguriran kao izlaz (žuta LED dioda)
    config_output(DDRB, PB3); // PB3 konfiguriran kao izlaz (zelena LED dioda)
    // tipkalo T1
    config_input(DDRD, PD4); // PD4 konfiguriran kao ulaz
    pull_up_on(PORTD, PD4); // uključen pull up otpornik na ulazu PD4
    // tipkalo T2
    config_input(DDRD, PD2); // PD2 konfiguriran kao ulaz
    pull_up_on(PORTD, PD2); // uključen pull up otpornik na ulazu PD2
    // tipkalo T3
    config_input(DDRD, PD5); // PD5 konfiguriran kao ulaz
    pull_up_on(PORTD, PD5); // uključen pull up otpornik na ulazu PD5
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja digitalnog ulaza PD4
        if (get_pin(PIND, PD4) == 0) {
            set_pin_on(PORTB, PB1); // uključi crvenu LED diodu
        } else {
            set_pin_off(PORTB, PB1); // isključi crvenu LED diodu
        }
        // provjera stanja digitalnog ulaza PD2
        if (get_pin(PIND, PD2) == 0) {
            set_pin_on(PORTB, PB2); // uključi žutu LED diodu
            } else {
            set_pin_off(PORTB, PB2); // isključi žutu LED diodu
        }
        // provjera stanja digitalnog ulaza PD5
        if (get_pin(PIND, PD5) == 0) {
            set_pin_on(PORTB, PB3); // uključi zelenu LED diodu
            } else {
            set_pin_off(PORTB, PB3); // isključi zelenu LED diodu
        }
    }

    return 0;
}
```

Prevedite datoteku `vjezba52.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili gore navedene korake, tipkalom T1 uključuje se crvena LED dioda, tipkalom T2 uključuje se zelena LED dioda, a tipkalom T3 žuta LED dioda.

Definirani problem u ovoj vježbi može se riješiti i funkcijskim pristupom (kao u programskom kodu 5.5). Modificirajte programski kod 5.7 na sljedeći način:

- umjesto naredbe `config_output(DDRB, PB1);` upišite naredbu `pinMode(B1, OUTPUT);`,
- umjesto naredbe `config_output(DDRB, PB2);` upišite naredbu `pinMode(B2, OUTPUT);`,
- umjesto naredbe `config_output(DDRB, PB3);` upišite naredbu `pinMode(B3, OUTPUT);`,
- umjesto naredbe `config_input(DDRD, PD4);` upišite naredbu `pinMode(D4, INPUT);`,
- umjesto naredbe `pull_up_on(PORTD, PD4);` upišite naredbu `pullUpOn(D4);`,

- umjesto naredbi `config_input(DDRD, PD2);` i `pull_up_on(PORTD, PD2);` upišite naredbu `pinMode(D2, INPUT_PULLUP);` - u ovom primjeru smo konfigurirali digitalni ulaz uz uključenje priteznog otpornika na pinu PD2 (drugi argument funkcije `pinMode` jest `INPUT_PULLUP`),
- umjesto naredbi `config_input(DDRD, PD5);` i `pull_up_on(PORTD, PD5);` upišite naredbu `pinMode(D5, INPUT_PULLUP);` - u ovom primjeru smo konfigurirali digitalni ulaz uz uključenje priteznog otpornika na pinu PD5 (drugi argument funkcije `pinMode` jest `INPUT_PULLUP`),
- uvjet `get_pin(PIND, PD4)== 0` u `if` uvjetovanom bloku zamijenite uvjetom `digitalRead(D4)== false`,
- umjesto naredbe `set_pin_on(PORTB, PB1);` upišite naredbu `digitalWrite(B1, HIGH);`,
- umjesto naredbe `set_pin_off(PORTB, PB1);` upišite naredbu `digitalWrite(B1, LOW);`,
- uvjet `get_pin(PIND, PD2)== 0` u `if` uvjetovanom bloku zamijenite uvjetom `digitalRead(D2)== false`,
- umjesto naredbe `set_pin_on(PORTB, PB2);` upišite naredbu `digitalWrite(B2, HIGH);`,
- umjesto naredbe `set_pin_off(PORTB, PB2);` upišite naredbu `digitalWrite(B2, LOW);`,
- uvjet `get_pin(PIND, PD5)== 0` u `if` uvjetovanom bloku zamijenite uvjetom `digitalRead(D5)== false`,
- umjesto naredbe `set_pin_on(PORTB, PB3);` upišite naredbu `digitalWrite(B3, HIGH);`,
- umjesto naredbe `set_pin_off(PORTB, PB3);` upišite naredbu `digitalWrite(B3, LOW);`.

Modificirani programski kod mora odgovarati programskom kodu 5.8.

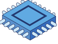
Programski kod 5.8: Sadržaj datoteke `vjezba52.cpp` - program koji uključuje crvenu, žutu i zelenu LED diodu ako su pritisnuta tipkala T1, T2 ili T3 (korištenje funkcija)

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlaz (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlaz (žuta LED dioda)
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlaz (zelena LED dioda)
    // tipkalo T1
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulaz
    pullUpOn(D4); // ukljucen pull up otpornik na ulazu PD4
    // tipkalo T2
    pinMode(D2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // ukljucen pull up otpornik na ulazu PD2
    // tipkalo T3
    pinMode(D5, INPUT_PULLUP);  // PD5 konfiguriran kao ulaz
    // ukljucen pull up otpornik na ulazu PD5
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
```

```
    while (1) { // beskonačna petlja
        // provjera stanja digitalnog ulaza PD4
        if (digitalRead(D4) == false) {
            digitalWrite(B1, HIGH); // uključi crvenu LED diodu
        } else {
            digitalWrite(B1, LOW); // isključi crvenu LED diodu
        }
        // provjera stanja digitalnog ulaza PD2
        if (digitalRead(D2) == false) {
            digitalWrite(B2, HIGH); // uključi žutu LED diodu
            } else {
            digitalWrite(B2, LOW); // isključi žutu LED diodu
        }
        // provjera stanja digitalnog ulaza PD5
        if (digitalRead(D5) == false) {
            digitalWrite(B3, HIGH); // uključi zelenu LED diodu
            } else {
            digitalWrite(B3, LOW); // isključi zelenu LED diodu
        }
    }

    return 0;
}
```

Ponovno prevedite datoteku `vjezba52.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Zanimljiva činjenica jest da programski kod 5.7 zauzima 278 B programske memorije i 0 B podatkovne memorije, dok programski kod 5.8 zauzima 792 B programske memorije i 24 B podatkovne memorije. Kao što smo spomenuli u prethodnoj vježbi, jednostavnost funkcijskog pristupa programiranja mikroupravljača plaća se većim zauzećem memorije (gotovo tri puta većim) što u konačnici rezultira i sporijim izvođenjem strojnog koda za istu funkcionalnost razvojnog okruženja. Također, cijelo vrijeme na umu treba imati da mikroupravljači imaju relativno malo programske i podatkovne memorije.

Zatvorite datoteku `vjezba52.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 5.3

Napišite program koji će uključiti crvenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P ako su pritisnuta tipkala T1 i T2, a žutu LED diodu ako nije pritisnuto tipkalo T3. Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, žuta LED dioda na digitalni izlaz PB2, tipkalo T1 na digitalni ulaz PD4, tipkalo T2 na digitalni ulaz PD2, a tipkalo T3 na digitalni ulaz PD5 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba53.cpp`. Omogućite prevođenje datoteke `vjezba53.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba53.cpp` prikazan je programskim kodom 5.9.

Programski kod 5.9: Početni sadržaj datoteke `vjezba53.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    config_output(DDRB, PB1); // PB1 konfiguriran kao izlaz (crvena LED dioda)
    config_output(DDRB, PB2); // PB2 konfiguriran kao izlaz (žuta LED dioda)
    // tipkalo T1
```

```
    config_input(DDRD, PD4); // PD4 konfiguriran kao ulaz
    pull_up_on(PORTD, PD4); // ukljucen pull up otpornik na ulazu PD4
    // tipkalo T2
    config_input(DDRD, PD2); // PD2 konfiguriran kao ulaz
    pull_up_on(PORTD, PD2); // ukljucen pull up otpornik na ulazu PD2
    // tipkalo T3
    config_input(DDRD, PD5); // PD5 konfiguriran kao ulaz
    pull_up_on(PORTD, PD5); // ukljucen pull up otpornik na ulazu PD5
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Problem koji moramo riješiti jest napisati program koji će:

- uključiti crvenu LED diodu koja je spojena na digitalni pin PB1 mikroupravljača ATmega328P ako su pritisnuta tipkala T1 koje je spojeno na digitalni pin PD4 i T2 koje je spojeno na digitalni pin PD2,

- uključiti žutu LED diodu koja je spojena na digitalni pin PB2 mikroupravljača ATmega328P ako nije pritisnuto tipkalo T3 koje je spojeno na digitalni pin PD5.

U početnom programskom kodu 5.9 napravljena je konfiguracija digitalnih izlaza PB1 i PB2 te digitalnih ulaza PD4, PD2 i PD5.

Crvena LED dioda uključuje se kada su tipkala T1 i T2 u niskom stanju (pritisnuta su). Prema tome, potrebno je uvjete `get_pin(PIND, PD4)== 0` i `get_pin(PIND, PD2)== 0` povezati s logičkim `&&` operatorom kako bi se ostvarila navedena funkcionalnost. Žuta LED dioda uključuje se kada je tipkalo T3 u visokom stanju (nije pritisnuto). Uvjet u ovom slučaju mora biti `get_pin(PIND, PD5)== 1`. Definirane uvjete ispitujemo unutar uvjetovanog `if` bloka. Program koji uključuje crvenu LED diodu ako su pritisnuta tipkala T1 i T2 te žutu LED diodu ako nije pritisnuto tipkalo T3 prikazan je programskim kodom 5.10.

Programski kod 5.10: Sadržaj datoteke `vjezba53.cpp` - program koji uključuje crvenu LED diodu ako su pritisnuta tipkala T1 i T2 te žutu LED diodu ako nije pritisnuto tipkalo T3 (korištenje makronaredbi)

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

    config_output(DDRB, PB1); // PB1 konfiguriran kao izlaz
    config_output(DDRB, PB2); // PB2 konfiguriran kao izlaz
    // tipkalo T1
    config_input(DDRD, PD4); // PD4 konfiguriran kao ulaz
    pull_up_on(PORTD, PD4); // uključen pull up otpornik na ulazu PD4
    // tipkalo T2
    config_input(DDRD, PD2); // PD2 konfiguriran kao ulaz
    pull_up_on(PORTD, PD2); // uključen pull up otpornik na ulazu PD2
    // tipkalo T3
    config_input(DDRD, PD5); // PD5 konfiguriran kao ulaz
    pull_up_on(PORTD, PD5); // uključen pull up otpornik na ulazu PD5
```

```
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja tipkala T1 i T2
        if ((get_pin(PIND, PD4) == 0) && (get_pin(PIND, PD2) == 0)) {
            set_pin_on(PORTB, PB1); // uključi crvenu LED diodu
            } else {
            set_pin_off(PORTB, PB1); // isključi crvenu LED diodu
        }

        // provjera stanja tipkala T3
        if (get_pin(PIND, PD5) == 1) {
            set_pin_on(PORTB, PB2); // uključi zelenu LED diodu
            } else {
            set_pin_off(PORTB, PB2); // isključi zelenu LED diodu
        }
    }

    return 0;
}
```

Prevedite datoteku `vjezba53.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili gore navedene korake, istovremenim pritiskom na tipkala T1 i T2 uključuje se crvena LED dioda. Žuta LED dioda će biti uključena dok se ne pritisne tipkalo T3.

Definirani problem u ovoj vježbi može se riješiti i funkcijskim pristupom. Rješenje će biti inspirirano programskom kodom 5.8. No, prije nego se bacimo na pisanje programskog koda, postavit ćemo si jedno pitanje. Kada bismo u programu trebali uključivati 10 uređaja i pri tome imali uvjete koji su proizašli s 10 senzora, da li bismo se snalazili u programskom kodu ako bismo koristili konstante `B1`, `B2`, `B3` itd? Odgovor bi vjerojatno bio ne! Pomoći si možemo tako da za pojedine uređaje definiramo simboličko ime pretprocesorskom naredbom `#define`. Na primjer, ako je crvena LED dioda spojena na pin PB1, tada bi za potrebe funkcijskog pristupa programiranju bilo korisno napraviti konstantu `#define LED_CRVENO B1`. Sada bismo umjesto konstante `B1` pri konfiguraciji pina PB1 mogli koristiti konstantu `LED_CRVENO`. Na primjer, digitalni izlaz na kojem je spojena crvena LED dioda sada bismo mogli konfigurirati pozivom funkcije `pinMode(LED_CRVENO, OUTPUT);`. Kreiranje svih konstanti (simboličkih imena) za crvenu i žutu LED diodu te tipkala T1, T2 i T3 prikazano je u programskom kodu 5.11.

Programski kod 5.11: Kreiranje svih konstanti (simboličkih imena) za crvenu i žutu LED diodu te tipkala T1, T2 i T3

```
#define LED_CRVENO B1 // umjesto B1, sada koristimo konstantu LED_CRVENO
#define LED_ZUTO B2 // umjesto B2, sada koristimo konstantu LED_ZUTO

#define TIPKALO1 D4 // umjesto D4, sada koristimo konstantu  TIPKALO1
#define TIPKALO2 D2 // umjesto D2, sada koristimo konstantu  TIPKALO2
#define TIPKALO3 D5 // umjesto D5, sada koristimo konstantu  TIPKALO3
```

Čitanje i pisanje programskog koda s navedenim konstantama je sada praktičnije jer samo na početku moramo znati da je crvena LED dioda spojena na pin PB1. Dalje pri pisanju programskog koda koristimo simboličko ime `LED_CRVENO` bez da razmišljamo na koji pin je crvena LED dioda spojena.

Program koji uključuje crvenu LED diodu ako su pritisnuta tipkala T1 i T2 te žutu LED diodu ako nije pritisnuto tipkalo T3 korištenjem funkcija prikazan je programskim kodom

5.12. Simbolička imena su navedena u globalnom prostoru programskog koda, odmah nakon uključivanja potrebitih zaglavlja.

U funkciji `init()` programskog koda 5.12 napravljena je konfiguracija digitalnih izlaza PB1 i PB2 te digitalnih ulaza PD4, PD2 i PD5, ali ovaj puta korištenjem simboličkih imena. Na primjer, digitalni ulaz PD4 na kojem je spojeno tipkalo T1 konfigurirali smo pozivom funkcije `pinMode(TIPKALO1, INPUT);`.

Za uključenje crvene LED diode sada je pomoću `&&` operatora potrebno povezati uvjete `digitalRead(TIPKALO1)== false` i `digitalRead(TIPKALO2)== false`. Uvjet za uključenje žute LED diode je `digitalRead(TIPKALO3)== true`. Definirane uvjete ispitujemo unutar uvjetovanog `if` bloka.

Programski kod 5.12: Sadržaj datoteke `vjezba53.cpp` - program koji uključuje crvenu LED diodu ako su pritisnuta tipkala T1 i T2 te žutu LED diodu ako nije pritisnuto tipkalo T3 (korištenje funkcija)

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_CRVENO B1 // umjesto B1, sada koristimo konstantu LED_CRVENO
#define LED_ZUTO B2 // umjesto B2, sada koristimo konstantu LED_ZUTO

#define TIPKALO1 D4 // umjesto D4, sada koristimo konstantu TIPKALO1
#define TIPKALO2 D2 // umjesto D2, sada koristimo konstantu TIPKALO2
#define TIPKALO3 D5 // umjesto D5, sada koristimo konstantu TIPKALO3

void init() {

    pinMode(LED_CRVENO, OUTPUT); // PB1 konfiguriran kao izlaz
    pinMode(LED_ZUTO, OUTPUT); // PB2 konfiguriran kao izlaz

    // tipkalo T1
    pinMode(TIPKALO1, INPUT); // PD4 konfiguriran kao ulaz
    pullUpOn(TIPKALO1); // uključen pull up otpornik na ulazu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na ulazu PD2
    // tipkalo T3
    pinMode(TIPKALO3, INPUT_PULLUP);  // PD5 konfiguriran kao ulaz
    // uključen pull up otpornik na ulazu PD5
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja tipkala T1 i T2
        if ((digitalRead(TIPKALO1) == false) &&
        (digitalRead(TIPKALO2) == false)) {
            digitalWrite(LED_CRVENO, HIGH); // uključi crvenu LED diodu
            } else {
            digitalWrite(LED_CRVENO, LOW); // isključi crvenu LED diodu
        }
        // provjera stanja tipkala T3
        if (digitalRead(TIPKALO3) == true) {
            digitalWrite(LED_ZUTO, HIGH); // uključi žutu LED diodu
            } else {
            digitalWrite(LED_ZUTO, LOW); // isključi žutu LED diodu
        }
    }
    return 0;
```

```
}
```

Ponovno prevedite datoteku `vjezba53.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Korištenje simboličkih imena jednostavnije je kod funkcijskog pristupa programiranju jer u našem slučaju, funkcije za konfiguraciju digitalnih pinova te za postavljanje i čitanje stanja digitalnih pinova primaju samo jednu konstantu (npr. `B1`). Kada bismo koristili makronaredbe ili konfigurirali registre bitovnom ILI i I operacijom, tada bismo za jedan digitalni pin trebali kreirati četiri simbolička imena. Primjer kreiranja simboličkih imena za crvenu LED diodu dan je programskim kodom 5.13. Primijetimo da za svaki uređaj (u ovom slučaju je to crvena LED dioda) moramo kreirati konstantu za `DDR`, `PORT` i `PIN` registar te za poziciju pina. Sada za konfiguraciju digitalnog izlaza na pinu PB1 (crvena LED dioda) umjesto naredbe `config_output(DDRB, PB1);` možemo koristiti naredbu `config_output(LED_CRVENO_DDR, LED_CRVENO);`. Za uključivanje crvene LED diode umjesto naredbe `PORTB |= (1 << PB1);` možemo koristiti naredbu `LED_CRVENO_PORT |= (1 << LED_CRVENO);`. I dalje ovaj pristup zauzima višestruko manje programske memorije od pristupa u kojem se koriste funkcije za konfiguraciju pinova i drugo.

Programski kod 5.13: Kreiranje simboličkih imena za crvenu LED diodu i uključenje crvene diode

```
#include <avr/io.h>
#include "AVR/avrlib.h"

// simbolicka imena za pin PB1
#define LED_CRVENO_DDR DDRB // DDR registar
#define LED_CRVENO_PORT PORTB // PORT registar
#define LED_CRVENO_PIN PINB // PIN registar
#define LED_CRVENO PB1 // pozicija pina

void init() {

    // konfiguracija izlaznog pina za crvenu LED diodu
    config_output(LED_CRVENO_DDR, LED_CRVENO);
    // uključenje crvene LED diode
    LED_CRVENO_PORT |= (1 << LED_CRVENO);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    return 0;
}
```

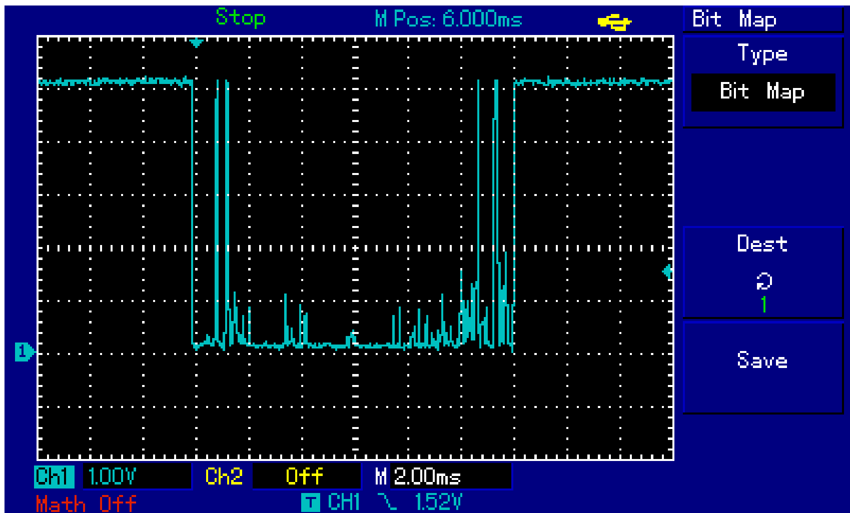
Zatvorite datoteku `vjezba53.cpp` i onemogućite prevođenje ove datoteke.

Vježba 5.4

Napišite program koji će uključiti zelenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P ako je pritisnuto tipkalo T2. Dodatno, potrebno je filtrirati smetnje koje se javljaju zbog istitravanja tipkala (engl. *bounce*). Prema shemi na slici 5.2, zelena LED dioda spojena je na digitalni izlaz PB3, a tipkalo T2 na digitalni ulaz PD2 mikroupravljača ATmega328P.

Kada tipkalo ili sklopka mijenjaju stanje pojavljuje se istitravanje kontakata tipkala ili sklopke. Ovaj efekt posebno je naglašen kod istrošenih tipkala. Primjer istitravanja kontakata

tipkala prikazan je na slici 5.3. Prilikom pritiska tipkala, stanje tipkala prelazi iz visokog u nisko stanje. U prijelaznoj fazi kontakti tipkala generiraju nekoliko bridova signala što može uzrokovati neželjeno ponašanje napisanog programskog koda.



Slika 5.3: Istitravanje kontakata tipkala, (Izvor: Veleučilište u Bjelovaru, odjel Mehatronika)

Pretpostavimo da moramo prebrojiti koliko je puta neko tipkalo bilo pritisnuto. Kada bismo za provjeru stanja tipkala koristili makronaredbu `get_pin` ili funkciju `digitalRead()`, varijabla u koju spremamo koliko je puta tipkalo bilo pritisnuto povećala bi se vrlo vjerojatno za 3[2] u slučaju signala na slici 5.3. Problem istitravanja kontakata tipkala može se riješiti hardverski ili softverski. Hardverski način uklanjanja smetnji koji se javljaju prilikom pritiska na tipkalo može se realizirati korištenjem niskopropusnog filtra na ulaznom digitalnom pinu. Najjednostavniji način jest paralelno tipkalu dodati keramički kondenzator kapaciteta 100 nF. No, ponekad ne možete intervenirati u elektronički uređaj pa je problem potrebno riješiti softverskim putem. Softverski se istitravanje kontakata tipkala može riješiti tako da unutar definiranog vremenskog okvira mjerimo koliko je tipkalo dugo bilo u niskom stanju (koliko je dugo pritisnuto). Ako je u definiranom vremenskom okviru tipkalo 90% vremena pritisnuto, onda možemo sa sigurnošću tvrditi da je tipkalo pritisnuto. Vrijednost 90% izabrana je iskustveno. Na ovaj način softverski filtriramo signal koji je proizveden tipkalom. Valja naglasiti da senzori koji u izlaznom krugu imaju tranzistore, nemaju ovakva istitravanja pa u tim slučajevima ulazni signal nije potrebno filtrirati.

Softverski se istitravanje kontakata tipkala može eliminirati funkcijom `filteredPinState()` čija se deklaracija nalazi u zaglavlju `avrlib.h`. Funkcija `filteredPinState()` prima tri argumenta (`bool filteredPinState(uint8_t pin, bool value, uint16_t T)`):

- `pin` - prvi argument je konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) koja predstavlja ime porta i poziciju pina koji želimo konfigurirati,
- `value` - drugi argument je stanje koje želimo ispitati na poziciji ulaznog pina. Ako želimo ispitati je li na poziciji pina nisko stanje, ovdje ćemo upisati `false` (ili 0), a ako želimo ispitati je li na poziciji pina visoko stanje, ovdje ćemo upisati `true` (ili 1)).
- `T` - treći argument je vremenska konstanta digitalnog filtra u milisekundama koja predstavlja vremenski okvir unutar kojeg mjerimo koliko je tipkalo dugo bilo u niskom ili visokom stanju.

[2] Povećanje će svakako ovisiti o vremenskim kašnjenjima koja se dešavaju unutar jednog ciklusa beskonačne `while` petlje. Stoga nije moguće sa sigurnošću tvrditi da li će povećanje biti 3, 2 ili neko drug.

Funkcija `filteredPinState()` vraća vrijednost `value` ako je pin 90% vremena `T` (treći argument funkcije `filteredPinState()`) u stanju `value`, a `!value` (not `value`) inače. Vrijednost 90% definirana je konstantom `FILTER_TIME`, a može se postaviti i na vrijednosti 70%, 80% i 100%. Pretpostavimo da u programskom kodu imamo poziv funkcije `filteredPinState(D4, false, 50)`. Ova će funkcija vratiti vrijednost `false` ako je ulazni digitalni pin PD4 90 % vremena definiranog vremenskom konstantom (treći argument jest 50 ms) bio u niskom stanju. Na ovaj način će smetnje istitravanja kontakata tipkala biti filtrirane i na primjeru slike 5.3 brojač koji broji koliko je puta tipkalo bilo pritisnuto uvećao bi se za 1. Vremenska konstanta digitalnog filtra odabire se iskustveno, a preporučena joj je vrijednost 25 ms. Definicija funkcije (tijelo funkcije) `filteredPinState()` nalazi se u datoteci `avrlib.cpp`. Ako želite znati više, sigurni smo da ćete zaviriti "ispod haube" funkcije `filteredPinState()`.

U projektnom stablu otvorite datoteku `vjezba54.cpp`. Omogućite prevođenje datoteke `vjezba54.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba54.cpp` prikazan je programskim kodom 5.14.

Programski kod 5.14: Početni sadržaj datoteke `vjezba54.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_ZELENO B3
#define TIPKALO2 D2

void init() {
    // PB3 konfiguriran kao izlaz (zelena LED dioda)
    pinMode(LED_ZELENO, OUTPUT);
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja tipkala T2
        if (digitalRead(TIPKALO2) == false) {
            digitalWrite(LED_ZELENO, HIGH); // uključi zelenu LED diodu
            } else {
            digitalWrite(LED_ZELENO, LOW); // isključi zelenu LED diodu
        }
    }

    return 0;
}
```

Problem koji moramo riješiti jest napisati program koji će uključiti zelenu LED diodu koja je spojena na digitalni pin PB2 mikroupravljača ATmega328P ako je pritisnuto tipkalo T2 koje je spojeno na digitalni pin PD5 uz filtriranje istitravanja kontakta tipkala. U početnom programskom kodu 5.14 nalazi se program koji stanje tipkala provjerava pomoću funkcije `digitalWrite()`. U funkciji `init()` konfiguriran je digitalni izlaz na koji je spojena zelena LED dioda (PB3) i digitalni ulaz na koji je spojeno tipkalo T2 (PD2). Koristili smo simbolička imena `LED_ZELENO` i `TIPKALO2`.

U `while` petlji prikazanoj u programskom kodu 5.14 nalazi se niz naredbi koje će uključiti zelenu LED diodu ako je pritisnuto tipkalo T2. Za provjeru stanja ulaznog digitalnog pina korištena je funkcija `digitalRead()`. Modificirajte programski kod 5.14 tako da zamijenite poziv funkcije `digitalRead(TIPKALO2)` u uvjetovanom bloku `if` s funkcijom

filteredPinState(TIPKALO2, false, 50) koja će filtrirati ulazni signal. Vremenska konstanta digitalnog filtra iznosi 50 ms (treći argument funkcije filteredPinState()). Modificirani program prikazan je programskim kodom 5.15.

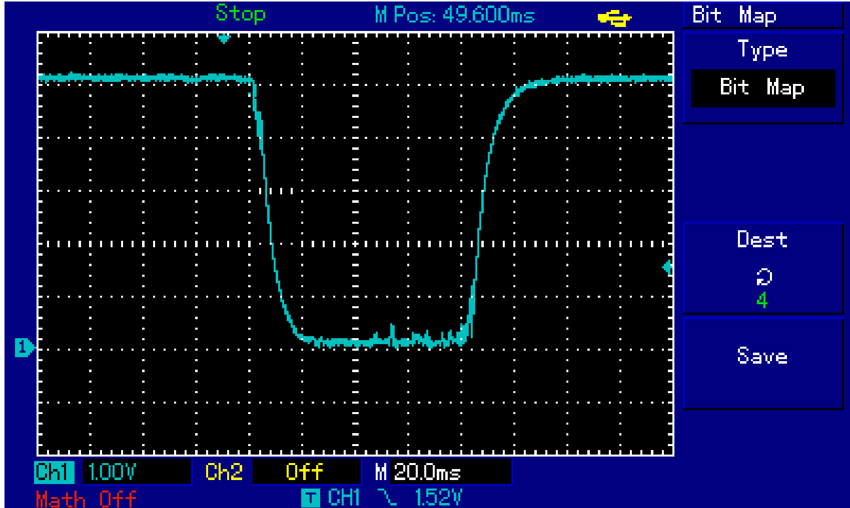
Programski kod 5.15: Sadržaj datoteke vjezba54.cpp - program koji uključuje zelenu LED diodu ako je pritisnuto tipkalo T2 uz filtriranje ulaznog signala

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_ZELENO B3
#define TIPKALO2 D2

void init() {
    // PB3 konfiguriran kao izlaz (zelena LED dioda)
    pinMode(LED_ZELENO, OUTPUT);
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        // provjera stanja tipkala T2
        if (filteredPinState(TIPKALO2, false, 50) == false) {
            digitalWrite(LED_ZELENO, HIGH); // uključi zelenu LED diodu
            } else {
            digitalWrite(LED_ZELENO, LOW); // isključi zelenu LED diodu
        }
    }

    return 0;
}
```

Prevedite datoteku vjezba54.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili gore navedene korake, pritiskom na tipkalo T2 uključit će se zelena LED dioda na razvojnom okruženju uz filtriranje ulaznog signala. Na prvi pogled razlike između rada programskog koda 5.14 i 5.15 nema, no ona se vidi kada tipkalo koristimo u svrhu brojanja impulsa.

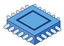
Promijenite vremensku konstantu digitalnog filtra s vrijednosti 50 ms na 1500 ms (treći argument funkcije filteredPinState()). Ponovno prevedite datoteku vjezba54.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Što primjećujemo? Povećanjem vremenske konstante digitalnog filtra unosimo vremensko kašnjenje pa će se i zelena LED dioda uključivati i isključivati s vremenskim zatezanjem. Ponekad to može biti i korisna funkcija ako morate npr. uključiti neki uređaj ako je tipkalo pritisnuto barem 2 s. Pokušajte proizvoljno promijeniti vremensku konstantu digitalnog filtra i testirati program.

Bolje rezultate filtriranja dat će hardverski pristup filtriranju. Hardversko filtriranje istitravanja kontakata tipkala prikazano je na slici 5.4. Na slici možemo vidjeti da prilikom pritiska tipkala i otpuštanja tipkala sada imamo samo jedan brid signala pa će pouzdan rezultat dati i korištenje funkcije digitalRead().



Slika 5.4: Hardversko filtriranje istitravanja kontakata tipkala, (Izvor: Veleučilište u Bjelovaru, odjel Mehatronika)

Zatvorite datoteku `vjezba54.cpp` i onemogućite prevođenje ove datoteke.



Vježba 5.5

Napišite program koji će uključiti zelenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega328P na padajući brid signala tipkala T1, a isključiti zelenu LED diodu na rastući brid signala tipkala T2. Nadalje, na padajući brid signala tipkala T3 potrebno je izmjenjivati stanja crvene i žute LED dioda (kada je uključena crvena LED dioda, žuta je isključena i obratno). Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, žuta LED dioda na digitalni izlaz PB2, zelena LED dioda na digitalni izlaz PB3, tipkalo T1 na digitalni ulaz PD4, tipkalo T2 na digitalni ulaz PD2, a tipkalo T3 na digitalni ulaz PD5 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba55.cpp`. Omogućite prevođenje datoteke `vjezba55.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba55.cpp` prikazan je programskim kodom 5.16.

Programski kod 5.16: Početni sadržaj datoteke `vjezba55.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_CRVENO B1
#define LED_ZUTO B2
#define LED_ZELENO B3
#define TIPKALO1 D4
#define TIPKALO2 D2
#define TIPKALO3 D5

void init() {
    // PB1 konfiguriran kao izlaz (crvena LED dioda)
    pinMode(LED_CRVENO, OUTPUT);
    // PB2 konfiguriran kao izlaz (žuta LED dioda)
    pinMode(LED_ZUTO, OUTPUT);
    // PB3 konfiguriran kao izlaz (zelena LED dioda)
    pinMode(LED_ZELENO, OUTPUT);
    // tipkalo T1
```

```
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
    // tipkalo T3
    pinMode(TIPKALO3, INPUT_PULLUP); // PD5 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD5

}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }
    return 0;
}
```

U dosadašnjim vježbama stanje LED dioda kopiralo je stanje tipkala. Ako je tipkalo pritisnuto, LED dioda je uključena i obratno. U ovoj vježbi potrebno je pritisnuti i/ili otpustiti tipkalo, a stanje LED dioda mora ostati uključeno i/ili isključeno bez obzira što tipkalo nije pritisnuto cijelo vrijeme. Dakle, problem koji je potrebno realizirati u ovoj vježbi jest:

- uključiti zelenu LED diodu na padajući brid signala tipkala T1 (na pritisak tipkala T1),
- isključiti zelenu LED diodu na rastući brid signala tipkala T2 (na otpuštanje tipkala T2),
- na svaki padajući brid signala tipkala T3 (na pritisak tipkala T3) mijenja se stanje crvene i žute LED diode:
  1. crvena LED dioda je na početku uključena, a žuta LED dioda je isključena,
  2. na prvi i na svaki sljedeći neparni padajući brid signala tipkala T3 crvena LED dioda se isključuje, a žuta LED dioda se uključuje,
  3. na drugi i na svaki sljedeći parni padajući brid signala tipkala T3 crvena LED dioda se uključuje, a žuta LED dioda se isključuje.

Tipkala, za razliku od sklopke, nemaju dva stabilna stanja. Zato ih je do sada bilo potrebno držati cijelo vrijeme kako bi LED dioda bila uključena. No, ponekad je potrebno na pritisak tipkala (ili dodir tipke (engl. *touch button*)) uključiti neki uređaj koji mora nastaviti raditi kada otpustimo tipkalo. Za potrebe navedene funkcionalnosti potrebno je detektirati bridove signala. Signali imaju dva brida: rastući i padajući. Funkcije kojima se detektiraju rastući i padajući bridovi signala na digitalnom ulazu deklarirane su u zaglavlju `avrlib.h`. Nazivi funkcija i njihov opis jest:

- `isFallingEdge(xi);` - funkcija koja prima jedan argument, a služi za detekciju padajućeg brida. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina na kojem želimo provesti detekciju padajućeg brida. Funkcija vraća `bool` vrijednost: `true` ako se pojavio padajući brid (prijelaz iz visokog u nisko stanje), a `false` ako se nije pojavio padajući brid.

- `isRisingEdge(xi);` - funkcija koja prima jedan argument, a služi za detekciju rastućeg brida. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina na kojem želimo provesti detekciju rastućeg brida. Funkcija vraća `bool` vrijednost: `true` ako se pojavio rastući brid (prijelaz iz niskog u visoko stanje), a `false` ako se nije pojavio rastući brid.

Primjenu navedenih funkcija prikazat ćemo na nekoliko sljedećih primjera:

- `isFallingEdge(C5)` - provjera je li se dogodio padajući brid na digitalnom ulazu C5,
- `isFallingEdge(D2)` - provjera je li se dogodio padajući brid na digitalnom ulazu D2,
- `isRisingEdge(D4)` - provjera je li se dogodio rastući brid na digitalnom ulazu D4,
- `isRisingEdge(D5)` - provjera je li se dogodio rastući brid na digitalnom ulazu D5.

U ovoj vježbi koristit ćemo još jednu funkciju koju nismo do sada obradili, a deklarirana je u zaglavlju `avrlib.h`:

- `digitalToggle(xi);` - funkcija koja prima jedan argument, a služi za promjenu stanja digitalnog izlaza. Konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina na kojem želimo promijeniti stanje.

Funkcija `digitalToggle(B1)` promijenit će stanje digitalnog izlaznog pina PB1 (ako je pin bio u niskom stanju, nakon poziva ove funkcije bit će u visokom stanju i obrnuto). Primijetimo da funkcija `digitalToggle(B1)` ima istu namjenu kao makronaredba `toggle_pin(PORTB, PB1)`.

Definicije funkcija `isFallingEdge()`, `isRisingEdge()` i `digitalToggle()` nalaze se u datoteci `avrlib.cpp`. Preporučujemo da pogledate način na koji su napisana tijela navedenih funkcija.

U programskom kodu 5.16 u funkciji `init()` provedena je konfiguracija digitalnih ulaza i izlaza. U `while` petlju potrebno je dodati niz naredbi koje će na bridove signala tipkala mijenja stanje LED dioda. Padajući brid signala na digitalnom pinu javit će se u trenutku kada pritisnemo tipkalo jer će stanje pina iz visokog prijeći u nisko. Rastući brid signala na digitalnom pinu javit će se u trenutku kada otpustimo tipkalo jer će stanje pina iz niskog prijeći u visoko.

Padajući brid signala na tipkalu T1 ispitat ćemo uvjetovanim blokom `if (isFallingEdge(TIPKALO1)){}`. Unutar ovoga uvjetovanog bloka naredbi potrebno je uključiti zelenu LED diodu pozivom funkcije `digitalWrite(LED_ZELENO, HIGH);`. Rastući brid signala na tipkalu T2 ispitat ćemo uvjetovanim blokom `if (isRisingEdge(TIPKALO2)){}`. Unutar ovoga uvjetovanog bloka naredbi potrebno je isključiti zelenu LED diodu pozivom funkcije `digitalWrite(LED_ZELENO, LOW);`. Padajući brid signala na tipkalu T3 ispitat ćemo uvjetovanim blokom `if (isFallingEdge(TIPKALO3)){}`. Unutar ovoga uvjetovanog bloka naredbi potrebno je mijenjati stanja crvene i žute LED diodu pozivom funkcija `digitalToggle(LED_CRVENO);` i `digitalToggle(LED_ZUTO);`. S obzirom da se crvena i žuta LED dioda uključuju naizmjenično, neposredno nakon poziva `init()` funkcije potrebno je inicijalno uključiti crvenu LED diodu pozivom funkcije `digitalWrite(LED_CRVENO, HIGH);`. Navedeno je potrebno implementirati u programski kod 5.16. Implementacija programa koji na bridove signala tipkala mijenja stanje LED dioda prikazan je programskim kodom 5.17.

Programski kod 5.17: Sadržaj datoteke `vjezba55.cpp` - program koji na bridove signala tipkala mijenja stanje LED dioda

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_CRVENO B1
#define LED_ZUTO B2
#define LED_ZELENO B3
#define TIPKALO1 D4
#define TIPKALO2 D2
#define TIPKALO3 D5
```

```
void init() {
    // PB1 konfiguriran kao izlaz (crvena LED dioda)
    pinMode(LED_CRVENO, OUTPUT);
    // PB2 konfiguriran kao izlaz (žuta LED dioda)
    pinMode(LED_ZUTO, OUTPUT);
    // PB3 konfiguriran kao izlaz (zelena LED dioda)
    pinMode(LED_ZELENO, OUTPUT);
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
    // tipkalo T3
    pinMode(TIPKALO3, INPUT_PULLUP); // PD5 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD5

}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    // pocetno uključena crvenu LED diodu
    digitalWrite(LED_CRVENO, HIGH);
    while (1) { // beskonačna petlja
        // provjera padajućeg brida tipkala T1
        if (isFallingEdge(TIPKALO1)) {
            digitalWrite(LED_ZELENO, HIGH); // uključi zelenu LED diodu
        }
        // provjera rastućeg brida tipkala T2
        if (isRisingEdge(TIPKALO2)) {
            digitalWrite(LED_ZELENO, LOW); // isključi zelenu LED diodu
        }
        // provjera padajućeg brida tipkala T3
        if (isFallingEdge(TIPKALO3)) {
            digitalToggle(LED_CRVENO); // promijeni stanje crvene LED diode
            digitalToggle(LED_ZUTO); // promijeni stanje žute LED diode
        }
    }
    return 0;
}
```

Prevedite datoteku `vjezba55.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili gore navedene korake, zelena LED dioda uključit će se pritiskom tipkala T1 i ostati uključena i nakon otpuštanja tipkala T1. Zelena LED dioda isključit će se nakon što tipkalo T2 bude otpušteno. Naravno, prvo ga morate pritisnuti. Crvena i žuta LED dioda mijenjat će svoja stanja na svaki pritisak tipkala T3.

Zatvorite datoteku `vjezba55.cpp` i onemogućite prevođenje ove datoteke.

Vježba 5.6

Napišite program koji će:

- na padajući brid signala tipkala T1 osigurati trčanje LED dioda u desno. Promjena stanja LED dioda mora se desiti pritiskom na tipkalo. Redoslijed trčanja LED dioda u desno je crvena → žuta → zelena → crvena → žuta → ...
- na padajući brid signala tipkala T2 osigurati trčanje LED dioda u lijevo. Promjena stanja

LED dioda mora se desiti pritiskom na tipkalo. Redoslijed trčanja LED dioda u lijevo je crvena → zelena → žuta → crvena → zelena → ...

- na padajući brid signala tipkala T3 resetirati trčanje LED dioda tako da se uključi crvena LED dioda, a isključe žuta i zelena.

Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, žuta LED dioda na digitalni izlaz PB2, zelena LED dioda na digitalni izlaz PB3, tipkalo T1 na digitalni ulaz PD4, tipkalo T2 na digitalni ulaz PD2, a tipkalo T3 na digitalni ulaz PD5 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba56.cpp`. Omogućite prevođenje datoteke `vjezba56.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba56.cpp` prikazan je programskim kodom 5.18.

Programski kod 5.18: Početni sadržaj datoteke `vjezba56.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"

void init() {

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    int korak = 1; // 1 = crvena, 2 = zuta, 3 = zelena

    while (1) { // beskonacna petlja
        // provjera padajuceg brida tipkala T1
        if (isFallingEdge(TIPKALO1)) {
            korak++; // korak u desno
            if (korak == 4) {
                korak = 1;
            }
        }
        // provjera padajuceg brida tipkala T2
        if (isFallingEdge(TIPKALO2)) {
            korak--; // korak u lijevo
            if (korak == 0) {
                korak = 3;
            }
        }
        // provjera padajuceg brida tipkala T3
        if (isFallingEdge(TIPKALO3)) {
            korak = 1; //reset na pocetak
        }

        switch (korak) {
            case 1:
            digitalWrite(LED_CRVENO, HIGH); // crvena LED dioda on
            digitalWrite(LED_ZUTO, LOW); // zuta LED dioda off
            digitalWrite(LED_ZELENO, LOW); // zelena LED dioda off
            break;
            case 2:
            digitalWrite(LED_CRVENO, LOW); // crvena LED dioda off
            digitalWrite(LED_ZUTO, HIGH); // zuta LED dioda on
            digitalWrite(LED_ZELENO, LOW); // zelena LED dioda off
            break;
```

```
            case 3:
            digitalWrite(LED_CRVENO, LOW); // crvena LED dioda off
            digitalWrite(LED_ZUTO, LOW); // zuta LED dioda off
            digitalWrite(LED_ZELENO, HIGH); // zelena LED dioda on
            break;
            default:
            break;
        }
    }
    return 0;
}
```

U programskom kodu 5.18 konfigurirajte potrebne digitalne ulaze i izlaze. S obzirom na iskustvo prethodnih vježbi, ovdje ćemo preskočiti objašnjenje liniju po liniju programskog koda. Prije same konfiguracije, napravite simbolička imena kao u prethodnim vježbama (`LED_CRVENO`, `LED_ZUTO`, `LED_ZELENO`, ...). Rješenje ove vježbe prikazano je programskim kodom 5.19.

Za potrebe trčanja LED dioda koristit ćemo cjelobrojnu varijablu `korak` koja poprima sljedeće vrijednosti:

- `korak = 1` - svijetli samo crvena LED dioda,
- `korak = 2` - svijetli samo žuta LED dioda,
- `korak = 3` - svijetli samo zelena LED dioda.

Signale s tipkala koristit ćemo za upravljanje varijablom `korak` na sljedeći način:

- na padajući brid signala tipkala T1 varijabla `korak` se mora povećati za 1 (`korak++`),
- na padajući brid signala tipkala T2 varijabla `korak` se mora smanjiti za 1 (`korak--`),
- na padajući brid signala tipkala T3 varijabla `korak` mora postati 1 (`korak = 1`).

Ako varijabla `korak` postane 4, tada bi je trebalo vratiti na vrijednost 1 kako bi se osiguralo izmjenjivanje koraka redoslijedom 1, 2, 3, 1, 2, 3, 1 ... . Ako varijabla `korak` postane 0, tada bi je trebalo vratiti na vrijednost 3 kako bi se osiguralo izmjenjivanje koraka redoslijedom 3, 2, 1, 3, 2, 1, 3 ... . Korištenjem varijable `korak` i `switch case` uvjetovanog grananja realizirano je trčanje LED dioda u oba smjera. Sve navedeno prikazano je programskim kodom 5.19.

Programski kod 5.19: Program za trčanje LED dioda na padajući brid tipkala

```
#include <avr/io.h>
#include "AVR/avrlib.h"

#define LED_CRVENO B1
#define LED_ZUTO B2
#define LED_ZELENO B3
#define TIPKALO1 D4
#define TIPKALO2 D2
#define TIPKALO3 D5

void init() {
    // PB1 konfiguriran kao izlaz (crvena LED dioda)
    pinMode(LED_CRVENO, OUTPUT);
    // PB2 konfiguriran kao izlaz (zuta LED dioda)
    pinMode(LED_ZUTO, OUTPUT);
    // PB3 konfiguriran kao izlaz (zelena LED dioda)
    pinMode(LED_ZELENO, OUTPUT);
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
```

```
    // ukljucen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // ukljucen pull up otpornik na pinu PD2
    // tipkalo T3
    pinMode(TIPKALO3, INPUT_PULLUP); // PD5 konfiguriran kao ulaz
    // ukljucen pull up otpornik na pinu PD5

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    int korak = 1; // 1 = crvena, 2 = zuta, 3 = zelena

    while (1) { // beskonacna petlja
        // provjera padajuceg brida tipkala T1
        if (isFallingEdge(TIPKALO1)) {
            korak++; // korak u desno
            if (korak == 4) {
                korak = 1;
            }
        }
        // provjera padajuceg brida tipkala T2
        if (isFallingEdge(TIPKALO2)) {
            korak--; // korak u lijevo
            if (korak == 0) {
                korak = 3;
            }
        }
        // provjera padajuceg brida tipkala T3
        if (isFallingEdge(TIPKALO3)) {
            korak = 1; //reset na pocetak
        }

        switch (korak) {
            case 1:
            digitalWrite(LED_CRVENO, HIGH); // crvena LED dioda on
            digitalWrite(LED_ZUTO, LOW); // zuta LED dioda off
            digitalWrite(LED_ZELENO, LOW); // zelena LED dioda off
            break;
            case 2:
            digitalWrite(LED_CRVENO, LOW); // crvena LED dioda off
            digitalWrite(LED_ZUTO, HIGH); // zuta LED dioda on
            digitalWrite(LED_ZELENO, LOW); // zelena LED dioda off
            break;
            case 3:
            digitalWrite(LED_CRVENO, LOW); // crvena LED dioda off
            digitalWrite(LED_ZUTO, LOW); // zuta LED dioda off
            digitalWrite(LED_ZELENO, HIGH); // zelena LED dioda on
            break;
            default:
            break;
        }
    }
    return 0;
}
```

Prevedite datoteku `vjezba56.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili gore navedene korake, LED diode će trčati u desno pritiskom na tipkalo T1, u lijevo pritiskom na tipkalo T2 i resetirat će trčanje pritiskom na tipkalo T3.

Zatvorite datoteku `vjezba56.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 6

LCD displej

LCD (engl. *Liquid Crystal Display*) displej koristi se za prikazivanje statusa i parametara elektroničkih sustava zasnovanih na mikroračunalima. U kombinaciji s tipkalima, potenciometrima i rotacijskim enkoderima, LCD može služiti za postavljanje parametara sustava. Nadalje, LCD se može primijeniti za prikaz procesnih varijabli kao što su temperatura, vlaga, tlak, masa, sila itd.

Za potrebe ovog udžbenika koristit ćemo LCD displej GDM1602A. Ovaj LCD displej se na razvojno okruženje s mikroupravljačem ATmega328P prikazanom na slici 2.1 povezuje konektorom i kabelom. LCD displej GDM1602A prikazuje znakove u dva retka. U svakom retku moguće je prikazati ukupno 16 znakova. Programska memorija LCD displeja omogućuje zapis 40 znakova, no samo 16 znakova je vidljivo. Korištenjem funkcija LCD displeja za pomak teksta u lijevo ili desno, svih 40 znakova može se prikazati na 16 vidljivih polja. Svaki znak dimenzije je 5x8 točaka (piksela). LCD displej GDM1602A ima ugrađeni mikroupravljač S6A0069 koji se brine o prikazu znakova koji pristižu na komunikacijske pinove LCD displeja. Biblioteka koju ćemo koristiti u vježbama za ispis znakova dizajnirana je za LCD ugrađeni mikroupravljač S6A0069 ili njemu ekvivalentan. Na LCD displejima s drugačijim mikroupravljačima ova biblioteka vrlo vjerojatno neće raditi. LCD displej omogućuje prikaz znakova iz donje polovica ASCII tablice. Ostali znakovi koje LCD može ispisati ovise o mikroupravljači LCD displeja. Slova kao što su č, ć, đ, š i ž (koja spadaju u kategoriju posebnih znakova) mogu se definirati i pohraniti u memoriju LCD displeja. Informacije o LCD displeju GDM1602A možete pronaći u literaturi [3].

6.1 Vježbe - LCD displej

Shema povezivanja LCD displeja s mikroupravljačem ATmega328P na razvojnom okruženju sa slike 2.1 prikazana je na slici 6.1. Primijetit ćete na shemi da radi preglednosti nismo prikazali LED diode, no one su i dalje spojene na iste pinove kako je prikazano shemom na slici 4.1. Elektronička komponenta koja je nova u ovoj vježbi jest LCD displej. Prijenos podataka između mikroupravljača i LCD displeja može biti putem 4-bitne i 8-bitne podatkovne sabirnice. S obzirom da se način komunikacije zbog perzistencije vida (tromosti oka) ne odražava na kvalitetu prikaza, mi ćemo koristiti 4-bitni način rada LCD displeja zbog uštede digitalnih pinova.

Za prijenos podataka u 4-bitnom načinu rada, pinove mikroupravljača potrebno je spojiti na pinove LCD displeja D4, D5, D6 i D7. Pri tome, za prijenos podataka možete koristiti bilo koja četiri digitalna pina mikroupravljača. Na razvojnom okruženju s mikroupravljačem ATmega328P, podatkovni pin LCD displeja D4 spojen je na pin mikroupravljača PC3, D5 na pin PC2, D6 na pin PC1 i D7 na pin PB0 (slika 6.1). Osim podatkovnih pinova, LCD displej ima tri upravljačka pina: RS, E i R/W. Upravljački pin R/W služi za osiguranje dvosmjerne

komunikacije s LCD displejom. Ovaj pin omogućuje da se na LCD zapisuju podaci (R/W = nisko stanje) te da se s LCD displeja čitaju podaci (R/W = visoko stanje). S obzirom da se čitanje podataka s LCD displeja u praksi gotovo nikada ne koristi, upravljački pin R/W spojen je na GND (slika 6.1) kako bi se na LCD displej mogli samo zapisivati podaci te isti prikazivati. Pomoću upravljačkog pina RS omogućuje se odabir memorije LCD displeja kojoj će se pristupiti, dok se upravljačkim pinom E odobrava upis podataka koji se trenutno nalaze na podatkovnoj sabirnici. Upravljački pin RS omogućuje pristup memorijama:

- DDRAM (engl. *Display Data RAM*) - memorija koja pamti znakove koje je potrebno prezentirati na LCD displeju veličine 80 B (80 znakova).

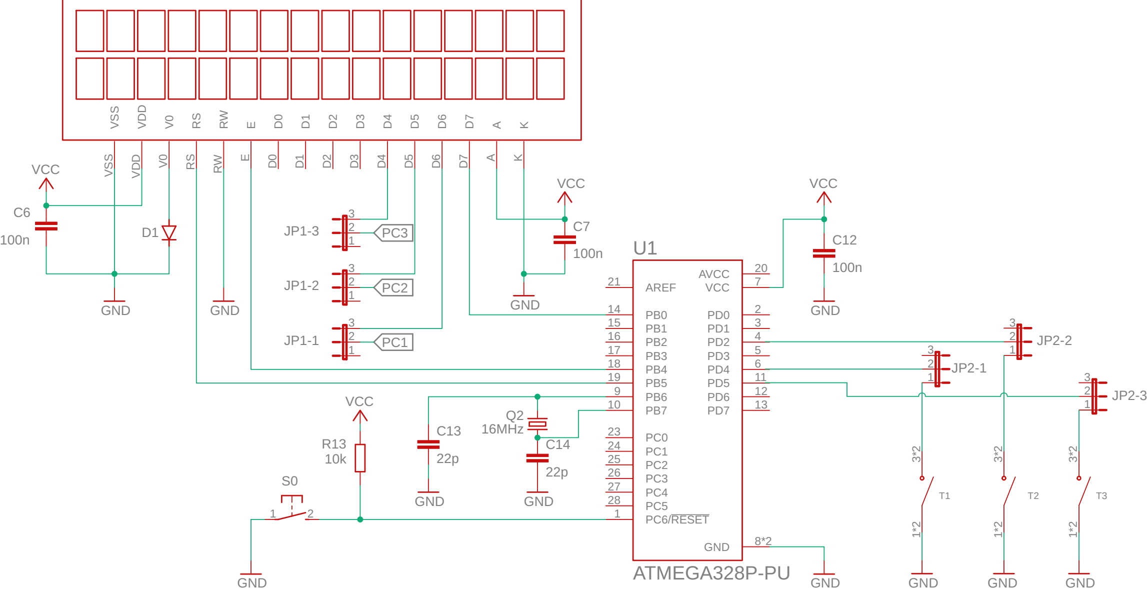
- CGRAM (engl. *Character Generator RAM*) - memorija u koju se mogu pohraniti vlastiti znakovi kao č, š, ć itd. Ova memorija omogućuje pohranu 8 znakova koji su veličine 5 x 8 piksela.

Upravljački pin LCD displeja RS spojen je na pin mikroupravljača PB5, a upravljački pin E na pin PB4 (slika 6.1).

Kako smo već naveli, pinovi LCD displeja D4, D5, D6, D7, RS i E mogu se spojiti na bilo koji dostupni digitalni pin mikroupravljača, što ovisi o zauzeću ostalih pinova. S obzirom da komunikacija s LCD displejom ne zahtjeva korištenje namjenskih pinova mikroupravljača, odluku o zauzeću pinova za LCD displej možete donijeti na samom kraju povezivanja elektroničkih komponenti na mikroupravljač. Bilo koji slobodan digitalni pin može se upotrijebiti za komunikaciju s LCD displejem. Na displeju se još nalaze sljedeći pinovi:

- A i K - služe za pozadinsko osvjetljenje i spajaju se na izvor napona 5 VDC (slika 6.1),

- V0 - služi za promjenu kontrasta (intezitet piksela znakova). Ovaj pin je interno pritegnut na VCC pomoću priteznog otpornika. Taj napon "srušen" je na prema shemi na slici 6.1 s običnom diodom kako bi se prema tehničkoj specifikaciji ostvario dostatan kontrast. Na pin V0 često se spaja potenciometar s kojim je moguće ugađati kontrast.



Slika 6.1: Shema povezivanja LCD displeja s mikroupravljačem ATmega328P na razvojnom okruženju sa slike 2.1

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `LCD displej.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `LCD displej.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\LCD displej\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `LCD displej`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

Prije korištenja LCD displeja na razvojnom okruženju s mikroupravljačem, potrebno je konfigurirati pinove koji su spojeni na pinove LCD displeja D4, D5, D6, D7, RS i E. U otvorenom projektu nalazi se mapa `LCD` u kojoj se nalazi zaglavlje `lcd.h`. Otvorite zaglavlje `lcd.h`. Konfiguracija pinova mikroupravljača koji su spojeni na pinove LCD displeja u zaglavlju `lcd.h` prikazana je programskim kodom 6.1.

Programski kod 6.1: Konfiguracija pinova mikroupravljača koji su spojeni na pinove LCD displeja u zaglavlju `lcd.h`

```
// korisnik mijenja samo konfiguraciju displeja pri korištenju ove biblioteke
// konfiguracija LCD displeja
#define LCD_D4_PIN       C3 // D4 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D5_PIN       C2 // D5 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D6_PIN       C1 // D6 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D7_PIN       B0 // D7 na LCD displeju - 4 bita komunikacija - pin

#define LCD_RS_PIN       B5 // RS na LCD displeju - pin za odabir registra
#define LCD_EN_PIN       B4 // E na LCD displeju - pin za odabir registra

#define LCD_LINES        2  // broj redova na LCD displeju
#define LCD_DISP_LENGTH 16 // broj znakova (vidljivih) u redu na LCD displeju
// kraj konfiguracije LCD displeja
```

Ako u vlastitim projektima koji uključuju korištenje LCD displeja preskočite konfiguraciju prikazanu programskim kodom 6.1, LCD displej neće raditi ispravno. Programski kod 6.1 prikazuje definirane konstante koje se koriste u ostatku zaglavlja `lcd.h` te u datoteci `lcd.cpp`. Pinovi mikroupravljača koji su povezani na podatkovne pinove LCD displeja konfiguriraju se na sljedeći način:

- `#define LCD_D4_PIN xi` - konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja D4,
- `#define LCD_D5_PIN xi` - konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja D5,
- `#define LCD_D6_PIN xi` - konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja D6,
- `#define LCD_D7_PIN xi` - konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja D7.

Pinovi mikroupravljača koji su povezani na upravljačke pinove LCD displeja konfiguriraju se na sljedeći način:

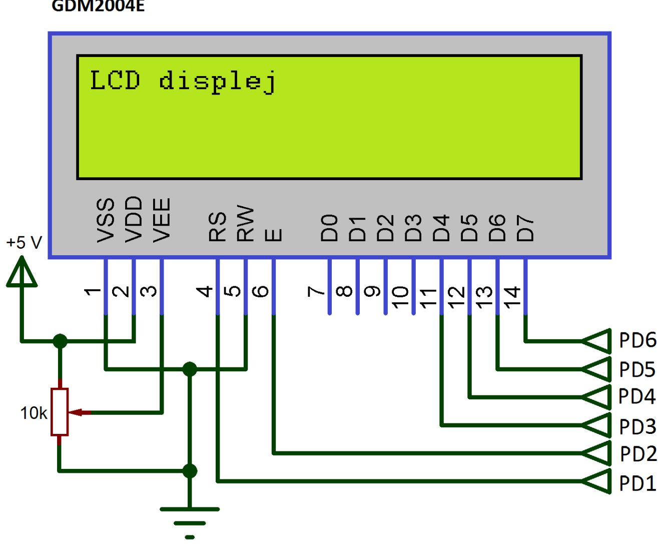
- `#define LCD_RS_PIN Pxi` - konstanta `xi` (x = B, C, D, i = 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja RS,
- `#define LCD_EN_PIN Pxi`, (x = A, B, C, D; i = 0,1,...,7) - konstanta `xi` (x = B, C, D, i

= 0, 1, 2, ..., 7) predstavlja ime porta i poziciju pina mikroupravljača koji je spojen na pin LCD displeja E.

Broj redaka LCD displeja i broj znakova u jednom retku konfiguriraju se pomoću sljedećih konstanti:

- `#define LCD_LINES x` - broj vidljivih redaka LCD displeja (najčešće je x = 1, 2, 4),
- `#define LCD_DISP_LENGTH y` - broj vidljivih znakova u jednom retku LCD displeja (najčešće je y = 10, 16, 20).

Primjer povezivanja LCD displeja GDM2004E na D port mikroupravljača ATmega328P prikazan je na slici 6.2.

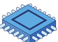


Slika 6.2: Primjer povezivanja LCD displeja GDM2004E na D port mikroupravljača ATmega328P

Prema kataloškom broju GDM2004E, ovaj LCD displej ima 4 retka i 20 vidljivih znakova u jednom retku. Podatkovni i upravljački pinovi LCD displeja spojeni su na port D mikroupravljača ATmega328P prema shemi slici 6.2. Konfiguracija LCD displeja u zaglavlju `lcd.h` za ovaj LCD displej prikazana je u programskom kodu 6.2.

Programski kod 6.2: Konfiguracija pinova mikroupravljača koji su spojeni na pinove LCD displeja prema shemi na slici 6.2 u zaglavlju `lcd.h`

```
// korisnik mijenja samo konfiguraciju displeja pri korištenju ove biblioteke
// konfiguracija LCD displeja
#define LCD_D4_PIN      D3 // D4 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D5_PIN      D4 // D5 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D6_PIN      D5 // D6 na LCD displeju - 4 bita komunikacija - pin
#define LCD_D7_PIN      D7 // D7 na LCD displeju - 4 bita komunikacija - pin

#define LCD_RS_PIN      D1 // RS na LCD displeju - pin za odabir registra
#define LCD_EN_PIN      D2 // E na LCD displeju - pin za odabir registra

#define LCD_LINES       4  // broj redova na LCD displeju
#define LCD_DISP_LENGTH 20 // broj znakova (vidljivih) u redu na LCD displeju
// kraj konfiguracije LCD displeja
```

 Vježba 6.1

Napišite program koji će svakih tri sekunde izmjenjivati sljedeća dva prikaza na LCD displeju:

- Vaše ime i prezime bez dijakritičkih znakova lijevo poravnato u dva retka (ime u prvom retku, prezime u drugom retku),
- tekst `Mikroupravljac ATmega328P` centrirano u dva retka.

Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba61.cpp`. Omogućite prevođenje datoteke `vjezba61.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba61.cpp` prikazan je programskim kodom 6.3.

Programski kod 6.3: Početni sadržaj datoteke `vjezba61.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include <util/delay.h>

void init() {

}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Problem koji moramo riješiti u ovoj vježbi je ispis dvije vrste različito poravnatog teksta svakih 3 sekunde na LCD displej. Prije nego krenemo pisati program, potrebno je proučiti funkcije koje se koriste za ispis i formatiranje teksta na LCD displeju. Funkcije koje se koriste za ispisivanje i formatiranje teksta na LCD displej deklarirane su u zaglavlju `lcd.h`, a definirane u datoteci `lcd.cpp`.

U programskom kodu 6.3 uključili smo zaglavlje `lcd.h` u datoteku koja se prevodi u strojni kod naredbom `#include "LCD/lcd.h"`. Funkcije koje se koriste za rad s LCD displejm su:

- `lcdInit()` - funkcija kojom se konfigurira rad LCD displeja. Ova funkcija konfigurira izlazne pinove mikroupravljača koji su povezani na podatkovne i upravljačke pinove LCD dipleja, briše znakove s ekrana i postavlja kursor u prvi redak i prvi stupac LCD displeja. Način komunikacije koji se konfiguira ovom funkcijom je 4-bitni.
- `lcdClrScr()` - funkcija kojom se brišu svi znakovi na LCD displeju, a zatim se kursor postavi na početak prvog reda.
- `lcdHome()` - funkcija kojom se kursor postavlja na početak prvog retka (prvi redak i prvi stupac).
- `lcdGotoXY(x, y)` - funkcija koja prima dva argumenta. Prvi argument postavlja kursor u

redak x, a drugi argument postavlja kursor u stupac y. Prvi redak ima indeks $x = 1$, a prvi stupac ima indeks $y = 1$. Za LCD displej sa sheme na slici 6.1 najveći indeks retka jest $x = 2$, a najveći indeks stupca za vidljivi dio ispisa jest $y = 16$.

- `lcdChar(c)` - funkcija koja prima argument c koji predstavlja ASCII kod znaka koji će se ispisati na LCD displej.
- `lcdprintf(const char *format, arg1, arg2, ...)` - funkcija koja služi za ispis teksta na LCD displej. Sintaksa funkcije `lcdprintf()` ista je sintaksi funkcije `printf()`[1] koja pripada C standardnoj biblioteci `stdio.h`. Za argumente funkcije vrijedi:
  - `const char *format` - string u funkciji `lcdprintf()` koji sadrži tekst koji će se ispisati na LCD displej. String može sadržavati ugrađene oznake formata (formate ispisa (engl. *format tags*)) koji se zamjenjuju tekstualnim zapisom liste argumenata `arg1, arg2, ...` u zadanom formatu.

Pažljivi čitatelj će primijetiti da su sve funkcije pisane stilom *camelCase* [2] osim funkcije `lcdprintf()`. Razlog tomu je što smo htjeli da ta funkcija poprimi naziv kao funkcije `sprintf()` i `fprintf()` koje su srodne funkciji `printf()`, ali formatirani tekst zapisuju u string ili datoteku. Funkcija `lcdprintf()` ispisuje formatirani string na LCD displej.

Primjenu funkcije `lcdprintf()` prikazat ćemo na nekoliko sljedećih primjera:

- `lcdprintf("Hello, World!");` - ispisat će na LCD displeju tekst `Hello, World!` u prvom retku i počevši od prvog stupca.
- `lcdprintf("Prikaz na LCD displeju");` - ispisat će na LCD displeju tekst `Prikaz na LCD di` u prvom retku i počevši od prvog stupca. Tekst je predugačak i ne može se prikazati sa 16 znakova u jednom retku.
- `lcdprintf("Prikaz na LCD\ndispleju");` - ispisat će u prvom retku LCD displeja tekst `Prikaz na LCD`, a u drugom retku tekst `displeju`. Zbog specijalnog znaka `'\n'` (engl. *newline*) ispis znakova nakon njega ide u novi redak.
- `lcdprintf("Cijeli broj %d", 314);` - ispisat će na LCD displeju tekst `Cijeli broj 314` u prvom retku i počevši od prvog stupca. Na mjesto specifikatora ispisa `%d` ispisat će se drugi argument funkcije `lcdprintf()`, odnosno broj 314.
- `lcdprintf("Pi je %.2f!", 3.14159265);` - ispisat će na LCD displeju tekst `Pi je 3.14!` u prvom retku i počevši od prvog stupca. Na mjesto specifikatora ispisa `%.2f` ispisat će se drugi argument funkcije `lcdprintf()` na dva decimalna mjesta, odnosno broj 3.14.

Sada kada smo opisali ključne funkcije za ispis i formatiranje teksta na LCD diplej, možemo napraviti program kojim će se ostvariti funkcionalnost definirana ovom vježbom.

U funkciji `init()` u datoteci `vjezba61.cpp` potrebno je najprije inicijalizirati rad LCD displeja. Pozovite u funkciji `init()` funkciju `lcdInit()`. Nakon inicijalizacije LCD displeja napravit ćemo dio programa kojim se ispisuje tekst na LCD displej. S obzirom da se dva teksta trebaju izmjenjivati svakih tri sekunde, ispis dvije vrste teksta ćemo napraviti u beskonačnoj `while` petlji s kašnjenjima od 3000 ms. Kao ime i prezime koje ćemo ispisati na LCD displej

[1] C library function - printf(), `https://www.tutorialspoint.com/c_standard_library/c_function_printf.htm`

[2] Stil u kojem se nazivi identifikatora nazivaju tako da se prva riječ u identifikatoru piše malim početnim slovom, a svaka sljedeća velikim početnim slovom. Na primjer: `ovoJeFunkcijaZaIspisNaLcd()`

odabrali smo Marija Horvat[3]. Vi ćete ispisati svoje ime i prezime bez dijakritičkih znakova. U beskonačnu petlju `while` napišite sljedeće linije programskog koda:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("Marija\n");` - ispisivanje imena `Marija` na početku prvog retka (lijevo poravnato). Dodatno, na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis. Kada ne bismo koristili specijalni znak `'\n'` , potrebno bi bilo pozvati funkciju `lcdGotoXY(2, 1)` za postavljanje kursora na početak drugog retka. Vi ćete upisati svoje ime.
- `lcdprintf("Horvat");` - ispisivanje prezimena `Horvat` na početku drugog retka (lijevo poravnato). Važno je naglasiti da ovaj ispis ne briše prethodni sadržaj na LCD displeju, već osvježava tekst od pozicije gdje je bio postavljen kursor. Vi ćete upisati svoje prezime.
- `_delay_ms(3000);` - kašnjenje 3 sekunde.

Tekst `Mikroupravljac ATmega328P` ima ukupno 25 znakova. Prema tome, ovaj tekst ne stane u jedan redak koji može prikazati samo 16 znakova. Iz tog razloga ćemo ga zapisati u dva retka i promatrati kao dva odvojena teksta: `Mikroupravljac` i `ATmega328P`.

Tekst `Mikroupravljac` ima ukupno 14 znakova, a potrebno ga je ispisati u sredini retka (centrirano). Redak na LCD displeju ima 16 vidljivih znakova, pa je u ovom slučaju pri ispisu teksta `Mikroupravljac` potrebno ostaviti jedno prazno mjesto s lijeve i jedno prazno mjesto s desne strane teksta u prvom retku ((16 - 14) : 2 = 1). Početak ispisa teksta `Mikroupravljac` bit će 2. stupac u 1. retku.

Tekst `ATmega328P` ima ukupno 10 znakova, a potrebno ga je također ispisati u sredini retka (centrirano). U ovom slučaju pri ispisu teksta `ATmega328P` potrebno je ostaviti tri prazna mjesta s lijeve i tri prazna mjesto s desne strane teksta u drugom retku ((16 - 10) : 2 = 3). Početak ispisa teksta `ATmega328P` bit će 4. stupac u 2. retku.

U beskonačnu petlju `while` nastavno na prethodne linije koda napišite sljedeće:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdGotoXY(1,2);` - pozicioniranje kursora za ispis teksta `Mikroupravljac` u 1. redak i 2. stupac.
- `lcdprintf("Mikroupravljac");` - ispisivanje riječi `Mikroupravljac` na LCD displeju.
- `lcdGotoXY(2,4);` - pozicioniranje kursora za ispis teksta `ATmega328P` u 2. redak i 4. stupac.
- `lcdprintf("ATmega328P");` - ispisivanje riječi `ATmega328P` na LCD displeju.
- `_delay_ms(3000);` - kašnjenje 3 sekunde.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba61.cpp` treba biti ista kao programski kod 6.4.

[3]Marija je najčešće žensko ime u RH u posljednjih 90 godina. Horvat je najčešće prezime u RH.

Programski kod 6.4: Sadržaj datoteke `vjezba61.cpp` - ispis imena i prezimena te teksta `Mikroupravljac ATmega328P` na LCD displej svakih 3 sekunde

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include <util/delay.h>
// s obzirom da se #include <util/delay.h> nalazi u lcd.h,
//ova se linija smije izostaviti

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("Marija\n");
        lcdprintf("Horvat");
        _delay_ms(3000); // kašnjenje 3000 ms

        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        lcdGotoXY(1,2); // pozicioniranje kursora u 1. red, 2. stupac
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("Mikroupravljac");
        lcdGotoXY(2, 4); // pozicioniranje kursora u 2. red, 4. stupac
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("ATmega328P");
        _delay_ms(3000); // kašnjenje 3000 ms
    }

    return 0;
}
```

Prevedite datoteku `vjezba61.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se Vaše ime i prezime lijevo poravnato koje će se zadržati 3 sekunde, a zatim tekst `Mikroupravljac ATmega328P` ispisan u dva retka i centriran koji će se također zadržati 3 sekunde. Ovi tekstovi će se neprestano izmjenjivati.

Ispis teksta `Mikroupravljac ATmega328P` se može provesti i na drugačiji način. Umjesto da koristimo funkciju `lcdGotoXY()` za pomicanje kursora u desno, možemo tekst ispisivati s vodećim praznim mjestima. U beskonačnu petlju `while` ispis teksta `Mikroupravljac ATmega328P` implementirajte na sljedeći način:

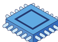
- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf(" Mikroupravljac\n");` - ispisivanje riječi `Mikroupravljac` na LCD displeju s dodanim jednim praznim mjestom ispred teksta (dodan je razmak ((engl. *white space*))).
- `lcdprintf("   ATmega328P");` - ispisivanje riječi `ATmega328P` na LCD displeju s tri dodana prazna mjesta ispred teksta.
- `_delay_ms(3000);` - kašnjenje 3 sekunde.

Ponovno prevedite datoteku `vjezba61.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Primijetite da nema razlike u ispisu u odnosu na programski kod 6.4.

Pokušajte ispisati Vaše ime i prezime tako da ga desno poravnate na LCD displeju.

Zatvorite datoteku `vjezba61.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 6.2

Napišite program koji će svakih dvije sekunde izmjenjivati sljedeća tri prikaza na LCD displeju:

- u prvom retku ispišite tekst `int:` i sadržaj varijable tipa `int`, a u drugom retku tekst `float:` i sadržaj varijable tipa `float` na 3 decimalna mjesta.
- u prvom retku ispišite tekst `int8:` i sadržaj varijable tipa `int8_t`, a u drugom retku tekst `uint8:` i sadržaj varijable tipa `uint8_t`.
- u prvom retku ispišite tekst `int32:` i sadržaj varijable tipa `int32_t`, a u drugom retku tekst `uint16:` i sadržaj varijable tipa `uint16_t`.

Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba62.cpp`. Omogućite prevođenje datoteke `vjezba62.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba62.cpp` prikazan je programskim kodom 6.5.

Programski kod 6.5: Početni sadržaj datoteke `vjezba62.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        _delay_ms(2000); // kašnjenje 2000 ms


        _delay_ms(2000); // kašnjenje 2000 ms


        _delay_ms(2000); // kašnjenje 2000 ms
    }

    return 0;
}
```

Cilj ove vježbe nije samo ispisati podatke koji se nalaze u varijablama na LCD displej, već se upoznati s dostupnim cjelobrojnim i realnim tipovima podataka pri programiranju mikroupravljača ATmega328P. Ako ste dobar poznavatelj programskog jezika C i programirali ste aplikacije za računalo, tada sigurno znate da cjelobrojni tip `int` zauzima 32 bita podatkovne memorije. Isto tako, realni tip `double` zauzima 64 bita podatkovne memorije. Poznato je kako je cjelobrojni tip `int` definiran kao tip koji ima najmanje 16 bitova, a širina tipa `int` ovisi

o platformi na kojoj se izvršava programski kod. Kod AVR 8-bitnih mikroupravljača veličina cjelobrojnih i realnih tipova podataka ipak se razlikuje od one na koju smo navikli programirajući aplikacije za računala. Na primjer, na mikroupravljaču ATmega328P cjelobrojni tip `int` zauzima 16 bita podatkovne memorije. O čemu ovisi veličina tipa podataka? Ovisi o prevoditelju koji prevodi programski kod u strojni kod za računalo ili mikroračunalo. Kada programirate bilo koji mikroupravljač, važno je saznati kako prevoditelj gleda na veličinu cjelobrojnih i realnih podataka. U suprotnom, vrlo je lako proizvesti preljev (engl. *overflow*) ili podljev (engl. *underflow*) podataka što može dovesti do neželjenog ponašanja sustava.

Tipovi podataka koje koristi prevoditelj u programskom razvojnom okruženju *Microchip Studio* za 8-bitne AVR mikroupravljače prikazani su u tablici 6.1.

Tablica 6.1: Tipovi podataka koje koristi prevoditelj u programskom razvojnom okruženju *Microchip Studio* za 8-bitne AVR mikroupravljače

Izvedeni tip <stdint.h>	Standardni tip podatka	Broj bitova	Min.	Max.	printf format
-	`char`	8	-128	127	%c
`int8_t`	`signed char`	8	-128	127	%d
`uint8_t`	`unsigned char`	8	0	255	%u
`int16_t`	`int` `signed int`	16	-32768	32767	%d
`uint16_t`	`unsigned int`	16	0	65535	%u
`int32_t`	`long int` `signed long int`	32	-2147483648	2147483647	%ld
`uint32_t`	`unsigned long int`	32	0	4294967295	%lld
`int64_t`	`long long`	64	-2^{63}	2^{63} - 1	%lu
`uint64_t`	`unsigned long long`	64	0	2^{64}	%llu
-	`float`	32	1.175494×10^{-38}	3.402823×10^{38}	%f
-	`double`	32	1.175494×10^{-38}	3.402823×10^{38}	%f
-	`long double`	64	$2.225074 \times 10^{-308}$	1.797693×10^{308}	%lf

Dobra navika jest da pri deklaraciji varijabli koristite izvedene tipove podataka koji se nalaze u zaglavlju `stdint.h`. Korištenje izvedenih tipova podataka iz zaglavlja `stdint.h` omogućava portabilnost koda bez obzira na platformu (računalo, mikroupravljač, mikroprocesor, ...) na kojoj se program izvršava. Izvedeni tipovi cjelobrojnih podataka prikazani su u prvom stupcu tablice 6.1. Korištenje izvedenih tipova podataka intuitivno je jer u imenu tipa podataka stoji informacija o veličini podataka. Na primjer, `int32_t` cijeli je broj s predznakom širine 32 bita, dok je `uint8_t` cijeli broj bez predznaka širine 8 bitova. Isto se indukcijom može zaključiti za ostale cjelobrojne podatke s ili bez predznaka veličine 8, 16, 32 i 64 bita.

U tablici 6.1 prikazani su svi izvedeni i standardni tipovi podataka s brojem bitova koje zauzimaju u podatkovnoj memoriji, minimalnim i maksimalnim vrijednostima te formatom za ispis pomoću funkcije `lcdprintf()`.

Problem koji moramo riješiti u ovoj vježbi jest ispis tri seta podataka svakih 2 sekunde na LCD displej. U programskom kodu 6.5 provedena je inicijalizacija LCD displeja pozivom funkcije `lcdInit()`. Koristit ćemo 6 različitih tipova podataka pa ih je potrebno deklarirati i pridružiti im proizvoljnu vrijednost (inicijalizirati varijable).

Na početku funkcije `main()` deklarirajte i inicijalizirajte sljedeće varijable:

- `int intVar = 125;` - deklaracija i inicijalizacija cjelobrojnog tipa s predznakom veličine 16 bitova,

- `float floatVar = 9.5828;` - deklaracija i inicijalizacija realnog tipa veličine 32 bita.
- `int8_t int8Var = -120;` - deklaracija i inicijalizacija cjelobrojnog tipa s predznakom veličine 8 bitova,
- `uint8_t uint8Var = 250;` - deklaracija i inicijalizacija cjelobrojnog tipa bez predznaka veličine 8 bitova,
- `int32_t int32Var = -250250;` - deklaracija i inicijalizacija cjelobrojnog tipa s predznakom veličine 32 bita,
- `uint16_t uint16Var = 50000;` - deklaracija i inicijalizacija cjelobrojnog tipa bez predznaka veličine 16 bitova.

U `while` beskonačnoj petlji programskog koda 6.5 nalaze se pozivi triju funkcija za kašnjenje `_delay_ms(2000);`. U beskonačnu petlju `while` ispred prve funkcije za kašnjenje `_delay_ms(2000);` napišite sljedeće linije programskog koda:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("int: %d\n", intVar);` - ispis teksta `int:` te cjelobrojne varijable `intVar` na mjestu specifikatora formata ispisa `%d`. Prema tablici 6.1, za cjelobrojni tip s predznakom `int` koristi se specifikatora formata ispisa `%d`. Na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.
- `lcdprintf("float: %.3f", floatVar);` - ispis teksta `float:` te realne varijable `floatVar` s preciznošću 3 decimalna mjesta na mjestu specifikatora formata ispisa `%.3f`. Prema tablici 6.1, za realni tip `float` koristi se specifikatora formata ispisa `%f`. Između znakova `%` i `f` nalaze se točka i broj koji definira preciznost ispisa.

U beskonačnu petlju `while` ispred druge funkcije za kašnjenje `_delay_ms(2000);` napišite sljedeće linije programskog koda:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("int8: %d\n", int8Var);` - ispis teksta `int8:` te cjelobrojne varijable `int8Var` na mjestu specifikatora formata ispisa `%d`. Prema tablici 6.1, za cjelobrojni tip s predznakom `int8_t` koristi se specifikatora formata ispisa `%d`. Na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.
- `lcdprintf("uint8: %u", uint8Var);` - ispis teksta `uint8:` te cjelobrojne varijable `uint8Var` na mjestu specifikatora formata ispisa `%u`. Prema tablici 6.1, za cjelobrojni tip bez predznaka `uint8_t` koristi se specifikatora formata ispisa `%u`.

U beskonačnu petlju `while` ispred treće funkcije za kašnjenje `_delay_ms(2000);` napišite sljedeće linije programskog koda:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("int32: %ld\n", int32Var);` - ispis teksta `int32:` te cjelobrojne varijable

int32Var na mjestu specifikatora formata ispisa %ld. Prema tablici 6.1, za cjelobrojni tip s predznakom int32_t koristi se specifikatora formata ispisa %ld. Na kraju stringa se nalazi specijalni znak '\n' koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.

- lcdprintf("uint16: %u", uint16Var); - ispis teksta uint16: te cjelobrojne varijable uint16Var na mjestu specifikatora formata ispisa %u. Prema tablici 6.1, za cjelobrojni tip bez predznaka uint16_t koristi se specifikatora formata ispisa %u.

Ako ste slijedili gore navedene korake, Vaša datoteka vjezba62.cpp treba biti ista kao programski kod 6.6.

Programski kod 6.6: Sadržaj datoteke vjezba62.cpp - ispis cjelobrojnih i realnih tipova podataka na LCD displej svakih 2 sekunde

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {

    int intVar = 125; // cjelobrojni tip s predznakom, 16 bitova
    float floatVar = 9.5828; // realni tip, 32 bita
    int8_t int8Var = -120; // cjelobrojni tip s predznakom, 8 bitova
    uint8_t uint8Var = 250; // cjelobrojni tip bez predznakom, 8 bitova
    int32_t int32Var = -250250; // cjelobrojni tip s predznakom, 32 bita
    uint16_t uint16Var = 50000; // cjelobrojni tip bez predznakom, 16 bitova

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("int: %d\n", intVar);
        lcdprintf("float: %.3f", floatVar);
        _delay_ms(2000); // kašnjenje 2000 ms

        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("int8: %d\n", int8Var);
        lcdprintf("uint8: %u", uint8Var);
        _delay_ms(2000); // kašnjenje 2000 ms

        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("int32: %ld\n", int32Var);
        lcdprintf("uint16: %u", uint16Var);
        _delay_ms(2000); // kašnjenje 2000 ms

    }

    return 0;
}
```

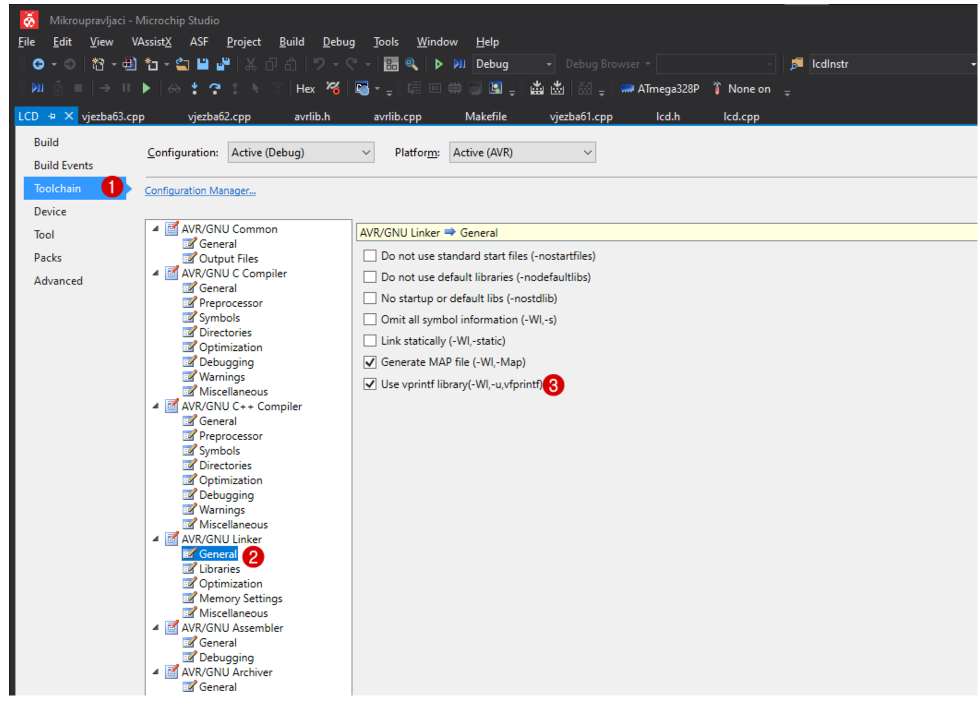
Prevedite datoteku vjezba62.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se tri seta podataka svakih sekunde.

Ako pri ispisu realnog broja (tip float) pomoću funkcije lcdprintf() na LCD displeju

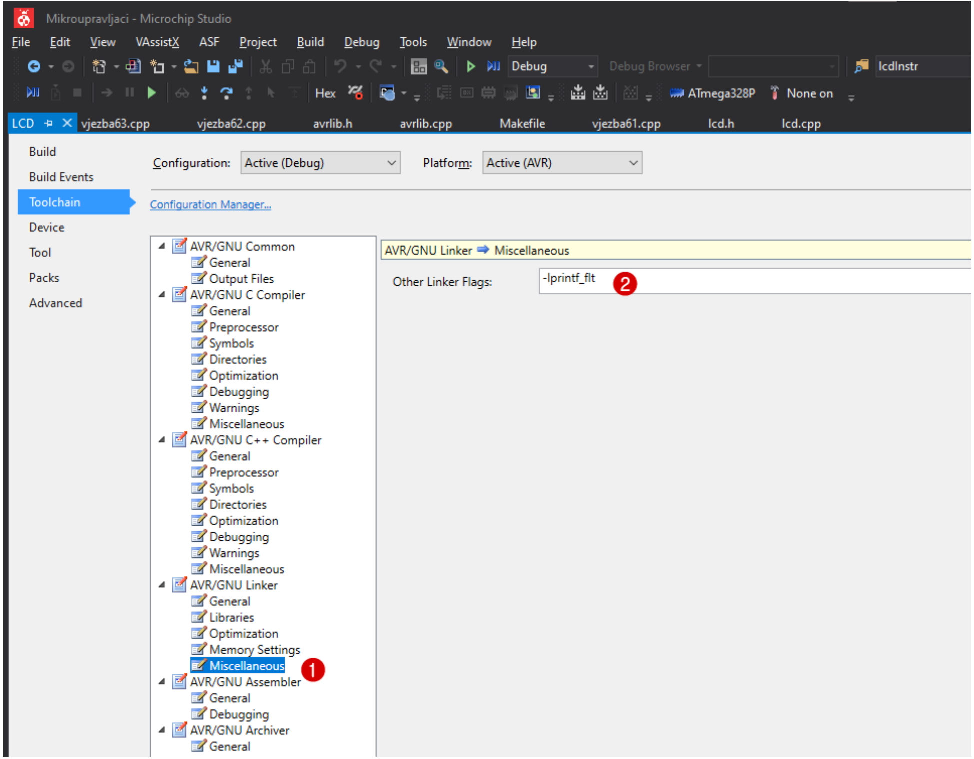
umjesto željenog broja dobijete znak `?`, potrebno je omogućiti prevođenje realnih brojeva u tekst. U programskom okruženju *Microchip Studio*početne su postavke takve da je onemogućen ispis realnih brojeva u funkcijama koje pozivaju funkcije `sprintf` (kao što je `lcdprintf()`) ili `printf()` radi uštede programske memorije mikroupravljača. Ako je nužan ispis realnih brojeva, tada je potrebno napraviti sljedeće korake:

- u programskom okruženju *Microchip Studio*odaberite izbornik *Project* → *LCD Properties...* ili pritisnite *Alt+F7* (slika 6.3),
- u otvorenom prozoru `LCD` odaberite (slika 6.4):
  1. *Toolchain*,
  2. *General* u *AVR/GNU Linker*,
  3. označite *Use vprintf librabry(-Wl,-u,vfprintf)*.
- u *Toolchain* i *AVR/GNU Linker* (slika 6.5):
  1. odaberite *Miscellaneous* ,
  2. u tekstualni okvir *Other Linker Flags* upišite `-lprintf_flt`.
- snimite nove postavke programskog okruženja *Microchip Studio*(Ctrl + S).

Slika 6.3: Omogućavanje ispisa realnih brojeva na AVR mikroupravljačima u programskom okruženju *Microchip Studio*(1/3)



Slika 6.4: Omogućavanje ispisa realnih brojeva na AVR mikroupravljačima u programskom okruženju *Microchip Studio*(2/3)



Slika 6.5: Omogućavanje ispisa realnih brojeva na AVR mikroupravljačima u programskom okruženju *Microchip Studio*(3/3)

Ponovno prevedite datoteku `vjezba62.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Sada će realni broj biti ispisan na 3 decimalna mjesta.

U programskom kodu 6.6 promijenite inicijalizacije sljedećih varijabli:

- `int8_t int8Var = -300;`
- `uint8_t uint8Var = 450;`
- `uint16_t uint16Var = 100000;`

Ponovno prevedite datoteku `vjezba62.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

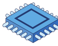
Pri ispisu varijable `int8Var` umjesto očekivanih -300, na LCD displeju ispisan je broj -44. Razlog tome je što broj -300 ne stane u cjelobrojnu varijablu s predznakom veličine 8 bita čija je minimalna vrijednost -128, a maksimalna vrijednost 127. Kod pohrane broja -300 u varijablu `int8Var` 9. bit nije pohranjen u varijablu. Taj bit u 2. komplementu ima težinu -256 koja je izgubljena (-300 + 256 = -44).

Pri ispisu varijable `uint8Var` umjesto očekivanih 450, na LCD displeju ispisan je broj 194. Razlog tome je što broj 450 ne stane u cjelobrojnu varijablu bez predznaka veličine 8 bita čija je minimalna vrijednost 0, a maksimalna vrijednost 255. Kod pohrane broja 450 u varijablu `uint8Var` 9. bit nije pohranjen u varijablu. Taj bit ima težinu 256 koja je izgubljena (450 - 256 = 194).

Pri ispisu varijable `uint16Var` umjesto očekivanih 100000, na LCD displeju ispisan je broj 34464. Razlog tome je što broj 100000 ne stane u cjelobrojnu varijablu bez predznaka veličine 16 bita čija je minimalna vrijednost 0, a maksimalna vrijednost 65535. Kod pohrane broja 100000 u varijablu `uint16Var` 17. bit nije pohranjen u varijablu. Taj bit ima težinu 65536 koja je izgubljena (100000 - 65536 = 34464).

Namjerno krivo inicijalizirane vrijednosti analizirane su kako biste ubuduće vodili računa o rasponu vrijednosti pojedine varijable.

Zatvorite datoteku `vjezba62.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 6.3

Napišite program koji će u prvom retku LCD displeju svaku sekundu ispisivati kut u rasponu od 0 do 180 °, a u drugom retku kosinus tog kuta na 4 decimalna mjesta. ASCII kod za znak ° jest 223[4]. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba63.cpp`. Omogućite prevođenje datoteke `vjezba63.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba63.cpp` prikazan je programskim kodom 6.7.

Programski kod 6.7: Početni sadržaj datoteke `vjezba63.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {
```

[4] ASCII kod za znak ° može ovisiti o LCD displeju i njegovom internom mikroupravljaču. Potrebno je pogledati tehničku specifikaciju LCD displeja ukoliko 223 nije ASCII kod za znak °.

```
    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        _delay_ms(1000); // kašnjenje 1000 ms

    }

    return 0;
}
```

U ovoj vježbi je potrebno koristiti trigonometrijsku funkciju kosinus. Matematičke funkcije kao što su kosinus, sinus, eksponencijala, logaritam i slične nalaze se deklarirane u standardnom zaglavlju `math.h`. U programski kod 6.7 napišite naredbu `#include <math.h>` kojom ćete uključiti zaglavlje `math.h`. Trigonometrijska funkcija kosinus prima argument kuta u radijanima. Iz ovog razloga će kut u stupnjevima biti potrebno pretvoriti u radijane prema realciji:

$$\alpha\,[rad] = \alpha\,[^\circ]\,\frac{\pi}{180} \tag{6.1}$$

Broj π je konstanta koju ćemo definirati tako da u programski kod 6.7 napišemo naredbu `#define PI 3.14159265L` [5].Na početku funkcije `main()` deklarirajte sljedeće varijable:

- `uint8_t kut = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjen kut u rasponu od 0 do 180,

- `float coskuta;` - deklaracija varijable realnog tipa u koju će biti pohranjen kosinus kuta.

U beskonačnu petlju `while` ispred funkcije za kašnjenje `_delay_ms(1000);` napišite sljedeće linije programskog koda:

- `coskuta = cos(kut * PI / 180.0);` - poziv funkcije kosinus uz pretvorbu kuta u radijane prema relaciji 6.1.

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.

- `lcdprintf("Kut: %u %c\n", kut, 223);` - ispis teksta `Kut:` te cjelobrojne varijable `kut` na mjestu specifikatora formata ispisa `%u`. Iza ispisa kuta dolazi znak za ° čiji je ASCII kod 223 i koji se ispisuje na mjestu specifikatora formata ispisa `%c`. Na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.

- `lcdprintf("Cos(%u): %.4f", kut, coskuta);` - ispis teksta `Cos(` te cjelobrojne varijable `kut` na mjestu specifikatora formata ispisa `%u`. Nakon toga se ispisuje tekst `):` te realna varijabla `coskuta` s preciznošću 4 decimalna mjesta na mjestu specifikatora formata ispisa `%.4f`.

- `kut++;` - povećanje varijable `kut` za 1.

- `if(kut > 180){kut = 0;}` - uvjet kojim se kut drži u rasponu od 0 do 180 °.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba63.cpp` treba biti ista kao programski kod 6.8.

[5] L je sufiks koji se koristi da se prevoditelju naglasi da je konstanta tipa `double`

Programski kod 6.8: Sadržaj datoteke `vjezba63.cpp` - ispis kuta i kosinusa kuta na LCD displej svakih sekundu

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include <math.h>
#define PI 3.14159265L // definirana konstanta za broj pi

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {

    uint8_t kut = 0; // početni kut je 0
    float coskuta;

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        coskuta = cos(kut * PI / 180.0);
        lcdClrScr(); // brisanje znakova LCD displeja +  home pozicija kursora
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("Kut: %u%c\n", kut, 223);
        lcdprintf("Cos(%u): %.4f", kut, coskuta);

        kut++; // povecaj stupanj za 1
        // ako je kut veći od 180, vratite ga na 0
        if(kut > 180) {
            kut = 0;
        }
        _delay_ms(1000); // kašnjenje 1000 ms
    }

    return 0;
}
```

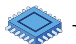
Prevedite datoteku `vjezba63.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se kut i kosinus kuta svakih sekundu.

Ovisno o LCD displeju i njegovom internom mikruopravljaču, mogući su različiti ASCII kodovi za znak °. Najjednostavniji način za pronalaženje znaka ° i bilo kojeg drugog znaka jest proći kroz sve ASCII znakove LCD displeja koristeći naredbu `lcdprintf("ASCII:%u znak: %c", uint8Var, uint8Var++);` u petlji svakih sekundu.

Promijenite raspon kuta od 0 do 360 ° te izračunajte i ispišite sinus kuta na LCD displej. Potrebno je promijeniti tip varijable `kut` iz `uint8_t` u `uint16_t`. Razlog tomu je što maksimalna vrijednost koja se može pohraniti u varijablu tipa `uint8_t` iznosi 255. Umjesto funkcije `cos()` potrebno je pozvati funkciju `sin()`.

Ponovno prevedite datoteku `vjezba63.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba63.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 6.4

Napišite program koji će u prvom retku LCD displeja ispisivati koliko puta je pritisnuto tipkalo T1, a u drugom retku koliko je puta pritisnuto tipkalo T2. Shema povezivanja LCD displeja na

razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Prema shemi na slici 6.1, tipkalo T1 spojeno je na digitalni ulaz PD4, a tipkalo T2 na digitalni ulaz PD2 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba64.cpp`. Omogućite prevođenje datoteke `vjezba64.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba64.cpp` prikazan je programskim kodom 6.9.

Programski kod 6.9: Početni sadržaj datoteke `vjezba64.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača


    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Cilj ove vježbe je prikazati na LCD displeju varijable koje su generirane pritiskom na tipkalo. Potrebno je prebrojiti koliko su puta tipkala T1 i T2 bila pritisnuto i te brojeve ispisati na LCD displej. Za potrebe brojanja koliko je puta pritisnuto tipkalo deklarirat ćemo dvije varijable. Na početku funkcije `main()` deklarirajte i inicijalizirajte sljedeće varijable:

- `uint16_t brojacTipkalo1 = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjeno koliko je puta pritisnuto tipkalo T1. Početno je varijabla inicijalizirana na vrijednost 0[6].

- `uint16_t brojacTipkalo2 = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjeno koliko je puta pritisnuto tipkalo T2. Početno je varijabla inicijalizirana na vrijednost 0.

Tipkalo je pritisnuto ako se pojavio padajući brid signala tipkala. Programski kod kojim se detektira padajući brid signala tipkala T1 i T2 možete kopirati iz vježbe 5.6. Unutar `if` uvjetovanog bloka za detektiranje padajućeg brida signala potrebno je:

[6] Varijable u kojima će se pribrajati vrijednosti uvijek je korisno inicijalizirati na početnu vrijednost. Ponekad je upitno što će prevoditelj napraviti nakon deklaracije varijable, no prema C standardu prevoditelj samo zauzima memorijski prostor i ne radi inicijalizaciju varijable.

- za tipkalo T1 dodati naredbu `brojacTipkalo1++;` - naredba kojom će se varijabla povećati za jedan na svaki padajući brid, odnosno na svaki pritisak na tipkalo T1,
- za tipkalo T2 dodati naredbu `brojacTipkalo2++;` - naredba kojom će se varijabla povećati za jedan na svaki padajući brid, odnosno na svaki pritisak na tipkalo T2.

U beskonačnu petlju `while` napišite sljedeće linije programskog koda:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("T1: %u\n", brojacTipkalo1);` - ispis teksta `T1:` te cjelobrojne varijable `brojacTipkalo1` na mjestu specifikatora formata ispisa `%u`. Na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.
- `lcdprintf("T2: %u", brojacTipkalo2);` - ispis teksta `T2:` te cjelobrojne varijable `brojacTipkalo2` na mjestu specifikatora formata ispisa `%u`.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba64.cpp` treba biti ista kao programski kod 6.10.

Programski kod 6.10: Sadržaj datoteke `vjezba64.cpp` - program koji broji i ispisuje koliko su puta pritisnuta tipkala T1 i T2 (prvi način)

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    // varijable za brojanje broj pritisaka tipkala
    uint16_t brojacTipkalo1 = 0;
    uint16_t brojacTipkalo2 = 0;

    while (1) { // beskonačna petlja
        // ako se pojavio padajući brid na tipkalu T1
        if (isFallingEdge(TIPKALO1)) {
            brojacTipkalo1++; // povećaj brojac za 1
        }
        // ako se pojavio padajući brid na tipkalu T2
        if (isFallingEdge(TIPKALO2)) {
            brojacTipkalo2++; // povećaj brojac za 1
        }
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
```

```
        lcdprintf("T1: %u\n", brojacTipkalo1);
        lcdprintf("T2: %u", brojacTipkalo2);

    }

    return 0;
}
```

Prevedite datoteku `vjezba64.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će koliko puta su pritisnuta tipkala T1 i T2. Možemo li biti zadovoljni s načinom na koji se trenutno tekst prikazuje na LCD displeju? Primijetit ćete da tekst nije stacionaran, već cijelo vrijeme titra slabijim intenzitetom prikaza nego inače. Razlog ovom problemu leži u tome što se u beskonačnoj `while` petlji neprestano poziva funkcija `lcdClrScr()` koja briše tekst i pozove se stotinjak puta u jednoj sekundi. Takav način primjene funkcije `lcdClrScr()` stvara vidno negativan efekt. Ovo je najčešća pogreška polaznika edukacija s kojom smo se susreli u dosadašnjoj praksi. Da bi se smanjio efekt titranja, prvo što Vam može "pasti na pamet" jest da u beskonačnu `while` petlju dodate kašnjenje od 1000 ms (može i neki drugi iznos). U beskonačnu petlju `while` dodajte poziv funkcije `_delay_ms(1000);`.

Ponovno prevedite datoteku `vjezba64.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Primijetit ćete da je tekst na LCD zaslonu sada stacionaran, no pokušajte pritiskati tipkala. Činjenica jest da se zbog prevelikog kašnjenja u beskonačnoj `while` petlji neće uvijek uspjeti detektirati padajući brid (osim ako držite stisnuto tipkalo barem jednu sekundu). Oba načina na koji smo pokušali riješiti definirani problem nisu dali dobre rezultate. Važno je da smo pokazali ova dva načina kako u praksi ne biste napravili iste pogreške.

Dobar pristup pri ispisivanju vrijednosti procesnih varijabli na LCD displeju jest promijeniti prikaz samo ako se dogodila promjena varijable. Da bismo mogli pratiti da li se neka varijabla promijenila unutar jednog ciklusa `while` petlje, potrebno je deklarirati varijable koje će pohranjivati vrijednosti pritisaka na tipkala iz prethodnog ciklusa. Iza deklaracije varijabli `brojacTipkalo1` i `brojacTipkalo2` deklarirajte i inicijalizirajte sljedeće varijable:

- `uint16_t brojacTipkalo1Old = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjeno koliko je puta pritisnuto tipkalo T1 u prethodnom ciklusu `while` petlje (tzv. stara vrijednost varijable `brojacTipkalo1`). Početno je varijabla inicijalizirana na vrijednost 0.

- `uint16_t brojacTipkalo2Old = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjeno koliko je puta pritisnuto tipkalo T2 u prethodnom ciklusu `while` petlje (tzv. stara vrijednost varijable `brojacTipkalo2`). Početno je varijabla inicijalizirana na vrijednost 0.

Dodatno napravite deklaraciju sljedeće varijable:

- `bool ispisNaLcd = true;` - deklaracija i inicijalizacija varijable *Boolean* tipa u kojoj će biti pohranjena vrijednost `true` ako je potrebno napraviti ispis na LCD displej, a `false` ako nije potrebno napraviti ispis na LCD displej. Početno je varijabla inicijalizirana na vrijednost `true` kako bi se na početku ispisao sadržaj na LCD displej.

Varijabla `ispisNaLcd` postavlja se na vrijednost `true` kada su parovi varijabli `brojacTipkalo1` i `brojacTipkalo1Old` te `brojacTipkalo2` i `brojacTipkalo2Old` različiti. Kada su navedeni parovi varijabli različiti, to znači da je došlo do promjene varijable `brojacTipkalo1`, odnosno `brojacTipkalo2`.

U beskonačnu petlju `while` napišite, ispod funkcija koje detektiraju padajuće bridove signala

tipkala T1 i T2, sljedeće uvjetovane blokove:

- `if (brojacTipkalo1 != brojacTipkalo1Old)` - provjera da li je nova vrijednost broja pritisaka tipkala T1 različita od stare. Ako jest, došlo je do promjene.
- `if (brojacTipkalo2 != brojacTipkalo2Old)` - provjera da li je nova vrijednost broja pritisaka tipkala T2 različita od stare. Ako jest, došlo je do promjene.

Unutar oba uvjetovana bloka potrebno je upisati naredbu `ispisNaLcd = true;` kako bi se zbog promjene vrijednosti varijabli `brojacTipkalo1` i `brojacTipkalo2` osvježio ispis na LCD displeju. Brisanje sadržaja na LCD displeju i prikaz novoga sada je potrebno uvjetovati pomoću varijable `ispisNaLcd`, odnosno uvjetovanog bloka `if (ispisNaLcd)`. Nakon ispisa novog sadržaja na LCD, unutar navedenog uvjetovanog bloka potrebno je upisati naredbu `ispisNaLcd = false;` kako bi se onemogućio ispis u sljedećem ciklusu `while` petlje.

Na kraju ciklusa `while` petlje potrebno je napisati sljedeća pridruživanja:

- `brojacTipkalo1Old = brojacTipkalo1` - priprema prethodne vrijednosti varijable `brojacTipkalo1` za sljedeći ciklus `while` petlje.
- `brojacTipkalo2Old = brojacTipkalo2` - priprema prethodne vrijednosti varijable `brojacTipkalo2` za sljedeći ciklus `while` petlje.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba64.cpp` treba biti ista kao programski kod 6.11.

Programski kod 6.11: Sadržaj datoteke `vjezba64.cpp` - program koji broji i ispisuje koliko su puta pritisnuta tipkala T1 i T2 (drugi način)

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    // varijable za brojanje broj pritisaka tipkala
    uint16_t brojacTipkalo1 = 0;
    uint16_t brojacTipkalo2 = 0;
    // varijable za stare vrijednosti broja pritisaka tipkala
    uint16_t brojacTipkalo1Old = 0;
    uint16_t brojacTipkalo2Old = 0;
    // boolean varijabla kojom se omogućuje ispis na LCD
    bool ispisNaLcd = true;

    while (1) { // beskonačna petlja
        // ako se pojavio padajući brid na tipkalu T1
        if (isFallingEdge(TIPKALO1)) {
```

```
            brojacTipkalo1++; // povećaj brojac za 1
        }
        // ako se pojavio padajući brid na tipkalu T2
        if (isFallingEdge(TIPKALO2)) {
            brojacTipkalo2++; // povećaj brojac za 1
        }
        // ako je bilo promjene brojača za tipkalo T1
        if (brojacTipkalo1 != brojacTipkalo1Old) {
            ispisNaLcd = true; // omogući ispis na LCD
        }
        // ako je bilo promjene brojača za tipkalo T2
        if (brojacTipkalo2 != brojacTipkalo2Old) {
            ispisNaLcd = true;  // omogući ispis na LCD
        }
        // ako je omogućen ispis
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("T1: %u\n", brojacTipkalo1);
            lcdprintf("T2: %u", brojacTipkalo2);
            ispisNaLcd = false; // onemogući ispis na LCD
        }
        // na kraju while petlje pripremmiti stare vrijednosti za novi ciklus
        brojacTipkalo1Old = brojacTipkalo1;
        brojacTipkalo2Old = brojacTipkalo2;
    }
    return 0;
}
```

Ponovno prevedite datoteku `vjezba64.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Sada će se sadržaj teksta mijenjati samo kada se pritisne tipkalo. Sada možemo biti zadovoljni s prikazom teksta koji je stacionaran, a s druge strane uspijevamo obraditi svaki pritisak na tipkalo.

Ipak, i u programskom kodu 6.11 su moguća poboljšanja. Pažljivi programer mikroupravljača primijetit će da u prikazu imamo fiksni dio teksta i varijabilni dio teksta. Prema programskom kodu 6.11 mi uvijek brišemo sav sadržaj s LCD displeja pri promjeni vrijednosti varijabli `brojacTipkalo1` i `brojacTipkalo2`. Kada bi te promjene bile brze, tada bi se okom moglo vidjeti da se fiksni dio teksta neprestano osvježava.

Fiksni dio teksta je `T1:` u prvom retku LCD displeja i `T2:` u drugom retku LCD displeja. Taj dio možemo ispisati samo jednom na LCD displej prije ulaska u beskonačnu `while` petlju. Upišite ispod deklaracije varijable `ispisNaLcd` sljedeće linije programskog koda;

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("T1: n");` - ispis fiksnog dijela teksta `T1:`. Na kraju stringa se nalazi specijalni znak `'\n'` koji će kursor postaviti na početak sljedećeg retka (2. redak, 1. stupac) za sljedeći ispis.
- `lcdprintf("T2: ");` - ispis fiksnog dijela teksta `T1:`.

U beskonačnoj `while` petlji sada je potrebno osvježavati samo vrijednosti varijabli `brojacTipkalo1` i `brojacTipkalo2`. Širina fiksnih dijela tekstova `T1:` i `T2:` je 4 stupca. Ispis varijabilnog dijela teksta potrebno je pozicionirati od 5. stupca. Postavlja se pitanje, kako izbrisati prethodni sadržaj varijabilnog teksta. Najjednostavniji način je da ga prepišemo s praznim mjestima (engl. *white space*). Tip varijabla `brojacTipkalo1` i `brojacTipkalo2` jest

uint16_t što znači da ove varijable mogu imati najviše 5 znamenaka (maksimalna vrijednost tipa uint16_t jest 65535). Princip brisanja varijabilnog sadržaja opisat ćemo na dva načina kroz sljedeće linije programskog koda. Unutar uvjetovanog bloka if (ispisNaLcd) zamijenite postojeće linije programskog koda sa sljedećima:

- lcdGotoXY(1, 5); - pozicioniranje kursora za ispis teksta u 1. redak i 5. stupac.
- lcdprintf("%u   ", brojacTipkalo1); - ispis cjelobrojne varijable brojacTipkalo1 na mjestu specifikatora formata ispisa %u. Dodatno je ispisano 4 prazna mjesta jer u najgorem slučaju je vrijednost varijable brojacTipkalo1 jednoznamenkasta pa je potencijalno potrebno izbrisati još 4 znamenke.
- lcdGotoXY(2, 5); - pozicioniranje kursora za ispis teksta u 2. redak i 5. stupac.
- lcdprintf("   "); - ispis 5 praznih mjesta jer je maksimalna širina tipa uint16_t 5 znamenaka.
- lcdGotoXY(2, 5); - vraćanje kursora za ispis teksta u 2. redak i 5. stupac. Ovo je važno napraviti jer je interni kursor bio na stupcu s indeksom 10 radi ispisa 5 praznih mjesta.
- lcdprintf("%u", brojacTipkalo2); - ispis cjelobrojne varijable brojacTipkalo1 na mjestu specifikatora formata ispisa %u.
- ispisNaLcd = false; - onemogućenje ispisa teksta na LCD do nove promjene varijabli brojacTipkalo1 i brojacTipkalo2.

Ako ste slijedili gore navedene korake, Vaša datoteka vjezba64.cpp treba biti ista kao programski kod 6.12.

Programski kod 6.12: Sadržaj datoteke vjezba64.cpp - program koji broji i ispisuje koliko su puta pritisnuta tipkala T1 i T2 (treći način)

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    // varijable za brojanje broj pritisaka tipkala
    uint16_t brojacTipkalo1 = 0;
    uint16_t brojacTipkalo2 = 0;
    // varijable za stare vrijednosti broja pritisaka tipkala
    uint16_t brojacTipkalo1Old = 0;
    uint16_t brojacTipkalo2Old = 0;
    // boolean varijabla kojom se omogućuje ispis na LCD
    bool ispisNaLcd = true;
```

```
    // ispis fiksnog dijela teksta na LCD
    lcdClrScr();
    lcdprintf("T1: \n");
    lcdprintf("T2: ");

    while (1) { // beskonačna petlja
        // ako se pojavio padajući brid na tipkalu T1
        if (isFallingEdge(TIPKALO1)) {
            brojacTipkalo1++; // povećaj brojac za 1
        }
        // ako se pojavio padajući brid na tipkalu T2
        if (isFallingEdge(TIPKALO2)) {
            brojacTipkalo2++; // povećaj brojac za 1
        }
        // ako je bilo promjene brojača za tipkalo T1
        if (brojacTipkalo1 != brojacTipkalo1Old) {
            ispisNaLcd = true; // omogući ispis na LCD
        }
        // ako je bilo promjene brojača za tipkalo T2
        if (brojacTipkalo2 != brojacTipkalo2Old) {
            ispisNaLcd = true;  //omogući ispis na LCD
        }
        // ako je omogućen ispis
        if (ispisNaLcd) {
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdGotoXY(1, 5); // pozicioniranje kursora u 1. red, 5. stupac
            lcdprintf("%u    \n", brojacTipkalo1); // ispiši broj + 4 space
            lcdGotoXY(2, 5); // pozicioniranje kursora u 2. red, 5. stupac
            lcdprintf("     "); // izbriši 5 mjesta (prepiši sa Space)
            lcdGotoXY(2, 5); // pozicioniranje kursora u 2. red, 5. stupac
            lcdprintf("%u", brojacTipkalo2); // ispiši broj
            ispisNaLcd = false; // onemogući ispis na LCD
        }
        // na kraju while petlje pripremmiti stare vrijednosti za novi ciklus
        brojacTipkalo1Old = brojacTipkalo1;
        brojacTipkalo2Old = brojacTipkalo2;
    }
    return 0;
}
```

Ponovno prevedite datoteku `vjezba64.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zadnji pristup ispisa na LCD displej je najsloženiji, ali je istovremeno i najstabilniji s obzirom na brzinu promjene varijabli koje se ispisuju. Ponekad će pristup prikazan programskim kodom 6.11 biti sasvim dostatan.

Zatvorite datoteku `vjezba64.cpp` i onemogućite prevođenje ove datoteke.

Vježba 6.5

Napišite program koji će na LCD displeja prikazivati 4 izbornika sa sljedećim sadržajem:

- Izbornik 1: - u prvom retku LCD displeja ispisati tekst `Frekvencija`, a u drugom retku tekst `f =` , broj koji predstavlja frekvenciju te tekst `Hz`
- Izbornik 2: - u prvom retku LCD displeja ispisati tekst `Napon`, a u drugom retku tekst `u =` , broj koji predstavlja napon te tekst `V`
- Izbornik 3: - u prvom retku LCD displeja ispisati tekst `Struja 1`, a u drugom retku tekst `i1 =` , broj koji predstavlja prvu struju te tekst `mA`

- Izbornik 4: - u prvom retku LCD displeja ispisati tekst `Struja 2`, a u drugom retku tekst `i2 =` , broj koji predstavlja prvu struju te tekst `mA`

Omogućite izmjenu parametara i izbornika na sljedeći način:

- pritiskom na tipkalo T1 smanjuje se parametar u aktualnom izborniku za definiran korak,
- pritiskom na tipkalo T2 povećava se parametar u aktualnom izborniku za definiran korak,
- pritiskom na tipkalo T3 cirkularno se mijenjaju izbornici: `Frekvencija` → `Napon` → `Struja 1` → `Struja 2` → `Frekvencija` ...

Za parametre vrijede sljedeća ograničenja:

- `Frekvencija` - minimalna vrijednost je 0, maksimalna vrijednost je 50, a pritiskom na tipkalo vrijednost se mijenja za 2,
- `Napon` - minimalna vrijednost je -12, maksimalna vrijednost je 12, a pritiskom na tipkalo vrijednost se mijenja za 1,
- `Struja 1` - minimalna vrijednost je -1000, maksimalna vrijednost je 1000, a pritiskom na tipkalo vrijednost se mijenja za 50,
- `Struja 2` - minimalna vrijednost je -20000, maksimalna vrijednost je 20000, a pritiskom na tipkalo vrijednost se mijenja za 500.

Vrijednost parametara ne smije se izgubiti prelaskom iz izbornika u izbornik, a početno svi parametri imaju vrijednost 0. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Prema shemi na slici 6.1, tipkalo T1 spojeno je na digitalni ulaz PD4, tipkalo T2 spojeno je na digitalni ulaz PD2, a tipkalo T3 na digitalni ulaz PD5 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba65.cpp`. Omogućite prevođenje datoteke `vjezba65.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba65.cpp` prikazan je programskim kodom 6.13.

Programski kod 6.13: Početni sadržaj datoteke `vjezba65.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2
#define TIPKALO3 D5

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD2
    pinMode(TIPKALO3, INPUT_PULLUP); // PD5 konfiguriran kao ulaz
    // uključen pull up otpornik na pinu PD5
}

int main(void) {
```

```
    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Cilj ove vježbe je promjena parametara sustava pomoću izbornika koji se prikazuju na LCD displeju. Izbornici su korisni kada je potrebno promijeniti parametre sustava ili kada je potrebno pogledati procesne varijable kao što su frekvencija, napon, struja, vrijeme rada, broj proizvedenih komada i slično. Programski kod ove vježbe možemo razdvojiti na dva segmenta: dio koji se bavi obradom tipkala i dio koji se bavi ispisom na LCD displej.

Na početku funkcije `main()` deklarirajte i inicijalizirajte sljedeće varijable:

- `uint8_t izbornik = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjen aktualni izbornik. Vrijednosti varijable su:
  - 0 za izbornik `Frekvencija`
  - 1 za izbornik `Napon`
  - 2 za izbornik `Struja 1`
  - 3 za izbornik `Struja 2`
- `int16_t p[4] = {};` - deklaracija i inicijalizacija polja sa četiri elementa koji su cjelobrojnog tipa u koje će se spremati parametri frekvencija (`p[0]`), napon (`p[1]`), struja 1 (`p[2]`) i struja 2 (`p[3]`). Elemente polja adresirat ćemo s varijablom `izbornik`. Sve početne vrijednosti parametra su 0.
- `const int16_t pMin[4] = {0, -12, -1000, -20000};` - deklaracija i inicijalizacija polja sa četiri elementa koji su konstante cjelobrojnog tipa u koje će se spremati minimalne vrijednosti parametara frekvencije (`pMin[0]`), napona (`pMin[1]`), struje 1 (`pMin[2]`) i struje 2 (`pMin[3]`). Elemente polja adresirat ćemo s varijablom `izbornik`.
- `const int16_t pMax[4] = {50, 12, 1000, 20000};` - deklaracija i inicijalizacija polja sa četiri elementa koji su konstante cjelobrojnog tipa u koje će se spremati maksimalne vrijednosti parametara frekvencije (`pMax[0]`), napona (`pMax[1]`), struje 1 (`pMax[2]`) i struje 2 (`pMax[3]`). Elemente polja adresirat ćemo s varijablom `izbornik`.
- `const int16_t pStep[4] = {2, 1, 50, 500};` - deklaracija i inicijalizacija polja sa četiri elementa koji su konstante cjelobrojnog tipa u koje će se spremati korak promjene vrijednosti parametara frekvencije (`pStep[0]`), napona (`pStep[1]`), struje 1 (`pStep[2]`) i struje 2 (`pStep[3]`) kada se pritisne tipkalo T1 ili T2. Elemente polja adresirat ćemo s varijablom `izbornik`.
- `bool ispisNaLcd = true;` - deklaracija i inicijalizacija *Boolean* varijable kojom se omogućuje ispis na LCD displej. Početno je varijabla postavljana u vrijednost `true` kako bi se na početku rada mikroupravljača ispisao početni sadržaj na LCD displej.

Zbog opsežnosti programskog koda u nastavku, prikazat ćemo najprije dio programskog koda koji se bavi obradom tipkala. U beskonačnu petlju `while` potrebno je napisati programski kod

6.14. Programski kod 6.14 se sastoji od tri uvjetovana `if` bloka naredaba koji detektiraju padajući brid signala na tipkalima T1, T2 i T3.

Prvi uvjetovan `if` blok naredaba detektira padajući brid signala tipkala T1. Ovo tipkalo smanjuje parametar `p[izbornik]` za korak koji je definiran u polju `pStep[]`. Na svaki padajući brid parametar `p[izbornik]` smanjuje se za korak `pStep[izbornik]`. Dodatno, ograničavamo parametar `p[izbornik]` da se ne smanji ispod vrijednosti `pMin[izbornik]`. Ako se ograničenje nije aktiviralo (`if (p[izbornik] < pMin[izbornik])`), onda je došlo do promjene parametra `p[izbornik]`, a varijabla `ispisNaLcd` se postavlja na vrijednost `true` kako bi se promjena parametra odmah vidjela na LCD displeju.

Drugi uvjetovan `if` blok naredaba detektira padajući brid signala tipkala T2. Ovo tipkalo povećava parametar `p[izbornik]` za korak koji je definiran u polju `pStep[]`. Na svaki padajući brid parametar `p[izbornik]` povećava se za korak `pStep[izbornik]`. Dodatno, ograničavamo parametar `p[izbornik]` da se ne poveća iznad vrijednosti `pMax[izbornik]`. Ako se ograničenje nije aktiviralo (`if (p[izbornik] > pMax[izbornik])`), onda je došlo do promjene parametra `p[izbornik]`, a varijabla `ispisNaLcd` se postavlja na vrijednost `true` kako bi se promjena parametra odmah vidjela na LCD displeju.

Treći uvjetovan `if` blok naredaba detektira padajući brid signala tipkala T3. Ovo tipkalo rotira parametar `izbornik` kroz vrijednosti $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 0 \rightarrow 1 \rightarrow ...$. Na svaki padajući brid varijabla `izbornik` povećava se za vrijednost 1. Kada varijabla `izbornik` poprimi vrijednost 4, vraćamo je nazad na početak, odnosno na vrijednost 0. Varijabla `ispisNaLcd` se na svaki padajući brid signala tipkala T3 postavlja na vrijednost `true` kako bi se promjena izbornika odmah vidjela na LCD displeju.

Programski kod 6.14: Obrada tipkala za promjenu parametara i promjenu izbornika

```
// ako se pojavio padajući brid na tipkalu T1
if (isFallingEdge(TIPKALO1)) {
    p[izbornik] -= pStep[izbornik]; // smanji za korak
    // ograniči parametar odozdo
    if (p[izbornik] < pMin[izbornik]) {
        p[izbornik] = pMin[izbornik];
    } else {
        ispisNaLcd = true; // omogući ispis na LCD
    }
}
// ako se pojavio padajući brid na tipkalu T2
if (isFallingEdge(TIPKALO2)) {
    p[izbornik] += pStep[izbornik]; // povećaj za korak
    // ograniči parametar odozgo
    if (p[izbornik] > pMax[izbornik]) {
        p[izbornik] = pMax[izbornik];
        } else {
        ispisNaLcd = true; // omogući ispis na LCD
    }
}
// ako se pojavio padajući brid na tipkalu T3
if (isFallingEdge(TIPKALO3)) {
    izbornik++; // povećaj brojac za 1
    // cirkularna promjena izbornika
    if (izbornik >= 4) {
        izbornik = 0;
    }
    ispisNaLcd = true; // omogući ispis na LCD
}
```

Primijetite da u ovom zadatku nismo pamtili stare vrijednosti parametara i izbornika, već smo iskoristili činjenicu da se pritisnulo tipkalo i na taj način omogućili osvježenje prikaza na LCD displeju. U prošloj vježbi pokazali smo kako se osvježava tekst na primjeru promjene

varijable koja se inače može desiti zbog tipkala, potenciometra, rotacijskog enkodera, serijske komunikacije i drugo.

U nastavku ćemo opisati programski kod koji se bavi ispisom izbornika i parametra pomoću uvjetovanog grananja `switch case` (automata stanja). Ispod programskog koda 6.14, u datoteku `vjezba65.cpp` potrebno je napisati programski kod 6.15. Grananje `switch case` u programskom kodu 6.15 uvjetovano je varijablom `ispisNaLcd` što osigurava da se promjena sadržaja LCD displeja dešava samo kada dolazi do promjene parametara ili izbornika. Prije novog ispisa na LCD displej, pozvana je funkcija `lcdClrScr()` kojom se briše stari sadržaj na LCD displeju. Unutar grananja `switch case` imamo 4 slučaja koji se adresiraju cjelobrojnom varijablom `izbornik`:

- `case 0:` - ispisuje tekst `Frekvencija` u prvom retku LCD displej. U drugom retku LCD displeja ispisuje se tekst `f =`, cjelobrojna varijabla (parametar `p[0]`) na mjestu specifikatora formata ispisa `%d` i tekst `Hz`.
- `case 1:` - ispisuje tekst `Napon` u prvom retku LCD displej. U drugom retku LCD displeja ispisuje se tekst `u =`, cjelobrojna varijabla (parametar `p[1]`) na mjestu specifikatora formata ispisa `%d` i tekst `V`.
- `case 2:` - ispisuje tekst `Struja 1` u prvom retku LCD displej. U drugom retku LCD displeja ispisuje se tekst `i1 =`, cjelobrojna varijabla (parametar `p[2]`) na mjestu specifikatora formata ispisa `%d` i tekst `mA`.
- `case 3:` - ispisuje tekst `Struja 2` u prvom retku LCD displej. U drugom retku LCD displeja ispisuje se tekst `i2 =`, cjelobrojna varijabla (parametar `p[3]`) na mjestu specifikatora formata ispisa `%d` i tekst `mA`.

Naravno, grananje `switch case` ima i slučaj `default:` u slučaju da varijabla `izbornik` nekim čudom poprimi vrijednost koja je drugačija od 0, 1, 2 i 3. Na kraju uvjetovanog bloka `if (ispisNaLcd)`, varijabla `ispisNaLcd` se postavlja na vrijednost `false` kako bi se onemogućio ispis na LCD displej do nove promjene parametara ili izbornika.

Programski kod 6.15: Ispis izbornika i parametra pomoću uvjetovanog grananja `switch case` (automat stanja)

```
// ako je omogućen ispis
if (ispisNaLcd) {
    // brisanje znakova LCD displeja +  home pozicija kursora
    lcdClrScr();
    // switch case za prikaz izbornika
    switch (izbornik) {
        case 0:
            lcdprintf("Frekvencija\n");
            lcdprintf("f = %d Hz", p[izbornik]);
            break;
        case 1:
            lcdprintf("Napon\n");
            lcdprintf("u = %d V", p[izbornik]);
            break;
        case 2:
            lcdprintf("Struja 1\n");
            lcdprintf("i1 = %d mA", p[izbornik]);
            break;
        case 3:
            lcdprintf("Struja 2\n");
            lcdprintf("i2 = %d mA", p[izbornik]);
            break;
        default:
            break;
```

```
    }
    ispisNaLcd = false; // onemogući ispis na LCD
}
```

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba65.cpp` treba biti ista kao programski kod 6.16.

Programski kod 6.16: Sadržaj datoteke `vjezba65.cpp` - program koji pomoću tipkala mijenja izbornike i parametre unutar svakog izbornika

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

#define TIPKALO1 D4
#define TIPKALO2 D2
#define TIPKALO3 D5

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // tipkalo T1
    pinMode(TIPKALO1, INPUT_PULLUP); // PD4 konfiguriran kao ulaz
    // ukljucen pull up otpornik na pinu PD4
    // tipkalo T2
    pinMode(TIPKALO2, INPUT_PULLUP); // PD2 konfiguriran kao ulaz
    // ukljucen pull up otpornik na pinu PD2
    pinMode(TIPKALO3, INPUT_PULLUP); // PD5 konfiguriran kao ulaz
    // ukljucen pull up otpornik na pinu PD5
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint8_t izbornik = 0;
    // parametri unutar izbornika
    // p[0], pMin[0], pMax[0], pStep[0] -> frekvencija
    // p[1], pMin[1], pMax[1], pStep[1] -> napon
    // p[2], pMin[2], pMax[2], pStep[2] -> struja 1
    // p[3], pMin[3], pMax[3], pStep[3] -> struja 2
    int16_t p[4] = {}; // inicijalno su svi parametri 0
    // minimalne vrijednosti parametara
    const int16_t pMin[4] = {0, -12, -1000, -20000};
    // maksimalne vrijednosti parametara
    const int16_t pMax[4] = {50, 12, 1000, 20000};
    // korak promjene parametara
    const int16_t pStep[4] = {2, 1, 50, 500};
    // varijabla kojom se omogućuje ispis na LCD
    bool ispisNaLcd = true;

    while (1) { // beskonačna petlja
        // ako se pojavio padajući brid na tipkalu T1
        if (isFallingEdge(TIPKALO1)) {
            p[izbornik] -= pStep[izbornik]; // smanji za korak
            // ograniči parametar odozdo
            if (p[izbornik] < pMin[izbornik]) {
                p[izbornik] = pMin[izbornik];
            } else {
                ispisNaLcd = true; // omogući ispis na LCD
            }
        }
        // ako se pojavio padajući?i brid na tipkalu T2
        if (isFallingEdge(TIPKALO2)) {
            p[izbornik] += pStep[izbornik]; // povećaj za korak
            // ograniči parametar odozgo
```

```
            if (p[izbornik] > pMax[izbornik]) {
                p[izbornik] = pMax[izbornik];
                } else {
                ispisNaLcd = true; // omogući ispis na LCD
            }
        }
        // ako se pojavio padajući brid na tipkalu T3
        if (isFallingEdge(TIPKALO3)) {
            izbornik++; // povečaj brojac za 1
            // cirkularna promjena izbornika
            if (izbornik >= 4) {
                izbornik = 0;
            }
            ispisNaLcd = true; // omogući ispis na LCD
        }
        // ako je omogućen ispis
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // switch case za prikaz izbornika
            switch (izbornik) {
                case 0:
                    lcdprintf("Frekvencija\n");
                    lcdprintf("f = %d Hz", p[izbornik]);
                    break;
                case 1:
                    lcdprintf("Napon\n");
                    lcdprintf("u = %d V", p[izbornik]);
                    break;
                case 2:
                    lcdprintf("Struja 1\n");
                    lcdprintf("i1 = %d mA", p[izbornik]);
                    break;
                case 3:
                    lcdprintf("Struja 2\n");
                    lcdprintf("i2 = %d mA", p[izbornik]);
                    break;
                default:
                    break;
            }
            ispisNaLcd = false; // onemogući ispis na LCD
        }
    }
    return 0;
}
```

Prevedite datoteku `vjezba65.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju izmjenjivat će se izbornici `Frekvencija`, `Napon`, `Struja 1` i `Struja 2` ovisno o tome koliko puta smo pritisnuli tipkalo T3. Unutar svakog izbornika tipkalom T1 smanjivat će se parametar, a tipkalom T2 povećavat će se parametar. Parametre pojedinog izbornika pohranjivali smo u polje `p[]` upravo kako bismo pri promjeni izbornika sačuvali vrijednosti parametara bez obzira na promjenu izbornika.

Grananje `switch case` je jedan od načina kako možemo dinamički ispisivati tekst izbornika (ili neku drugu vrstu teksta). S obzirom da smo parametre `p[]` adresirali pomoću varijable `izbornik`, postavlja se logično pitanje, može li se tako adresirati i imena izbornika, parametara i mjernih jedinica? Odgovor je, može se! Promjenom izbornika mijenja se naziv izbornika, ime parametra i mjerna jedinica. Te nazive potrebno je pohraniti u polje stringova (dvodimenzionalno polje znakova). Važno je da druga dimenzija polja odgovara najvećem broju znakova koji se pojavljuje u pojedinom imenu izbornika plus 1 za zaključni znak `'\0'`. Polje stringova za

dinamički ispis imena izbornika, parametara i mjernih jedinica na LCD displej prikazano je programskim kodom 6.17. Ovaj programski kod potrebno je dodati iza deklaracije varijable `ispisNaLcd`.

Programski kod 6.17: Polje stringova za dinamički ispis imena izbornika, parametara i mjernih jedinica na LCD displej

```
// imena izbornika koji se ispisuju na LCD
const char imeIzbornika[4][12] = {
    "Frekvencija",
    "Napon",
    "Struja 1",
    "Struja 2",
};
// imena parametara koji se ispisuju na LCD
const char imeParametra[4][3] = {
    "f",
    "u",
    "i1",
    "i2",
};
// imena mjernih jedinica koji se ispisuju na LCD
const char imeMjerneJedinice[4][3] = {
    "Hz",
    "V",
    "mA",
    "mA",
};
```

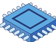
Ispis na LCD displej sada se može provesti bez grananja `switch case`. Dinamički ispis imena izbornika, parametara i mjernih jedinica na LCD displej prikazan je programskim kodom 6.18. Funkcija `lcdprintf()` sada ispisuje imena izbornika, varijabli i mjernih jedinica pomoću specifikatora ispisa `%s`. Na mjestu specifikatora ispisa `%s` ispisuje se tekst koji se nalazi u poljima `imeIzbornika[]`, `imeParametra[]` i `imeMjerneJedinice[]`. Programski kod 6.18 potrebno je prepisati umjesto bloka naredaba koji se nalazi u uvjetovanom grananju `if (ispisNaLcd)`.

Programski kod 6.18: Dinamički ispis imena izbornika, parametara i mjernih jedinica na LCD displej

```
if (ispisNaLcd) {
    // brisanje znakova LCD displeja +  home pozicija kursora
    lcdClrScr();
    // dinamički ispis izbornika i parametre
    lcdprintf("%s\n", imeIzbornika[izbornik]);
    lcdprintf("%s = %d %s", imeParametra[izbornik], p[izbornik],
    imeMjerneJedinice[izbornik]);
    ispisNaLcd = false; // onemogući ispis na LCD
}
```

Ponovno prevedite datoteku `vjezba65.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba65.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 6.6

Napišite program koji će na LCD displeju ispisivati korisnički definirane znakove kao što su dijakritički znakovi č, ć, đ, š i ž. Definirajte navedene znakove. Ispišite na LCD displej tekst `Mikroupravljači su najžešći!` u dva retka i centrirano. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

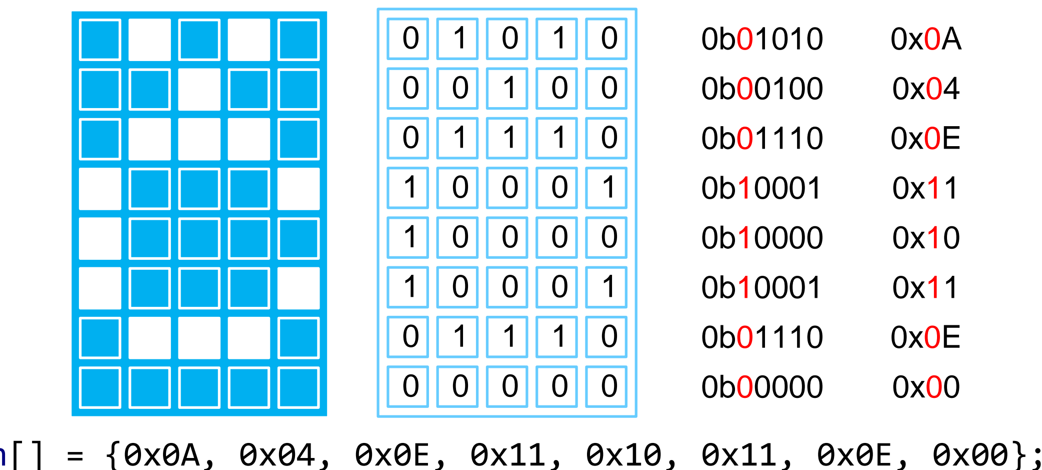
U projektnom stablu otvorite datoteku `vjezba66.cpp`. Omogućite prevođenje datoteke `vjezba66.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba66.cpp` prikazan je programskim kodom 6.19.

Programski kod 6.19: Početni sadržaj datoteke `vjezba66.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja

}

int main(void) {

    init(); // inicijalizacija mikroupravljača


    while (1) { // beskonačna petlja
        _delay_ms(100);
    }

    return 0;
}
```

Živimo u području gdje se koriste znakovi poput č, ć, đ, š i ž. Globalno, engleski jezik je standard u svijetu tehničkih znanosti. Stoga svi LCD displejevi imaju slova engleske abecede koji se nalaze na adresama koje odgovaraju ASCII kodu pojedinog slova. LCD displej ima slobodan memorijski prostor za kreiranje osam vlastitih znakova. Kreirani znakovi se pohranjuju na prvih memorijskih 8 lokacija u LCD displeju i pripadaju im adrese od 0 do 7. Na taj način praktički intervenirate u ASCII tablicu i prvih 8 ASCII kodova prepisujete s vlastitim znakovima. Pojedini se kreirani znak dohvaća adresom, odnosno novim ASCII kodom baš kao i ostali znakovi koristeći sprecifikator formata ispisa `%c`.

Pojedini znak na LCD dsipleju se sastoji od 8 x 5 piksela (8 redova s 5 piksela u retku). Primjer kreiranja vlastitog znaka prikazan je na slici 6.6. Kreiranje znaka se svodi na uključivanje pojedinog piskela znaka. Svaki se redak kodira s binarnom vrijednosti na način da isključeni piksel ima vrijednost 0, a uključeni piksel ima vrijednost 1. Kao rezultat ćemo dobiti broj širine 5 bitova koje je potrebno pretvoriti u dvoznamenkasti heksadecimalan broj. Na primjer, prema slici 6.6 za slovo č uključen je drugi i četvrti piksel, a isključen je prvi, treći i peti piksel što će rezultirati s binarnom kombinacijom 0b01010, odnosno heksadecimalnom kombinacijom 0x0A. Kada se kodiraju svih 8 redaka za znak, dobivene heksadecimalne kombinacije potrebno je prepisati odozgo prema dolje u cjelobrojno polje dimenzije 8 (dno slike 6.6).



Slika 6.6: Kreiranje vlastitog znaka č za LCD displej

Ako si želite olakšati posao definiranja vlastitog znaka, potrebno je koristiti neku postojeću web-aplikaciju za kreiranje znakova. Jedna od njih dostupna je na poveznici `https://maxpromer.github.io/LCD-Character-Creator/`[7]. Pripremu vlastitih znakova možete napraviti u zaglavlju `lcd.h`. Polja definiranih znakova u zaglavlju `lcd.h` prikazani su programskim kodom 6.20.

Programski kod 6.20: Polja definiranih znakova

```
// definiranje vlastitih znakova
const uint8_t customChar1[] = {0x00, 0x08, 0x0E, 0x1F, 0x1F, 0x1E, 0x00, 0x00}; //like
const uint8_t customChar2[] = {0x0A, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00}; //č
const uint8_t customChar3[] = {0x02, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00}; //ć
const uint8_t customChar4[] = {0x02, 0x07, 0x02, 0x0E, 0x12, 0x12, 0x0E, 0x00}; //đ
const uint8_t customChar5[] = {0x0A, 0x04, 0x0E, 0x10, 0x0E, 0x01, 0x1E, 0x00}; //š
const uint8_t customChar6[] = {0x0A, 0x04, 0x1F, 0x02, 0x04, 0x08, 0x1F, 0x00}; //ž
const uint8_t customChar7[] = {0x00, 0x00, 0x0A, 0x00, 0x0E, 0x11, 0x00, 0x00}; // :(
const uint8_t customChar8[] = {0x00, 0x00, 0x0A, 0x00, 0x11, 0x0E, 0x00, 0x00}; // :)
```

Komentarima smo opisali pojedini znak koji se krije iza kodiranih elemenata polja. Predvidjeli smo definiranje osam vlastitih znakova, no po potrebi možete ih napraviti i više. Treba biti svjestan da istovremeno LCD može pohranjivati samo 8 definiranih znakova, no dinamički možete promijeniti neki znak tijekom izvođenja programa. Pokušajte navedene znakove kreirati pomoću gore spomenute web aplikacije i usporedite rezultate.

Kreirani se znakovi u LCD displej upisuju sljedećom funkcijom:

- `void lcdCreateChar(uint8_t addr, const uint8_t *customChar)` - funkcija prima dva argumenta. Prvi argument `addr` jest adresa memorijske lokacije LCD displeja na koju se upisuju definirani znakovi. Moguće adrese su 0, 1, 2, 3, 4, 5, 6 i 7. Drugi argument je pokazivač na definirano polje znakova u zaglavlju `lcd.h`.

Na primjer, ako želimo koristiti slovo č na LCD displeju, nakon inicijalizacije LCD displeja potrebno je pozvati funkciju `lcdCreateChar(1, customChar2);` kojom se slovo č smješta na adresu 1 u LCD displeju. Polje `customChar2`deklarirano je i inicijalizirano u zaglavlju `lcd.h`. Jednom kad definirate znakove i upišete ih u LCD displej, oni tamo ostaju pohranjeni trajno ili do prepisivanja s drugim znakom.

Već smo spomenuli kako funkcija `lcdprintf()` ima istu sintaksu kao standardna funkcija `printf()` programskog jezika C. Tekst koji se ispisuje na LCD displej je, kao i svaki string u

[7] Ako je ova poveznica nekim slučajem istekla, u *Google* upišite *LCD Character Generator* i pronaći ćete brojne druge web-aplikacije.

programskom jeziku C, zaključan s tzv. *null* znakom čiji je ASCII kod 0. Iz tog se razloga jedino memorijska lokacija 0 ne može koristiti u funkciji `lcdprintf()` jer bi prevoditelj znak s ASCII kodom 0 shvatio kao kraj teksta koji ispisuje. Iz ovog se razloga znak na adresi 0 mora ispisivati pomoću funkcije s deklaracijom `void lcdChar(uint8_t data);`. Funkcija `lcdChar()` prima ASCII kod znaka koji želite ispisati te isti prikazuje na mjestu kursora na LCD displeju.

Pretpostavimo da su pozvane funkcije `lcdCreateChar(0, customChar1);` i `lcdCreateChar(1, customChar2);` (polja `customChar1` i `customChar2` deklarirana su i inicijalizirana u zaglavlju `lcd.h`). Navedimo sada primjere poziva funkcije `lcdChar()`:

- `lcdChar(0)` - funkcija će na mjestu kursora na LCD displeja ispisati znak 👍.
- `lcdChar(1)` - funkcija će na mjestu kursora na LCD displeja ispisati znak č.
- `lcdChar(65)` - funkcija će na mjestu kursora na LCD displeja ispisati znak A (ASCII kod za slovo A jest 65).
- `lcdChar(97)` - funkcija će na mjestu kursora na LCD displeja ispisati znak a (ASCII kod za slovo a jest 65).
- `lcdChar(48)` - funkcija će na mjestu kursora na LCD displeja ispisati znak 0 (ASCII kod za broj 0 jest 48).

Sada možemo napraviti primjer programa kojim ćemo ispisati sve definirane znakove programskim kodom 6.20. U inicijalizacijsku funkciju `init()` upišite sljedeće pozive funkcija:

- `lcdCreateChar(0, customChar1);` - zapisivanje znaka 👍 u memoriju LCD displeja na adresu 0.
- `lcdCreateChar(1, customChar2);` - zapisivanje znaka č u memoriju LCD displeja na adresu 1.
- `lcdCreateChar(2, customChar3);` - zapisivanje znaka ć u memoriju LCD displeja na adresu 2.
- `lcdCreateChar(3, customChar4);` - zapisivanje znaka đ u memoriju LCD displeja na adresu 3.
- `lcdCreateChar(4, customChar5);` - zapisivanje znaka š u memoriju LCD displeja na adresu 4.
- `lcdCreateChar(5, customChar6);` - zapisivanje znaka ž u memoriju LCD displeja na adresu 5.
- `lcdCreateChar(6, customChar7);` - zapisivanje znaka :((zakrenut za 90° u smjeru kazaljke na satu) u memoriju LCD displeja na adresu 6.
- `lcdCreateChar(7, customChar8);` - zapisivanje znaka :) (zakrenut za 90° u smjeru kazaljke na satu) u memoriju LCD displeja na adresu 7.

U funkciju `main()` upišite sljedeće pozive funkcija:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdChar(0);` - ispis znaka na adresi 0 (ASCII kod je 0) na LCD displej. Ispisat će se znak

na početku prvog retka.

- `lcdprintf("%c%c%c%c%c%c%c", 1,2,3,4,5,6,7);` - ispis teksta `čćđšž` te emotikona `:(` i `:)` koji su zakrenuti za 90° u smjeru kazaljke na satu. Navedeni znakovi se ispisuju redom na mjesta svih 7 specifikatora formata ispisa `%c` kojima su kao argumenti predane adrese 1, 2, 3, 4, 5, 6 i 7. Na tim adresama su upravo definirani znakovi koji su ispisani.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba66.cpp` treba biti ista kao programski kod 6.21.

Programski kod 6.21: Sadržaj datoteke `vjezba66.cpp` - ispis znakova `čćđšž` te emotikona `:(` i `:)` koji su zakrenuti za 90° u smjeru kazaljke na satu

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // kreiranje samo onih znakova koji su potrebni
    lcdCreateChar(0, customChar1); // znak like (FB)
    lcdCreateChar(1, customChar2); // znak č
    lcdCreateChar(2, customChar3); // znak ć
    lcdCreateChar(3, customChar4); // znak đ
    lcdCreateChar(4, customChar5); // znak š
    lcdCreateChar(5, customChar6); // znak ž
    lcdCreateChar(6, customChar7); // znak :-) (emotikon)
    lcdCreateChar(7, customChar8); // znak :-( (emotikon)
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    // prikazi kreirane znakove
    lcdClrScr(); // obriši LCD
    lcdChar(0); // ispiši znak na adresi 0
    // ispisi znakove na adresama 1, 2, 3, 4, 5, 6 i 7
    lcdprintf("%c%c%c%c%c%c%c", 1,2,3,4,5,6,7);

    while (1) { // beskonačna petlja
        _delay_ms(100);
    }
    return 0;
}
```

Prevedite datoteku `vjezba66.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se tekst `čćđšž` te emotikoni `:(` i `:)` koji su zakrenuti za 90° u smjeru kazaljke na satu. Primijetite da beskonačna petlja `while` u ovom primjeru ne radi ništa (osim kašnjenja).

Sada kada znamo koristiti definirane znakove, možemo ispisati tekst `Mikroupravljači su najžešći!`. Nakon što jednom na LCD displej upišemo vlastite znakove, oni trajno ostaju u memoriji LCD displeja. Za ispis teksta `Mikroupravljači su najžešći!` nije potrebno ponovno zapisati znakove č, ž, š i ć u memoriju LCD displeja pomoću funkcije `lcdCreateChar()`. No, za potrebe teksta `Mikroupravljači su najžešći!` pravit ćemo se da prvi puta koristimo LCD displej s praznim memorijskim prostorom od adrese 0 do 7. U postojećoj funkciji `init()` izbrišite sve pozive funkcija `lcdCreateChar()` za one znakove koji se ne pojavljuju u tekstu `Mikroupravljači su najžešći!`. Ostat će samo pozivi funkcije `lcdCreateChar()` za znakove č, ć, š i ž. Obrišite prethodno pozvane funkcija `lcdChar()` i `lcdprintf()` te napišite sljedeće

naredbe u funkciji `main()`:

- `lcdGotoXY(1,2);` - pozicioniranje kursora za spis u 1. redak i 2. stupac radi centriranja teksta `Mikroupravljači`.
- `lcdprintf("Mikroupravlja%ci\n", 1);` - ispis teksta `Mikroupravljači`. Na mjesto specifikatora formata ispisa `%c` ispisat će se slovo č jer smo kao argument proslijedili adresu 1.
- `lcdGotoXY(2,3);` - pozicioniranje kursora za spis u 2. redak i 3. stupac radi centriranja teksta `su najžešći`.
- `lcdprintf("su naj%ce%c%ci!\n", 5, 4, 2);` - ispis teksta `su najžešći`. Na mjesta 3 specifikatora formata ispisa `%c` ispisat će se slova ž, š i ć jer smo kao argument proslijedili adrese 5, 4 i 2.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba66.cpp` treba biti ista kao programski kod 6.22.

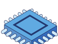
Programski kod 6.22: Sadržaj datoteke `vjezba66.cpp` - ispis teksta `Mikroupravljači su najžešći!` u dva retka

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // kreiranje samo onih znakova koji su potrebni
    lcdCreateChar(1, customChar2); // znak č
    lcdCreateChar(2, customChar3); // znak ć
    lcdCreateChar(4, customChar5); // znak š
    lcdCreateChar(5, customChar6); // znak ž
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    lcdClrScr(); // obriši LCD
    lcdGotoXY(1,2); // ispis u 1. redak, 2. stupac
    lcdprintf("Mikroupravlja%ci\n", 1); // adresa slova č je 1
    lcdGotoXY(2,3); // ispis u 2. redak, 3. stupac
    // adresa slova ž je 5, š je 4 i ć je 2
    lcdprintf("su naj%ce%c%ci!\n", 5, 4, 2);

    while (1) { // beskonačna petlja
        _delay_ms(100);
    }
    return 0;
}
```

Ponovno prevedite datoteku `vjezba66.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se tekst `Mikroupravljači su najžešći!` u dva retka.

Možete pokušati kreirati neke proizvoljne znakove i ispisati ih na LCD displej.

Zatvorite datoteku `vjezba66.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 6.7

Napišite program koji će na LCD displeju ispisivati tekst `Mikroupravljači su najžešći!` u jednom retku. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba67.cpp`. Omogućite prevođenje datoteke `vjezba67.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba67.cpp` prikazan je programskim kodom 6.23.

Programski kod 6.23: Početni sadržaj datoteke `vjezba67.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // kreiranje samo onih znakova koji su potrebni
    lcdCreateChar(1, customChar2); // znak č
    lcdCreateChar(2, customChar3); // znak ć
    lcdCreateChar(4, customChar5); // znak š
    lcdCreateChar(5, customChar6); // znak ž
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    lcdClrScr(); // obrisi LCD displej
    // ispiši tekst dužine 28 znakova
    lcdprintf("Mikroupravlja%ci su naj%ce%c%ci!", 1, 5, 4, 2);

    while (1) { // beskonačna petlja

        _delay_ms(500); // kašnjenje 500 ms
    }
    return 0;
}
```

Tekst `Mikroupravljači su najžešći!` ima ukupno 28 znakova. Vidljivi dio LCD displeja može prikazati 16 znakova, a pojedini redak može pohraniti 40 znakova za prikaz. LCD displej ima mogućnost posmaka teksta u lijevo ili u desno kako bi se svih 40 znakova u pojedinom retku mogli prikazati. U zaglavlju `lcd.h` deklarirane su dvije funkcije za posmak teksta na LCD displeju:

- `lcdShiftLeft()` - funkcija koja kad se pozove pomiče tekst za jedno mjesto u lijevo.
- `lcdShiftRight()` - funkcija koja kad se pozove pomiče tekst za jedno mjesto u desno.

Da bismo cijeli tekst od 28 znakova prikazali na LCD displeju od 16 znakova, potrebno je tekst pomaknuti za jedno mjesto u lijevo 12 puta. Dodatno, napravit ćemo program u kojem će se tekst pomicati lijevo i desno cijelo vrijeme. U programskom kodu 6.23 pomoću funkcije `lcdprintf()` na LCD displej ispisan je (ili u ovom slučaju bolje rečeno poslan je) tekst `Mikroupravljači su najžešći!`. Da bismo pomicali tekst u lijevo i u desno potreban nam je brojač koraka. Na početku funkcije `main()` deklarirajte sljedeću varijablu:

- `uint8_t brojac = 0;` - deklaracija i inicijalizacija varijable cjelobrojnog tipa u kojoj će biti pohranjen broj posmicanja teksta u lijevo i u desno.

U beskonačnu petlju `while` napišite sljedeće:

- otvorite `if else` uvjetovano grananje,
- kao uvjet grananja postavite `(brojac < 12)`,
- ako je uvjet zadovoljen (`if` blok), pozovite funkciju `lcdShiftLeft()`,
- ako uvjet nije zadovoljen (`else` blok), pozovite funkciju `lcdShiftRight()`,
- `brojac++;` - povećanje varijable `brojac` za 1.
- `if(brojac >= 24){brojac = 0;}` - uvjet kojim se varijabla `brojac` drži u rasponu od 0 do 23 (12 posmicanja u lijevo i 12 posmicanja u desno).

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba67.cpp` treba biti ista kao programski kod 6.24.

Programski kod 6.24: Sadržaj datoteke `vjezba67.cpp` - ispis teksta `Mikroupravljači su najžešći!` u jednom retku s posmicanjem teksta

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // kreiranje samo onih znakova koji su potrebni
    lcdCreateChar(1, customChar2); // znak č
    lcdCreateChar(2, customChar3); // znak ć
    lcdCreateChar(4, customChar5); // znak š
    lcdCreateChar(5, customChar6); // znak ž
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    uint8_t brojac = 0;

    lcdClrScr(); // obrisi LCD displej
    // ispiši tekst duzine 28 znakova
    lcdprintf("Mikroupravlja%ci su naj%ce%c%ci!", 1, 5, 4, 2);

    while (1) { // beskonačna petlja
        // ako je brojač manji od 12
        if (brojac < 12) {
            lcdShiftLeft(); // posmakni tekst u lijevo
        } else { // ako je veći i jednak 12
            lcdShiftRight(); // posmakni tekst u desno
        }

        brojac++; // povečaj za 1
        if (brojac >= 24) { // kada dođe do 24
            brojac = 0; // vrati brojač na 0
        }
        _delay_ms(500); // kašnjenje 500 ms
    }
    return 0;
}
```

Prevedite datoteku `vjezba67.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se tekst `Mikroupravljači su najžešći!` u

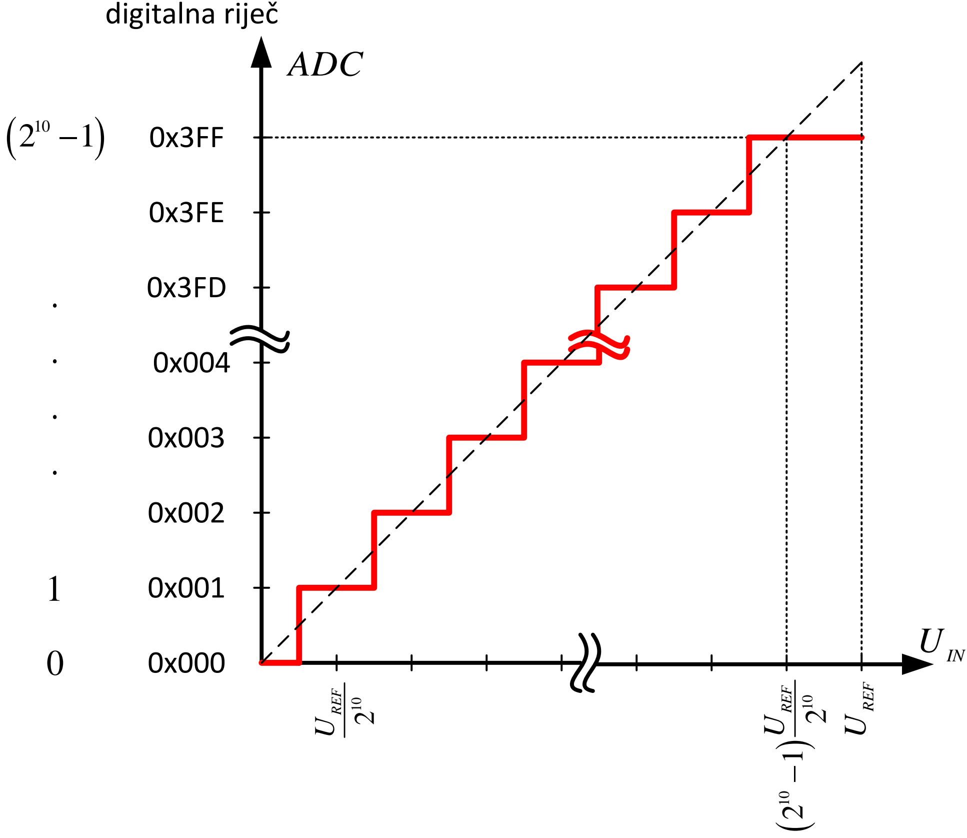
jednom retku i posmicat će se lijevo i desno.

Zatvorite datoteku `vjezba67.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio 7*.

Poglavlje 7

Analogno-digitalna pretvorba

Kada je potrebno prezentirati analogne vrijednosti napona na mikroupravljaču, koristit ćemo analogno-digitalnu (AD) pretvorbu. Na primjer, korištenjem LDR (engl. *Light Dependent Resistor*) otpornika u naponskom dijelilu moguće je mjeriti osvjetljenje u prostoru mjerenjem pada napona na LDR otporniku. Zbog ovakvih svrha, mikroupravljači su opremljeni s AD pretvornicima koji vrijednosti napona pretvaraju u digitalnu riječ. Mapiranje analogne vrijednosti napona u digitalnu riječ u procesu AD pretvorbe prikazano je na slici 7.1.



Slika 7.1: Mapiranje analogne vrijednosti napona u digitalnu riječ u procesu AD pretvorbe ($n = 10$)

Analogna vrijednost napona u rasponu $[0, U_{REF}]$ podijeljena je u 2^n područja, gdje n predstavlja broj bitova koji se koristi za digitalnu riječ nastalu AD pretvorbom analogne

vrijednost napona. Digitalna riječ je u rasponu $[0, 2^n - 1]$. Broj n naziva se rezolucija (razlučivost) AD pretvorbe ili širina digitalne riječi. Tipične vrijednosti za n u svijetu mikroupravljača su 8 i 10. Razlučivost AD pretvorbe na slici 7.1 iznosi $n = 10$.

Najmanja razlika napona koju AD pretvornik može pouzdano razlikovati jest $U_{REF}/2^n$. Jasno je da AD pretvorbom s konačnom razlučivosti ne možemo osigurati beskonačnu točnost prezentacije analogne vrijednosti. Pogreška AD pretvorbe jest $\pm 0.5 U_{REF}/2^n$ što se vidi na slici 7.1. AD pretvorbom nije moguće dobiti referentnu vrijednost napona U_{REF}, već je maksimalni iznos napona koji se može dobiti jednak $(2^n - 1)\frac{U_{REF}}{2^n}$ V. Ako se na ulazu AD pretvornika pojavi vrijednost napona U_{REF}, pogreška AD pretvorbe će biti $U_{REF}/2^n$ (to je jedini slučaj u kojoj pogreška nije spomenutih $\pm 0.5 U_{REF}/2^n$).

Rezultat (digitalna riječ) AD pretvorbe ulaznog napona U_{IN} može se dobiti prema jednadžbi:

$$ADC = \frac{2^n}{U_{REF}} U_{IN} \tag{7.1}$$

Primijetite da je digitalna riječ ADC u jednadžbi 7.1 cjelobrojna pa je i dijeljenje u jednadžbi cjelobrojno. U praksi je potrebno temeljem AD pretvorbe i dobivene digitalne riječi rekonstruirati ulazni napon U_{IN}. Analogna vrijednost ulaznog napona U_{IN} može se izvesti iz jednadžbe 7.1 na sljedeći način:

$$U_{IN} = \frac{U_{REF}}{2^n} ADC \tag{7.2}$$

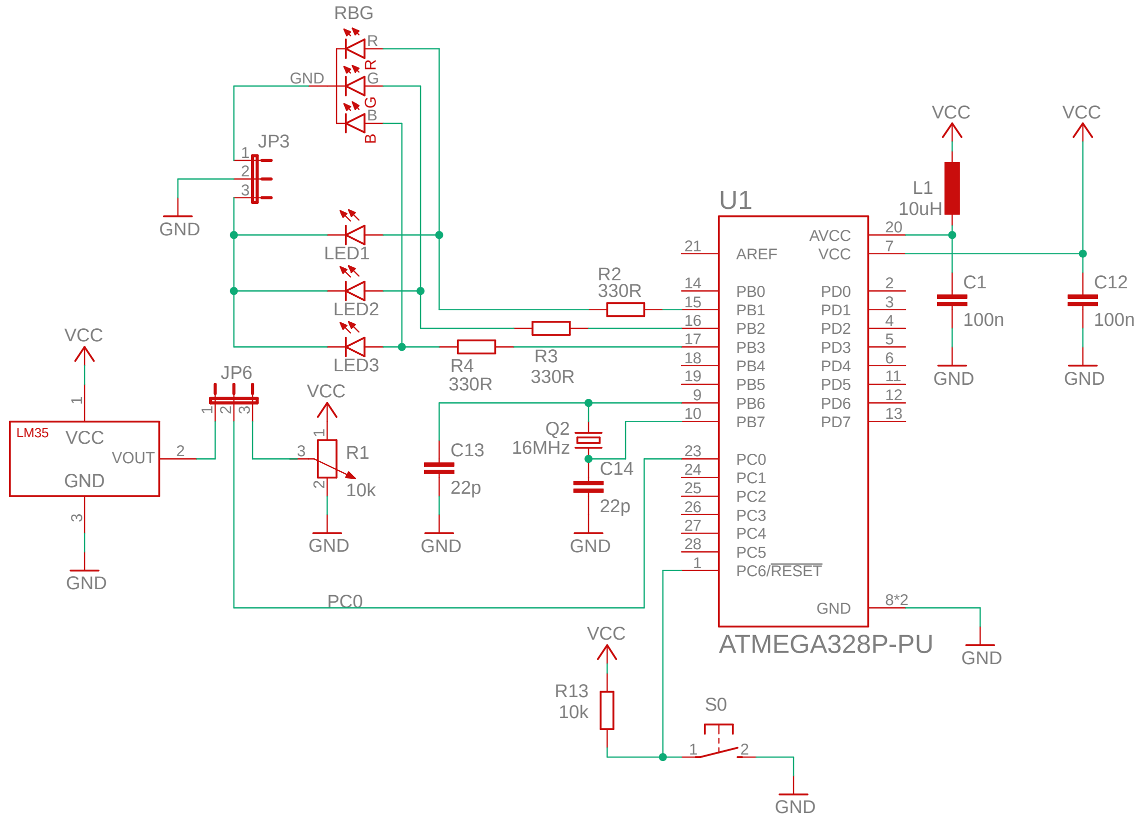
Na analogne ulaze mikroupravljača spajaju se analogni senzori. Analogni senzori mjerni su uređaji koji električne i neelektrične veličine pretvaraju u električne (na primjer: struju u napon (ACS712), tlak u napon (HR202), temperaturu u napon (NTC, LM35, termopar K tipa) i drugi). Izlazni signal s analognih senzora često je napon u rasponu 0 - 5 V ili 0 - 10 V. Taj je napon potrebno mjeriti i prezentirati u mikroupravljaču. Ovisno o referentnom naponu U_{REF} koji može poprimiti različite iznose (najčešće je to VCC koji može biti 3.3 V ili 5 V), ponekad je potrebno napraviti prilagodbu naponskog signala analognom pinu mikroupravljača. Na mikropravljačima često postoji pin koji preslikava iznos referentnog napona U_{REF} pa se takav pin može koristiti i za napajanje analognih senzora (najčešće naponskih dijelila). Mikroupravljači, ovisno o kataloškom broju, mogu imati različiti broj analognih ulaza (na primjer: 5, 8, 12, itd.). Ovi pinovi su višenamjenski i ako se koriste u svrhu AD pretvorbe, ne bi se trebali koristiti kao digitalni ulazi ili izlazi.

7.1 Vježbe - analogno-digitalna pretvorba

Mikroupravljač ATmega328P ima ukupno 6 analognih ulaza raspoređenih na sljedeće pinove: ADC0 (PC0), ADC1 (PC1), ADC2 (PC2), ADC3 (PC3), ADC4 (PC4) i ADC5 (PC5). Navedeni pinovi PC0 - PC5 višenamjenski su pinovi. Osim što se mogu koristiti za AD pretvorbu, mogu biti digitalni ulazi i izlazi te mogu generirati vanjske prekide (nešto više o tome u budućim poglavljima). Dodatno, pinovi PC4 i PC5 mogu biti korišteni za I^{2}C komunikaciju. AD pretvornik u mikroupravljaču ATmega328P pretvara analognu vrijednost napona u digitalnu riječ širine 10 bitova metodom sukcesivne aproksimacije [4]. Dakle, razlučivost AD pretvorbe mikroupravljača ATmega328P jest $n = 10$. Vrijeme AD pretvorbe može biti od 65 do 260 μs.

Shema spajanja potenciometra i temperaturnog senzora LM35 na analogni pin ADC0 prikazana je na slici 7.2. Potenciometar i temperaturni senzor LM35 spojeni su pomoću kratkospojnika `JP6` na pin PC0 (ADC0). Kada se koristi potenciometar, kratkospojnik `JP6` spojen je između trnova 2 i 3. Temperaturni senzor LM35 povezan je na pin ADC0 ako je kratkospojnik `JP6` spojen između trnova 1 i 2. Na shemi sa slike 7.2 prikazane su i LED diode

koje su povezane na digitalne pinove PB1, PB2 i PB3. U ovoj vježbi koristit ćemo i LCD displej čija je shema spajanja prikazana na slici 6.1.



Slika 7.2: Shema spajanja potenciometra i temperaturnog senzora LM35 na analogni pin ADC0

AD pretvornik mikroupravljača ATmega328P ima zasebno napajanje. Napajanje za njega dovodi se na pinove mikroupravljača GND i AVCC (slika 7.2). Napajanje mikroupravljača i AD pretvornika na razvojnom okruženja sa slike 2.1 isto je i iznosi 5 V.

AD pretvornik u procesu pretvorbe analogne vrijednosti napona u digitalnu riječ koristi referentni napon U_{REF}. Prema navedenom u uvodu ovog poglavlja, analogna vrijednost napona u rasponu $[0, U_{REF}]$ V pretvara se u digitalnu riječ u rasponu $[0, 2^{10}] = [0, 1023]$. Referentni naponski izvor odabire se pomoću bitova `REFS0` i `REFS1` u registru `ADMUX` prema tablici 23-3 u literaturi [2]. Izvor referentnog napona U_{REF} može biti:

- napon napajanja AD pretvornika koji se dovodi na AVCC pin mikroupravljača ATmega328P (`REFS1 = 0`, `REFS0 = 1`),
- unutarnji izvor napona iznosa 1,1 V (`REFS1 = 1`, `REFS0 = 1`),
- napon koji se dovodi na AREF pin mikroupravljača ATmega328P (`REFS1 = 0`, `REFS0 = 0`).

Ako je kao izvor referentnog napona izabran AVCC ili unutarnji 1,1 V, tada se napon referentnog naponskog izvora prosljeđuje na AREF pin mikroupravljača ATmega328P.

Rezultat AD pretvorbe na pinu ADCi može se dobiti primjenom relacije 7.1 na sljedeći način:

$$ADCi = \frac{U_{ADCi} \cdot 1024}{U_{REF}} \tag{7.3}$$

gdje je:

- $ADCi$ - 10-bitna digitalna riječ AD pretvorbe na pinu ADCi ($i = 0, 1, 2, 3, 4, 5$),
- U_{ADCi} - analogna vrijednost napona na pinu ADCi,
- U_{REF} - referentni iznos napona.

Pretpostavimo da je referentni iznos napona $U_{REF} = 5$ V (kao naponski izvor AD pretvornika izabran je pin AVCC). Neka analogna vrijednost napona na pinu `ADC0` iznosi 3,85 V ($U_{ADC0} =$ 3,85 V). Prema relaciji (7.3), 10-bitna digitalna riječ AD pretvorbe na pinu `ADC0` (PC0) iznosi:

$$ADC0 = \frac{U_{ADC2} \cdot 1024}{5} = \frac{3,85 \cdot 1024}{5} = 788. \tag{7.4}$$

Pretpostavimo da analognu vrijednost napona generira analogni senzor koji mjeri neku fizikalnu veličinu. Ako poznajemo ovisnost fizikalne veličine o naponu senzora, tada da bismo izračunali fizikalnu veličinu, moramo moći izmjeriti napon na analognom ulazu. Napon na pinu ADCi temeljem AD pretvorbe možemo dobiti primjenom relacije 7.2 na sljedeći način:

$$U_{ADCi} = \frac{ADCi \cdot U_{REF}}{1024}. \tag{7.5}$$

Pretpostavimo da je referentni iznos napona $U_{REF} = 5$ V (kao naponski izvor AD pretvornika izabran je pin AVCC). Neka digitalna riječ kao rezultat AD pretvorbe na pinu `ADC0` iznosi 244. Prema relaciji (7.5), na pinu `ADC0` (PC0) iznosi:

$$U_{ADC0} = \frac{ADC0 \cdot 5\,V}{1024} = \frac{244 \cdot 5\,V}{1024} = 1.1914\,\text{V}. \tag{7.6}$$

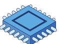
Frekvencija rada AD pretvorbe konfigurira se pomoću bitova `ADPS0`, `ADPS1` i `ADPS2` u registru `ADCSRA` prema tablici 23-5 u literaturi [2].

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `ADC.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `ADC.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\ADC\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Analogno-digitalna pretvorba`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

Prije korištenja AD pretvorbe na mikroupravljaču, potrebno je namjestiti referentni napon pomoću funkcije `void adcSetReference(uint8_t ref)`. Ova funkcija prima argument `ref` koji može poprimite sljedeće vrijednosti definirane konstantama:

- `ADC_REFERENCE_AREF` - referentni napon je napon pina AREF,
- `ADC_REFERENCE_AVCC` - referentni napon je napon pina AVCC,
- `ADC_REFERENCE_11V` - referentni napon iznosi 1,1 V.

Ako se ne odabere drugačije, inicijalizacijom AD pretvorbe referentni napon će biti napon pina AVCC. Frekvencija rada AD pretvornika konfigurira se automatski u ovisnosti o frekvenciji ranog takta.

 Vježba 7.1

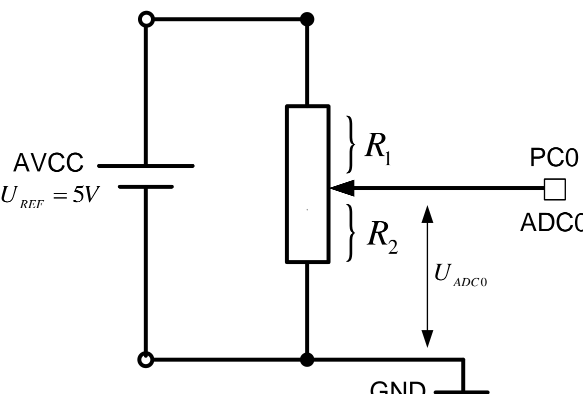
Napišite program koji će na početku prvog retka LCD displeja ispisati rezultat AD pretvorbe na pinu ADC0 (PC0), a na početku drugog retka napon koji se trenutno mjeri na pinu ADC0 (PC0). AD pretvorbu provedite jednom svakih 500 ms. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3. Nazivna vrijednost otpora potenciometra jest 10 kΩ. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba71.cpp`. Omogućite prevođenje datoteke `vjezba71.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba71.cpp` prikazan je programskim kodom 7.1.

Programski kod 7.1: Početni sadržaj datoteke `vjezba71.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

    }

    return 0;
}
```

Prije izrade vježbe, uvjerite se da je na razvojnom okruženju sa slike 2.1 kratkospojnik `JP6` spojen između trnova 2 i 3 kako bi na pin ADC0 bio spojen potenciometar. Potenciometar kao dijelilo napona spojeno na pin ADC0 i s referentnim naponom $U_{REF} = 5$ V prikazan je na slici 7.3.



Slika 7.3: Potenciometar kao dijelilo napona spojeno na pin ADC0 i s referentnim naponom $U_{REF} = 5$ V

Potenciometar je promjenjivo dijelilo napona koje ima tri pina. Dva pina su za ulazni napon (u našem slučaju $U_{REF} = 5$ V) i jedan pin za izlazni napon (u našem slučaju U_{ADC0}). Ovisno

o zakretu potenciometra od 0 do 100%, izlazni napon se linearno[1] mijenja od 0 do $U_{REF} = 5$ V. Zbog svoje karakteristike, potenciometar se često koristi za namještanje referentnih vrijednosti u sustavu (intenzitet LED diode, brzina vrtnje ili pozicija DC motora itd.).

Problem koji moramo riješiti u ovoj vježbi jest napraviti AD pretvorbu analognog naponskog signala na pinu ADC0. Dodatno, potrebno je izračunati kolika je vrijednost napona na pinu ADC0. Mikroupravljač analognu vrijednost napona AD pretvorbom pretvara u 10-bitnu digitalnu riječ. Naš zadatak je dohvatiti tu digitalnu riječ koja se dobije sukcesivnom aproksimacijom AD pretvornika.

Funkcije koje se koriste za AD pretvorbu definirane su u zaglavlju `adc.h`. U programskom kodu 7.1 uključite zaglavlje `adc.h` naredbom `#include "ADC/adc.h"`. Prije korištenja AD pretvorbe u mikroupravljaču, potrebno je inicijalizirati AD pretvornik funkcijom `adcInit()` koju dodajte u tijelo funkcije `init()` u programskom kodu 7.1. Funkcija `adcInit()` konfigurira referentni napon (postavlja ga na AVCC) i frekvenciju AD pretvorbe te omogućuje AD pretvorbu.

Korisne funkcije koje su deklarirane u zaglavlju `adc.h` su:

- `uint16_t adcRead(uint8_t ch)` - funkcija koja pokreće AD pretvorbu na kanalu koji je zadan varijablom `ch`. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`. Funkcija `adcRead()` vraća 10-bitnu digitalnu riječ (rezultat AD pretvorbe na kanalu `ch`) u rasponu od 0 do 1023.

- `float adcReadVoltage(uint8_t ch)` - funkcija koja pokreće AD pretvorbu na kanalu koji je zadan varijablom `ch` i vraća vrijednost napona na pinu `ch`. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`. Funkcija `adcReadVoltage()` vraća realnu vrijednost napona na pinu `ch` u rasponu od 0 do U_{REF} V.

- `float adcConvertToVoltage(uint16_t adc)` - funkcija koja radi pretvorbu digitalne riječi `adc` koja je neposredno prije dobivena AD pretvorbom u vrijednost napona u rasponu od 0 do U_{REF} V (ovu vrijednost funkcija vraća).

- `float adcReadScale0To100(uint8_t ch)` - funkcija koja pokreće AD pretvorbu na kanalu koji je zadan varijablom `ch` i vraća vrijednost AD pretvorbe skaliranu u interval od 0 do 100. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`. Funkcija `adcReadScale0To100()` vraća realnu vrijednost u rasponu od 0 do 100. Ova funkcija može se koristiti za generiranje referentne vrijednosti neke veličine od 0 do 100%.

- `float adcScale0To100(uint16_t adc)` - funkcija koja prima rezultat AD pretvorbe `adc` i skalira ga u interval od 0 do 100. Funkcija `adcScale0To100()` vraća realnu vrijednost u rasponu od 0 do 100.

- `float adcReadScaleAToB(uint8_t ch, float a, float b)` - funkcija koja prima tri argumenta: kanal koji je zadan varijablom `ch`, donju granicu intervala koja je zadana s varijablom `a` i gornju granicu intervala koja je zadana s varijablom `b`. Funkcija pokreće AD pretvorbu na kanalu koji je zadan varijablom `ch` i vraća vrijednost AD pretvorbe skaliranu u interval od a do b. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`. Funkcija `adcReadScaleAToB()` vraća realnu vrijednost u rasponu od a do b.

- `float adcScaleAToB(uint16_t adc, float a, float b)` - funkcija koja prima tri argumenta: rezultat AD pretvorbe koji je zadan varijablom `adc`, donju granicu intervala

[1] Potenciometar koji mi koristimo ima linearnu karakteristiku. Postoje potenciometri s logaritamskom ovisnošću promjene napona o zakretu.

koja je zadana s varijablom `a` i gornju granicu intervala koja je zadana s varijablom `b`. Funkcija `aadcScaleAToB()` vrijednost AD pretvorbe `adc` skalira u rasponu od a do b i vraća tu skaliranu vrijednost.

- `uint16_t adcReadAverage(uint8_t ch, uint8_t n)` - funkcija koja prima dva argumenta: kanal koji je zadan varijablom `ch` i broj AD pretvorbi koji će se usrednjiti zadan varijablom `n`. Ova funkcija pokreće ukupno n AD pretvorbi na kanalu koji je zadan varijablom `ch` svakih 10 ms. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`. Broj n može biti u rasponu od 1 do 64. Funkcija `adcReadAverage()` vraća prosječnu vrijednost od n 10-bitnih digitalnih riječi u rasponu od 0 do 1023. Ova funkcija može se koristiti za filtriranje zašumljenog analognog naponskog signala.

Pokažimo sada nekoliko primjera korištenja gore navedenih funkcija:

- `adcRead(ADC0)` - rezultat AD pretvorbe na kanalu `ADC0`,
- `adcRead(ADC3)` - rezultat AD pretvorbe na kanalu `ADC3`,
- `adcReadVoltage(ADC5)` - vrijednost napon na pinu `ADC5`,
- `adcConvertToVoltage(500)` - vrijednost napon na analognom pinu za rezultat AD pretvorbe koji iznosi 500,
- `adcReadScale0To100(ADC4)` - postotak napona na pinu `ADC4` u odnosu na napon U_{REF},
- `adcScale0To100(320)` - rezultat AD pretvorbe od 0 do 1023 skaliran u raspon od 0 do 100,
- `adcReadScaleAToB(ADC1, 4.0, 20.0)` - rezultat AD pretvorbe na pinu `ADC1` od 0 do 1023 skaliran u raspon od 4.0 do 20.0,
- `adcScaleAToB(114, -10, 10)` - rezultat AD pretvorbe iznosa 114 skaliran u raspon od -10.0 do 10.0,
- `adcReadAverage(ADC3, 20)` - srednja vrijednost 20 AD pretvorbi na kanalu `ADC3`.

Za referentni iznos napona U_{REF} odabran je napon napajanja AD pretvornika AVCC. Taj napon spojen je i na potenciometar nazivne vrijednosti 10 kΩ (slika 7.2). Sada možemo krenuti s pisanjem programa kojim ćemo provesti AD pretvorbu na pinu `ADC0`.

U programski kod 7.1 ispod poziva funkcije `init()` napišite sljedeće deklaracije varijabli:

- `uint16_t adc0` - deklaracija cjelobrojne varijable u koju će se spremati rezultat AD pretvorbe na pinu `ADC0`,
- `float Uadc0` - deklaracija realne varijable u koju će se spremati napon na pinu `ADC0`.

U beskonačnu petlju `while` dodajte sljedeće naredbe:

- `adc0 = adcRead(ADC0);` - pozivanje funkcije za AD pretvorbu na kanalu (pinu) `ADC0`. Rezultat AD pretvorbe (10-bitna riječ) sprema se u varijablu `adc0`.
- `Uadc0 = adc0 / 1024.0f * 5.0f;` - izračun napona na pinu `ADC0` prema relaciji 7.5. Kako desna strana izraza ne bi provela cjelobrojno dijeljenje (što bi uzrokovalo gubitkom informacije), važno je za konstante 1024 i 5 koristiti literale 1024.0f i 5.0f kako bi prevoditelj

znao da se radi o realnim brojevima i proveo dijeljenje s realnim brojevima.

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("ADC0: %d\n", adc0);` - ispis teksta `ADC0:` te rezultata AD pretvorbe `adc0`.
- `lcdprintf("UADC0: %.6fV", Uadc0);` - ispis teksta `UADC0:` te realne varijable `Uadc0` na 6 decimalnih mjesta i mjernu jedinicu `V`.
- `_delay_ms(500);` - kašnjenje od 500 ms.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba71.cpp` treba biti ista kao programski kod 7.2.

Programski kod 7.2: Početni sadržaj datoteke `vjezba71.cpp` - program kojim se provodi AD pretvorba na pinu `ADC0` te izračun napona na pinu `ADC0` (prvi način)

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint16_t adc0; // digitalna rijec ADC0
    float Uadc0; // napon na pinu ADC0

    while (1) { // beskonačna petlja
        adc0 = adcRead(ADC0); // AD pretvorba na pinu ADC0
        Uadc0 = adc0 / 1024.0f * 5.0f; // napon na pinu ADC0
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("ADC0: %d\n", adc0);
        lcdprintf("UADC0: %.6fV\n", Uadc0);
        _delay_ms(500); // kasnjenje 500 ms
    }
    return 0;
}
```

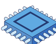
Prevedite datoteku `vjezba71.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se rezultat AD pretvorbe i napon na pinu `ADC0`.

Izračun napona `Uadc0 = adc0 / 1024.0f * 5.0f` možemo zamijeniti s pozivom jedne od dviju funkcija:

- `Uadc0 = adcReadVoltage(ADC0);` - pokretanje AD pretvorbe na pinu `ADC0` i pretvorbe digitalne riječi u napon na pinu `ADC0`. Pri ovom pristupu ponovno se pokreće AD pretvorba, iako je ona provedena naredbom koja se nalazi u programskoj liniji iznad.
- `Uadc0 = adcConvertToVoltage(adc0);` - pokretanje pretvorbe digitalne riječi `adc0` u napon na pinu `ADC0`. Korištenjem ove funkcije samo jednom se poziva AD pretvorba na pinu `ADC0`.

Promijenite programski kod 7.2 na način da koristite obje funkcije. Ponovno prevedite datoteku `vjezba71.cpp` (za svaku primjenu gore navedenih funkcija) u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba71.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 7.2

Napišite program koji će na početku prvog retka LCD displeja ispisati filtriran rezultat AD pretvorbe na pinu ADC0 (PC0), a na početku drugog retka skaliranu vrijednost filtriranog rezultata AD pretvorbe u raspon 0 - 100. Vrijednosti na LCD displej ispisujte samo ako se one mijenjaju. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3. Nazivna vrijednost otpora potenciometra jest 10 kΩ. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba72.cpp`. Omogućite prevođenje datoteke `vjezba72.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba72.cpp` prikazan je programskim kodom 7.3.

Programski kod 7.3: Početni sadržaj datoteke `vjezba72.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint16_t adc0; // digitalna riječ ADC0
    uint16_t adc0Old; // digitalna riječ ADC0
    float adc0Scale; // napon na pinu ADC0
    bool ispisNaLcd = true; // omogućavanje ispisa

    while (1) { // beskonačna petlja


        // uvjetni ispis na LCD displej
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kurso
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("ADC0: %d\n", adc0);
            lcdprintf("Scale: %.2f%%\n", adc0Scale);
            ispisNaLcd = false; //onemogući ispis na LCD
        }
    }
    return 0;
}
```

U ovoj vježbi provest ćemo filtriranje rezultata AD pretvorbe kako bi se smanjio potencijalni šum superponiran na analognu vrijednosti napona. Filtrirani rezultat AD pretvorbe potrebno

je ispisati na LCD displeju. Dodatno, potrebno je osigurati ispis skalirane vrijednosti filtiranog rezultata AD pretvorbe u rasponu od 0 do 100.

U programskom kodu 7.3 prikazana je inicijalizacija mikroupravljača, deklarirane su varijable koje se koriste te je prikazan uvjetovani ispis na LCD displeju na isti način kako smo to radili u prošlom poglavlju.

Ispod poziva funkcije `init()` u programskom kodu 7.3 deklarirane su sljedeće varijable:

- `uint16_t adc0` - deklaracija cjelobrojne varijable u koju će se spremati rezultat AD pretvorbe na pinu `ADC0`,
- `uint16_t adc0Old` - deklaracija cjelobrojne varijable u koju će se spremati stara vrijednost rezultata AD pretvorbe na pinu `ADC0`,
- `float adc0Scale` - deklaracija realne varijable u koju će se spremati skalirana vrijednost rezultata AD pretvorbe na pinu `ADC0` u rasponu od 0 do 100,
- `bool ispisNaLcd = true` - deklaracija i inicijalizacija *Boolean* varijable koja služi za omogućenje i onemogućenje ispisa na LCD displej.

U beskonačnu petlju `while` dodajte sljedeće naredbe:

- `adc0 = adcReadAverage(ADC0, 10);` - pozivanje funkcije za izračun srednje vrijednosti 10 AD pretvorbi na kanalu (pinu) `ADC0`. Filtrirani rezultat AD pretvorbe (10-bitna riječ) sprema se u varijablu `adc0`. Broj uzoraka za izračun srednje vrijednosti može biti i različit od 10 (na primjer 50).
- `adc0Scale = adcScale0To100(adc0);` - skaliranje prethodno dobivene digitalne riječi `adc0` u raspon od 0 do 100.

Iza navedenih poziva funkcija dodajte sljedeći uvjetovani blok:

- `if ((adc0 != adc0Old)) { }` - provjera da li je novi rezultat filtrirane AD pretvorbe različit od starog rezultata. Ako je došlo do promjene, potrebno je omogućiti ispis na LCD displej naredbom `ispisNaLcd = true;` i osvježiti staru vrijednost rezultata AD pretvorbe naredbom `adc0Old = adc0;`. Navedene dvije naredbe upišite unutar uvjetovanog bloka.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba72.cpp` treba biti ista kao programski kod 7.4.

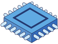
Programski kod 7.4: Početni sadržaj datoteke `vjezba72.cpp` - program kojim se provodi filtriranje rezultata AD pretvorbe na pinu `ADC0` te skaliranje tog rezultata u raspon od 0 do 100

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint16_t adc0; // digitalna rijec ADC0
    uint16_t adc0Old; // digitalna rijec ADC0
    float adc0Scale; // napon na pinu ADC0
```

```
    bool ispisNaLcd = true; // omogućavanje ispisa

    while (1) { // beskonačna petlja
        // AD pretvorba s usrednjavanjem 10 uzastopnih mjerenja
        adc0 = adcReadAverage(ADC0, 10);
        // skaliranje rezultata AD pretvorbe u raspon 0 do 100
        adc0Scale = adcScale0To100(adc0);

        // ako je nova vrijednost adc0 razlicita od stare
        if ((adc0 != adc0Old)) {
            ispisNaLcd = true; // omoguci ispis na LCD
            adc0Old = adc0; // osvjezi staru vrijednost adc0
        }
        // uvjetni ispis na LCD displej
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("ADC0: %d\n", adc0);
            lcdprintf("Scale: %.2f%%\n", adc0Scale);
            ispisNaLcd = false; //onemoguci ispis na LCD
        }
    }
    return 0;
}
```

Prevedite datoteku `vjezba72.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se filtrirani rezultat AD pretvorbe te skalirana vrijednost rezultata u rasponu od 0 do 100%.

Zatvorite datoteku `vjezba72.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 7.3

Napišite program kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini i ispisivati je u °C na LCD displeju. Dodatno, napišite dio programa za signalizaciju LED diodama na sljedeći način:

- ako je temperatura manja od 22 °C, neka je uključena samo zelena LED dioda,
- ako je temperatura veća i jednaka 22 °C i manja od 25 °C, neka su uključene zelena i žuta LED dioda,
- ako je temperatura veća i jednaka 25 °C, neka su uključene sve LED diode.

Vrijednosti na LCD displej ispisujte samo ako se one mijenjaju, ali ne češće od svakih jednu sekundu. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Prema shemi na slici 7.2, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P.

U projektnom stablu otvorite datoteku `vjezba73.cpp`. Omogućite prevođenje datoteke `vjezba73.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba73.cpp` prikazan je programskim kodom 7.5.

Programski kod 7.5: Početni sadržaj datoteke `vjezba73.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    DDRB |= (1 << PB1); // PB1 konfiguriran kao izlazni pin
    DDRB |= (1 << PB2); // PB2 konfiguriran kao izlazni pin
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint16_t adc0; // digitalna riječ ADC0
    uint16_t adc0Old; // digitalna riječ ADC0
    float T; // temperatura u okolini senzora LM35
    bool ispisNaLcd = true; // omogućavanje ispisa
    int brojac_vremena = 0; // brojač za sekunde

    while (1) { // beskonačna petlja

        // uvjetni ispis na LCD displej i izmjena stanja LED dioda
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("T = %.2f%c", T, 223);
            ispisNaLcd = false; //onemogući ispis na LCD
        }
    }
    return 0;
}
```

Temperaturni senzor LM35 visoko je precizni senzor temperature s mjernim opsegom od -55°C do 150 °C [5]. Na izlaznom pinu `VOUT`, temperaturni senzor LM35 generira napon U_{OUT} koji je proporcionalan temperaturi u okolini senzora s konstantom proporcionalnosti koja iznosi 10 mV/°C. Napon U_{OUT} može se izračunati prema sljedećoj jednadžbi:

$$U_{OUT} = T \cdot 10 \frac{\mathtt{mV}}{^\circ\mathtt{C}} = T \cdot 0,01 \frac{\mathtt{V}}{^\circ\mathtt{C}}. \tag{7.7}$$

Temperaturu okoline iz jednadžbe (7.7) možemo izračunati na sljedeći način:

$$T = U_{OUT} \cdot 100 \frac{^\circ\mathtt{C}}{\mathtt{V}}. \tag{7.8}$$

Na primjer, ako je izlazni napon temperaturnog senzora LM35 jednak 235 mV, temperatura u njegovoj okolini jest 23,5 °C. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji mora biti spojen između trnova 1 i 2. Provjerite da li je kratkospojnik `JP6` postavljen između trnova 1 i 2!

Pin `VOUT` temperaturnog senzora LM35 možemo spojiti na bilo koji analogni pin mikroupravljača. AD pretvorbom mjerimo napon na pinu `VOUT` temperaturnog senzora LM35 te pomoću relacije (7.8) izračunamo temperaturu u okolini temperaturnog senzora LM35.

Analogna vrijednost napona U_{OUT} temperaturnog senzora LM35 mjerit ćemo pomoću AD pretvorbe na pinu `ADC0` prema sljedećoj jednadžbi:

$$U_{OUT} = U_{ADC0} = \frac{ADC0 \cdot U_{REF}}{1024}. \tag{7.9}$$

Temperaturu T [°C] u okolini temperaturnog senzora LM35 izračunat ćemo na sljedeći način:

$$T = U_{ADC0} \cdot 100 = \frac{ADC0 \cdot U_{REF}}{1024} 100. \tag{7.10}$$

Problem koji moramo riješiti u ovoj vježbi jest prezentirati temperaturu okoline senzora LM35 te osigurati da se ispis na LCD displej provodi samo pri promjeni temperature, ali ne češće od jednom u sekundi. Vremenski zahtjev je ciljano postavljen radi problema koji se događa pri promjeni temperature koja se nalazi na granici između dvije digitalne riječi. To je čest problem kod analognih senzora. Stoga se u prijelazi između dviju diskretnih razina temperature javljaju brze promjene (na primjer, digitalna riječ izmjenjuje vrijednosti 50 i 51 dok se ne ustabili na jednoj od vrijednosti). Prikaz bi na LCD displeju u ovom periodu izmjene digitalne riječi bio vrlo nestabilan.

U programskom kodu 7.5 prikazana je inicijalizacija mikroupravljača (inicijalizirani su LCD displej i AD pretvornik te su konfigurirani pinovi PB1, PB2 i PB3 kao izlazni za LED diode), deklarirane su varijable koje se koriste te je prikazan uvjetovani ispis na LCD displeju na isti način kako smo to radili u prošloj vježbi.

Ispod poziva funkcije `init()` u programskom kodu 7.5 deklarirane su sljedeće varijable (opisat ćemo samo one koje se razlikuju od prethodne vježbe):

- `float T` - deklaracija realne varijable u koju će se temperatura okoline senzora LM35.
- `int brojac_vremena = 0` - deklaracija i inicijalizacija cjelobrojne varijable kojom ćemo mjeriti broj prolaska kroz beskonačnu `while` petlju koja će imati kašnjenje od 10 ms. Ukupno 100 prolaza kroz `while` petlju iznosit će vrijeme od jedne sekunde[2].

U projektnom stablu nalazi se mapa `Senzori` u kojoj se nalazi zaglavlje `lm35.h`. Ovo zaglavlje sadrži samo dvije funkcije kojima se može dobiti temperatura okoline senzora LM35. Definicije funkcija koje se nalaze u ovom zaglavlju su:

- `float readTempLM35(uint8_t ch)` - funkcija koja pokreće AD pretvorbu na kanalu koji je zadan varijablom `ch` i vraća realnu vrijednost temperature senzora LM35 u °C koji je spojen na pin `ch`. Vrijednosti koje se mogu koristiti za kanal `ch` su `ADC0`, `ADC1`, `ADC2`, `ADC3`, `ADC4` i `ADC5`.
- `float adcConvertToTempLM35(uint16_t adc)` - funkcija koja radi konverziju digitalne riječi `adc` koja je neposredno prije dobivena AD pretvorbom u realnu vrijednost temperature senzora LM35 u °C.

U datoteku `vjezba73.cpp` uključite zaglavlje `lm35.h` naredbom `#include "Senzori/lm35.h"`. Dodatno, radi lakšeg praćenja programskog koda napravit ćemo konstantu naredbom `#define LM35 ADC0` kako bismo u programskom kodu umjesto pina `ADC0` mogli koristiti naziv senzora `LM35`.

Varijabla `ispisNaLcd` inicijalno je postavljena na vrijednost `true` kako ispis na LCD displeju ne bi kasnio za uključenjem sustava jednu sekundu. Iz tog razloga potrebno je izmjeriti temperaturu okoline odmah prije ulaska u beskonačnu `while` petlju. U programski kod 7.5, neposredno prije beskonačne `while` petlje, upišite sljedeću naredbu:

[2] Ovo je trenutno najbolje što znamo napraviti, a vrijeme će zbog niza naredbi koji se izvodi biti nešto veće od jedne sekunde.

- `T = readTempLM35(LM35);` - pozivanje funkcije za čitanje temperature okoline senzora LM35 i pohrana u varijablu `T`.

Funkcija za kašnjenje `_delay_ms()` zaustavlja program na mjestu poziva i čeka da istekne zadano vrijeme. U tom vremenu ništa se unutar beskonačne `while` petlje neće obrađivati. Kada bi to vrijeme bilo 1000 ms, naredbe unutar beskonačne `while` petlje provodile bi se samo jednom u sekundi. Zbog toga često nije moguće pročitati pritisnuto tipkalo ili promjenu stanja senzora na vrijeme[3]. Bolji pristup jest da unutar beskonačne `while` petlje napravimo kašnjenje od 10 ms te kašnjenje od jedne sekunde ostvarimo tako da beskonačnu `while` petlju izvedemo 100 puta što brojimo pomoću brojača. Brojač svakih 100 puta resetiramo na 0. Na taj će način beskonačna `while` petlja niz naredaba unutar sebe izvršiti 100 puta u jednoj sekundi što će omogućiti očitanje pritiska na tipkalo i druge slične aktivnosti senzora. Stoga, na kraj beskonačne `while` petlje napišite poziv funkcije `_delay_ms(10);`.

Na početak beskonačne `while` napisat ćete programski kod prema sljedećim koracima:

- otvorite uvjetovani blok `if (++brojac_vremena == 100) { }`. Ovim blokom provjerava se da li je varijabla `brojac_vremena` dosegla vrijednost 100 čime se osigurava provedba uvjetovanog bloka jednom u sekundi.

- unutar uvjetovanog bloka upišite sljedeće naredbe:
  - `adc0 = adcRead(LM35);` - pozivanje funkcije za AD pretvorbu na kanalu `LM35` (odnosno `ADC0`). Rezultat AD pretvorbe (10-bitna riječ) sprema se u varijablu `adc0`.
  - `T = adcConvertToTempLM35(adc0);` - pozivanje funkcije koja prima rezultat AD pretvorbe `adc0` i vraća realnu vrijednost temperature okoline senzora LM35.
  - `brojac_vremena = 0;` - postavljanje varijable `brojac_vremena` na iznos 0 kako bi se ponovno moglo izmjeriti vrijeme od jedne sekunde.

Ispis na LCD displej provodi se samo ako se promijenila temperatura okoline i ako je prošlo minimalno jedna sekunda od posljednjeg mjerenja temperature. Da li se neka varijabla promijenila, do sada smo detektirali tako da smo uspoređivali novu i staru vrijednost te varijable. Ne preporučuje se da se uspoređuju realne varijable zbog premale preciznosti tipa `float` što može dovesti do krivih logičkih rezultata. Upravo zato ćemo uspoređivati da li je došlo do promjene varijable `adc0`, a ne varijable `T`.

U nastavku beskonačne `while` petlje provedite sljedeće korake:

- otvorite uvjetovani blok `if ((adc0 != adc0Old) && (brojac_vremena == 0)) { }`. Ovaj uvjetovani blok provodi se ako je prošlo vrijeme od jedne sekunde i ako se promijenila vrijednost varijable.

- unutar uvjetovanog bloka upišite sljedeće naredbe:
  - `ispisNaLcd = true;` - osigurava se ispis na LCD displeju,
  - `adc0Old = adc0;` - osvježava se stara vrijednost rezultata AD pretvorbe.

Preostaje nam napraviti programski odsječak za signalizaciju pomoću LED dioda. Promjenu stanja LED dioda potrebno je napraviti kada se vrijednost temperature promijeni i kada prođe jedna sekunda. Uvjet je identičan kao i za ispis temperature na LCD displej, stoga se signalizacija LED diodama može provesti unutar uvjetovanog bloka `if (ispisNaLcd) { }`.

Unutar uvjetovanog bloka `if (ispisNaLcd) { }` napišite programski kod prema sljedećim

[3] Ovaj problem se inače rješava prekidima.

koracima:

- otvorite uvjetovani blok `if (T < 22.0) { }`. Ovim blokom provjerava se da li je temperatura manja od 22 °C i uključuje se zelena LED dioda, a ostale LED diode se isključuju. Unutar ovog uvjetovanog bloka upišite sljedeće naredbe:
  - `PORTB &= ~(1 << PB1);` - isključivanje crvene LED diode,
  - `PORTB &= ~(1 << PB2);` - isključivanje žute LED diode,
  - `PORTB |= (1 << PB3);` - uključivanje zelene LED diode.
- otvorite uvjetovani blok `if (T >= 22.0 && T < 25.0) { }`. Ovim blokom provjerava se da li je temperatura veća i jednaka 22 °C i manja od 25 °C i uključuju se zelena i žuta LED dioda, a crvena LED dioda se isključuje. Unutar ovog uvjetovanog bloka upišite sljedeće naredbe:
  - `PORTB &= ~(1 << PB1);` - isključivanje crvene LED diode,
  - `PORTB |= (1 << PB2);` - uključivanje žute LED diode,
  - `PORTB |= (1 << PB3);` - uključivanje zelene LED diode.
- otvorite uvjetovani blok `if (T >= 25.0) { }`. Ovim blokom provjerava se da li je temperatura veća i jednaka 25 °C i uključuju se sve LED diode. Unutar ovog uvjetovanog bloka upišite sljedeće naredbe:
  - `PORTB |= (1 << PB1);` - uključivanje crvene LED diode,
  - `PORTB |= (1 << PB2);` - uključivanje žute LED diode,
  - `PORTB |= (1 << PB3);` - uključivanje zelene LED diode.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba73.cpp` treba biti ista kao programski kod 7.6.

Programski kod 7.6: Početni sadržaj datoteke `vjezba73.cpp` - program kojim se ispisuje vrijednost temperature okoline senzora LM35

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    uint16_t adc0; // digitalna riječ ADC0
    uint16_t adc0Old; // digitalna riječ ADC0
    float T; // temperatura u okolini senzora LM35
    bool ispisNaLcd = true; // omogućavanje ispisa
    int brojac_vremena = 0; // brojač za sekunde
```

```
    T = readTempLM35(LM35); // čitaj temperaturu na LM35
    while (1) { // beskonačna petlja
        // ako je brojač = 100, prošla je jedna sekunda
        if (++brojac_vremena == 100) {
            adc0 = adcRead(LM35); // AD pretvorba na pinu ADC0
            // pretvori rezultat ADC u temperaturu
            T = adcConvertToTempLM35(adc0);
            brojac_vremena = 0; // vrati brojač na 0 za novu sekundu
        }
        // ako se promijenila vrijednost AD pretvorbe i ako je prošla sekunda
        if ((adc0 != adc0Old) && (brojac_vremena == 0)) {
            ispisNaLcd = true; // omogući ispis na LCD
            adc0Old = adc0; // osvježi staru vrijednost adc0
        }
        // uvjetni ispis na LCD displej i izmjena stanja LED dioda
        if (ispisNaLcd) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("T = %.2f%c", T, 223);
            if (T < 22.0) {
                PORTB &= ~(1 << PB1); // LED crvena off
                PORTB &= ~(1 << PB2); // LED žuta off
                PORTB |= (1 << PB3); // LED zelena on
            }
            if (T >= 22.0 && T < 25.0) {
                PORTB &= ~(1 << PB1); // LED crvena off
                PORTB |= (1 << PB2); // LED žuta on
                PORTB |= (1 << PB3); // LED zelena on
            }
            if (T >= 25.0) {
                PORTB |= (1 << PB1); // LED crvena on
                PORTB |= (1 << PB2); // LED žuta on
                PORTB |= (1 << PB3); // LED zelena on
            }
            ispisNaLcd = false; //onemogući ispis na LCD
        }
        _delay_ms(10);
    }
    return 0;
}
```

Prevedite datoteku `vjezba73.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se vrijednost temperature okoline senzora LM35 pri promjeni temperature, ali ne češće od jednom u sekundi.

Zatvorite datoteku `vjezba73.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 8

Prekidi mikroupravljača

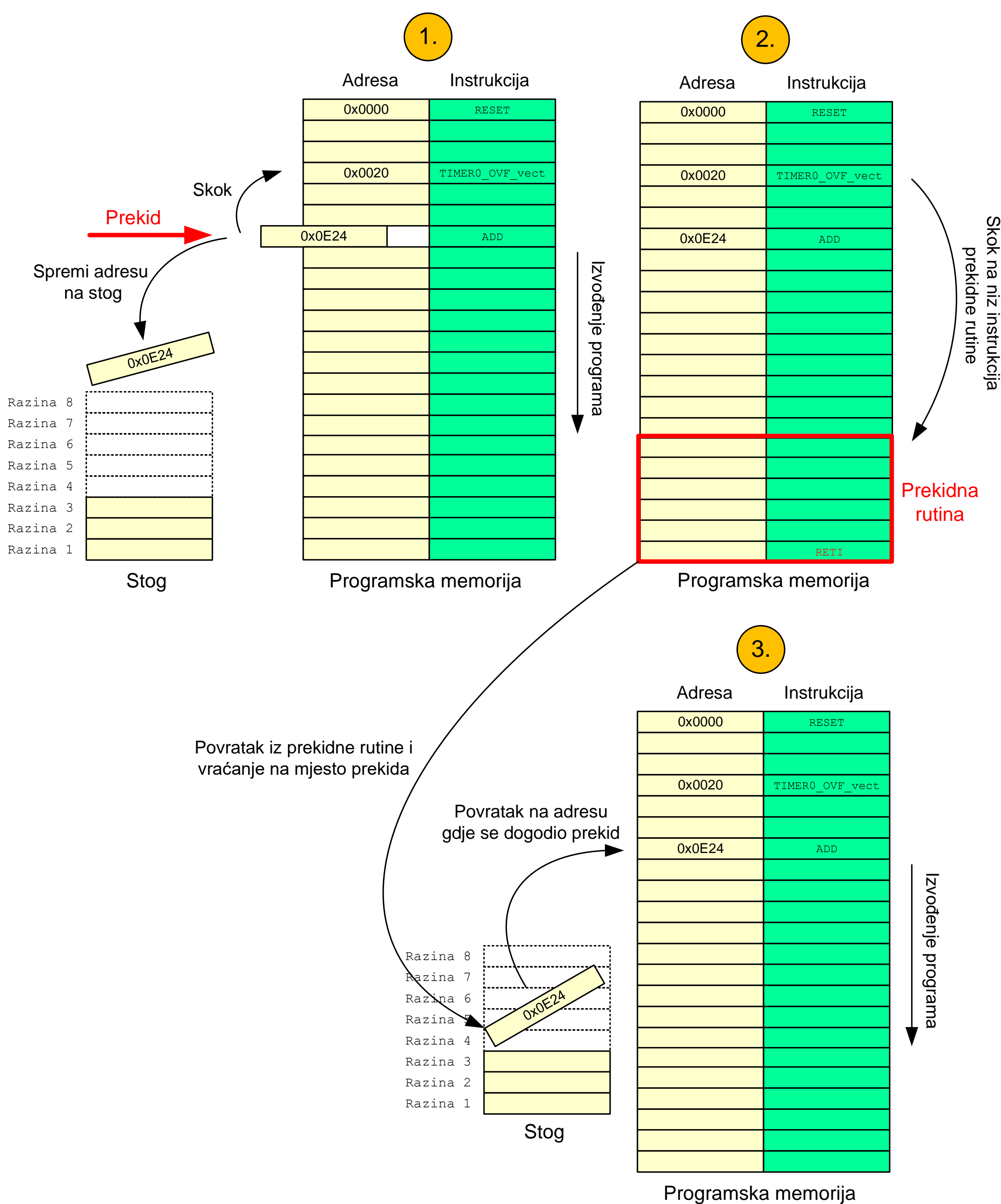
Mikroupravljači moraju moći odgovoriti na događaje koji se dešavaju u sustavima kojima upravljaju. Događaji u sustavu označavaju stanje promjene i zahtijevaju trenutnu reakciju mikroupravljača na njih. Vrste događaja mogu biti od jednostavnih, kao što je uvećavanje brojača predmeta koji prolaze pored fotoelektričnog senzora na proizvodnoj liniji, pa do složenih, kao što je isključivanje cijele proizvodne linije ako je netko ušao u radni prostor strojeva koji su opasni po život. U oba navedena slučajeva, reakcija mikroupravljača bi trebala biti trenutna bez obzira što se trenutno izvodi u glavnom programu mikroupravljača.

U svrhu obrade događaja na koje se mora odgovoriti u trenutku kada se oni dogode, mikroupravljači nude mehanizam koji se naziva prekid (engl. *Interrupt*). Kada nastupi događaj koji izaziva prekid, mikroupravljač uobičajeno završi instrukciju koju trenutno izvršava, a nakon toga se izvršavanje programa prebacuje na specifični prekidni zadatak. Prekidni zadaci organizirani su u takozvane prekidne servisne rutine (engl. *Interrupt Service Routine*) (ISR). Različiti izvori prekida (tajmeri, serijska komunikacija, brid signala itd.) imaju vlastite prekidne rutine.

Obradu prekidne rutine prikazat ćemo pomoću slike 8.1. Pretpostavimo da se prekid dogodio neposredno prije nego što je mikroupravljač trebao obraditi instrukciju na adresi `0x0E24`. U tom trenutku, kada se dogodio prekid, na stog se sprema adresa `0x0E24` iz programske memorije (1. korak na slici 8.1). Razlog tomu je taj što će program morati nastaviti obrađivati instrukcije na mjestu gdje je prekinut. Nakon što se adresa `0x0E24` spremi na stog, program skače na adresu vektora prekidne rutine koja je izazvala prekid. U slučaju na slici 8.1 prekid je izazvao tajmer nakon isteka zadanog vremena (prekidni vektor `TIMER0_OVF_vect`). Adresa vektora prekidne rutine sadrži adresu tijela prekidne rutine u programskoj memoriji na koju se preusmjerava daljnje izvođenje programa (2. korak na slici 8.1). Tijelo prekidne rutine sadrži niz instrukcija koje će mikroupravljač izvršiti. Na kraju prekidne rutine nalazi se instrukcija `RETI` (engl. *Interrup Return*) koja pokreće postupak vraćanja izvođenja programa na adresu na kojoj se dogodio prekid (3. korak na slici 8.1). Pri povratku iz prekidne rutine sa stoga se kao sljedeća adresa u programskoj memoriji uzima adresa `0x0E24`. To je upravo adresa koja nije izvedena zbog dešavanja prekida. Program zapisan u programskoj memoriji nastavlja se izvoditi instrukciju po instrukciju do pojave novog prekida.

Kako smo već spomenuli, izvor prekida može biti izazvan odbrojavanjem tajmera, dolaskom novog znaka serijskom komunikacijom, rastući ili padajući brid signala na pinovima za vanjski prekid i drugi. Neki od izvora prekida mikroupravljača ATmega328P prikazani su u tablici 8.1. RESET mikroupravljača je prekid najvećeg prioriteta i njegova uloga je pokrenuti izvođenje programa ispočetka. Ovaj prekid nema prekidnu rutinu, ni prekidni vektor. Svi ostali izvori prekida dohvaćaju se pomoću prekidnih vektora koji su za mikroupravljač ATmega328P prikazani

tablicom 11-1 u literaturi [2].



Slika 8.1: Obrada prekidne rutine

Korištenje prekidnih mehanizama u programskom kodu zahtijeva konfiguraciju prekida, omogućavanje prekida (globalno i konkretnog izvora prekida) te pisanje prekidne servisne rutine (ISR). Konfiguracijom prekida možemo odrediti na koji način će se aktivirati prekid. Na primjer, kada govorimo o prekidu vanjskih pinova, tada prekid može biti izazvan na rastući brid signala, na padajući brid signala i na oba brida signala. Da bismo mogli koristiti prekide, potrebno ih je globalno omogućiti makronaredbom `sei()`. Ako želimo onemogućiti prekide, potrebno je pozvati makronaredbu `cli()`. Nadalje, da bismo mogli koristiti prekide u programski kod potrebno

je uključiti zaglavlje avr/interrupt.h naredbom #include <avr/interrupt.h>. Osim što je globalno potrebno omogućiti prekide, svaki izvor prekida mora se i pojedinačno omogućiti.

Prekidna servisna rutina definirana je blokom ISR(izvor_vect){}. Unutar ovog bloka naredaba piše se tijelo prekidne rutine, odnosno niz naredbi koje je potrebno izvesti kada se dogodi prekid. Primjer prekidne servisne rutine prikazan je programskim kodom 8.1. Argument prekidne rutine izvor_vect je prekidni vektor. Neki od prekidnih vektora prikazani su u tablici 8.1, dok ostale možete pronaći u zaglavlju avr/io.h ili preciznije u zaglavlju avr/iom328p.h[1].

Programski kod 8.1: Primjer prekidne servisne rutine

```
// osnovna zaglavlja za prekide
#include <avr/io.h>
#include <avr/interrupt.h>
// prekidna rutina za proizvoljni izvor prekida
ISR(izvor_vect) {
    // kod koji se izvršava nakon prekida
}
```

Tablica 8.1: Popis nekoliko prekidnih vektora mikorupravljača ATmega328P [2]

Vektor br.	Izvor prekida	Konstanta prekidnog vektora	Opis prekida
1	RESET	-	Prekid koji se dešava kod bilo kojeg izvora RESETa: External pin, Power-on Reset, Brown-out Reset, Watchdog Reset, JTAG AVR Reset
2	INT0	INT0_vect	Prekid koji se javlja na rastući ili padajući brid na pinu mikroupravljača INT0 (PD2)
3	INT1	INT1_vect	Prekid koji se javlja na rastući ili padajući brid na pinu mikroupravljača INT1 (PD3)
10	TIMER2 OVF	TIMER2_OVF_vect	Prekid koji se javlja pri isteku vremena tajmera broj 2. Istek vremena se dešava kada se dogodi preljev u registru tajmera 2.
14	TIMER1 OVF	TIMER1_OVF_vect	Prekid koji se javlja pri isteku vremena tajmera broj 1. Istek vremena se dešava kada se dogodi preljev u registru tajmera 1.
17	TIMER0 OVF	TIMER0_OVF_vect	Prekid koji se javlja pri isteku vremena tajmera broj 0. Istek vremena se dešava kada se dogodi preljev u registru tajmera 0.
19	USART, RX	USART_RX_vect	Prekid koji se javlja kada je primljen znak putem serijske komunikacije na mikroupravljač.

Primjer konfiguracije prekida i prekidne servisne rutine za vanjski prekid INT0 prikazan je programskim kodom 8.2. Ovdje nećemo opisivati detaljno ovaj programski kod. Ključno je da

[1]Zaglavljem avr/io.h uključuje se zaglavlje mikroupravljača kojeg definirate u projektu. Za mikroupravljač ATmega328P uključuje se zaglavlje avr/iom328p.h.

uočimo zaglavlja koja je potrebno uključiti, globalno omogućenje prekida, omogućenje prekida za izvor INT0, konfiguriranje načina rada prekida i prekidnu servisnu rutinu s pripadajućim prekidnim vektorom.

Programski kod 8.2: Primjer konfiguracije prekida i prekidne rutine za vanjski prekid INT0

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include <avr/interrupt.h>
// globalna varijabla za brojanje impulsa na INT0
volatile int brojac_int0 = 0;
// globalna varijabla za omogućenje ispisa na LCD
volatile bool lcd_update = true;
// prekidna rutina za vanjski prekid INT0
ISR(INT0_vect) {
    brojac_int0++; // povečaj za 1
    lcd_update = true; //omogući ispis na LCD
}
void init() {
    lcdInit();
    // konfiguracija prekida INT0
    EIMSK |= (1 << INT0);  // omogući prekid INT0
    // padajući brid generira prekid na INT0
    EICRA |= (1 << ISC01) | (0 << ISC00);
    sei(); // globalno omogućenje prekida
}
int main(void) {

    init(); // inicijalizacija mikroupravljača
    while (1) { // beskonačna petlja
        if(lcd_update) {
            lcdClrScr();
            lcdprintf("INT0: %d", brojac_int0);
            lcd_update = false;
        }
    }
    return 0;
}
```

U programskom kodu 8.2, kod deklaracije globalnih varijabli, koristili smo ključnu riječ `volatile`. Prije nego što objasnimo razlog uvođenja ključne riječi `volatile`, objasnit ćemo kako *gcc* prevoditelj radi optimizaciju pri prevođenju programskog koda u strojni. Prevoditelji su izuzetno napredni, a za cilj imaju uštedu programske memorije. Pretpostavimo za trenutak da smo globalnu varijablu `lcd_update` deklarirali bez korištenja ključne riječi `volatile`. Procjena optimizacije jest da će se programski kod uvjetovan varijablom `lcd_update` izvesti samo jednom u `while` petlji jer prevoditelj vidi kako je ova varijabla početno u vrijednosti `true`, a nijedna funkcija koja se poziva iz glavne funkcije ne mijenja tu varijablu. S pozicije prevoditelja, ova varijabla je literal. Iz tog će razloga prevoditelj obrisati cijelu `while` petlju i ostaviti samo naredbe za ispis na LCD displej. Rezultat optimizacije bio bi završetak programa nakon prvog ispisa na LCD displej jer bi funkcija `main()` s izvođenjem došla do samoga kraja. Iako na prvu izgleda kako je prevoditelj napravio propust, on je ipak optimizaciju odradio potpuno ispravno. Prevoditelj programskog koda u jeziku C ili C++ ne razumije da će prekidnu rutinu pozvati neki vanjski sklop i promijeniti varijablu `lcd_update` te se stoga ovakvi "propusti" prevoditelja u slučaju optimizacije pojavljuju samo onda kada se koriste prekidne rutine. Ključna riječ `volatile` ima za cilj spriječiti primjenu optimizacije programskog koda nad objektima koji se mogu promijeniti na načine koje prevoditelj ne može utvrditi. Ovu ključnu riječ primijenit ćemo uvijek kod globalnih varijabli koje se mijenjaju u prekidnim rutinama.

U sljedećim poglavljima bavit ćemo se izvorima prekida i njima svojstvenim prekidnim vektorima.

Poglavlje 9

Tajmeri i brojači

Jedan od najvažnijih razloga zašto se mikroupravljači koriste u ugradbenim sustavima jest sposobnost mikroupravljača za obavljanje vremenskih zadataka. U jednostavnijim aplikacijama potrebno je uključivati i isključivati vanjske uređaje u zadanim vremenskim intervalima koji mogu biti ili ne moraju biti periodični. Složeniji vremenski zadaci koje mikroupravljač obavlja su sinkronizacija događaja u mikroupravljaču, generiranje periodičnih valnih oblika, uzimanje mjernih uzoraka u točnim vremenskim trenucima, mjerenje frekvencije i perioda te obrada PID algoritma za upravljanje. Vremenske zadatke moguće je obavljati sa sklopovljem koje se naziva tajmeri ili vremenski brojači (engl. *timer*). Sljedeći važan razlog korištenja mikroupravljača u ugradbenim sustavima jest primjena u zadacima koji uključuju brojanje predmeta na proizvodnoj traci, brojanje impulsa enkodera te druge vrste brojanja. Brojanje se provodi sa sklopovljem koje se naziva brojači (engl. *counter*). Brojači i vremenski brojači dio su istog sklopovlja mikroupravljača.

Vremenski brojač ili tajmer je brojač kojim se broje impulsi poznate frekvencije. Tajmeri su izuzetno važan dio mikroupravljača, a većina ih ima jedan tajmer ili više rezolucije 8 i/ili 16 bitova [4]. Frekvencija impulsa koje tajmer broji može se mijenjati, odnosno dijeliti u odnosu na frekvenciju radnog takta mikroupravljača s djeliteljima frekvencije koji su u pravilu potencije broja 2.

Tajmer i brojač su na hardverskoj razini isti sklop koji se može konfigurirati ili kao tajmer ili kao brojač. Ako je izvor impulsa koji se broje vezan uz radni takt mikroupravljača, onda je ovaj sklop konfiguriran kao tajmer, a ako je izvor impulsa vanjski uređaj (senzor prisutnosti, enkoder, kristal kvarca), tada je ovaj sklop konfiguriran kao brojač.

Rezolucija tajmera i brojača ovisi o širini registra koji se koristi za brojanje impulsa. Ako je rezolucija tajmera i brojača n, tada se vrijednosti registra u kojem se broje impulsi kreću u rasponu [0, 2^n - 1]. Na primjer, ako je rezolucija tajmera i brojača 16, tada se vrijednosti njihova registra kreću u rasponu od [0, 65535]. Na svaki impuls s izvora impulsa tajmera i brojača, vrijednost se u registru povećava ili smanjuje za 1 što ovisi o konfiguraciji sklopovlja.

Pri korištenju tajmera i brojača potrebno je voditi računa o rezoluciji. Na primjer, ako vrijednost registra 8-bitnog tajmera iznosi 255 (0xFF) i njegova se vrijednost povećava za 1 sa svakim impulsom, tada će nakon sljedećeg impulsa njegova vrijednost biti 0 (0x00). U prijelazu stanja u registru iz 255 u 0 događa se preljev registra (engl. *overflow*) jer broj 256 (0x1FF) ne stane u registar širine 8 bitova. Tajmeri i brojači u mikroupravljačima najčešće generiraju prekid u trenutku kada se dogodi preljev. Činjenica da se dogodio preljev koji generira prekid omogućuje nam da obrađujemo vremenske zadatke u jednakim vremenskim intervalima. Prekidne mehanizme obradili smo u prethodnom poglavlju.

Mjerenje vremena pomoću tajmera moguće je brojanjem impulsa dobivenih na temelju radnog

takta poznate frekvencije (npr. 16 MHz). Radni je takt moguće podijeliti cijelim brojem m, (m u ovisnosti o mikroupravljaču može biti 1, 2, 4, 8, 32, 64, 128, 256, 1024). Ako je radni takt poznate frekvencije F_CPU podijeljen brojem m, tada će tajmer brojati svaki m-ti impuls, a frekvencija tih impulsa će biti F_CPU/m. Sklop za dijeljenje frekvencije radnog takta zove se djelitelj frekvencije radnog takta (engl. *prescaler*).

Mikroupravljač ATmega328P ima tri tajmera i brojača opće namjene:

- *Timer/Counter0* - tajmer i brojač rezolucije 8 bitova s rasponom brojanja od [0, 255],
- *Timer/Counter1* - tajmer i brojač rezolucije 16 bitova s rasponom brojanja od [0, 65535],
- *Timer/Counter2* - tajmer i brojač rezolucije 8 bitova s rasponom brojanja od [0, 255].

Rad tajmera i brojača prikazat ćemo na sklopu *Timer/Counter0*. U nastavku teksta tajmer i brojač s indeksom 0 zvat ćemo izvornim imenom *Timer/Counter0* iz literature [2]. Na slici 9.1 prikazan je blokovski dijagram za sklop *Timer/Counter0* rezolucije 8 bitova.

Sklop *Timer/Counter0* ima pet 8-bitnih registara sa sljedećim funkcijama [1]:

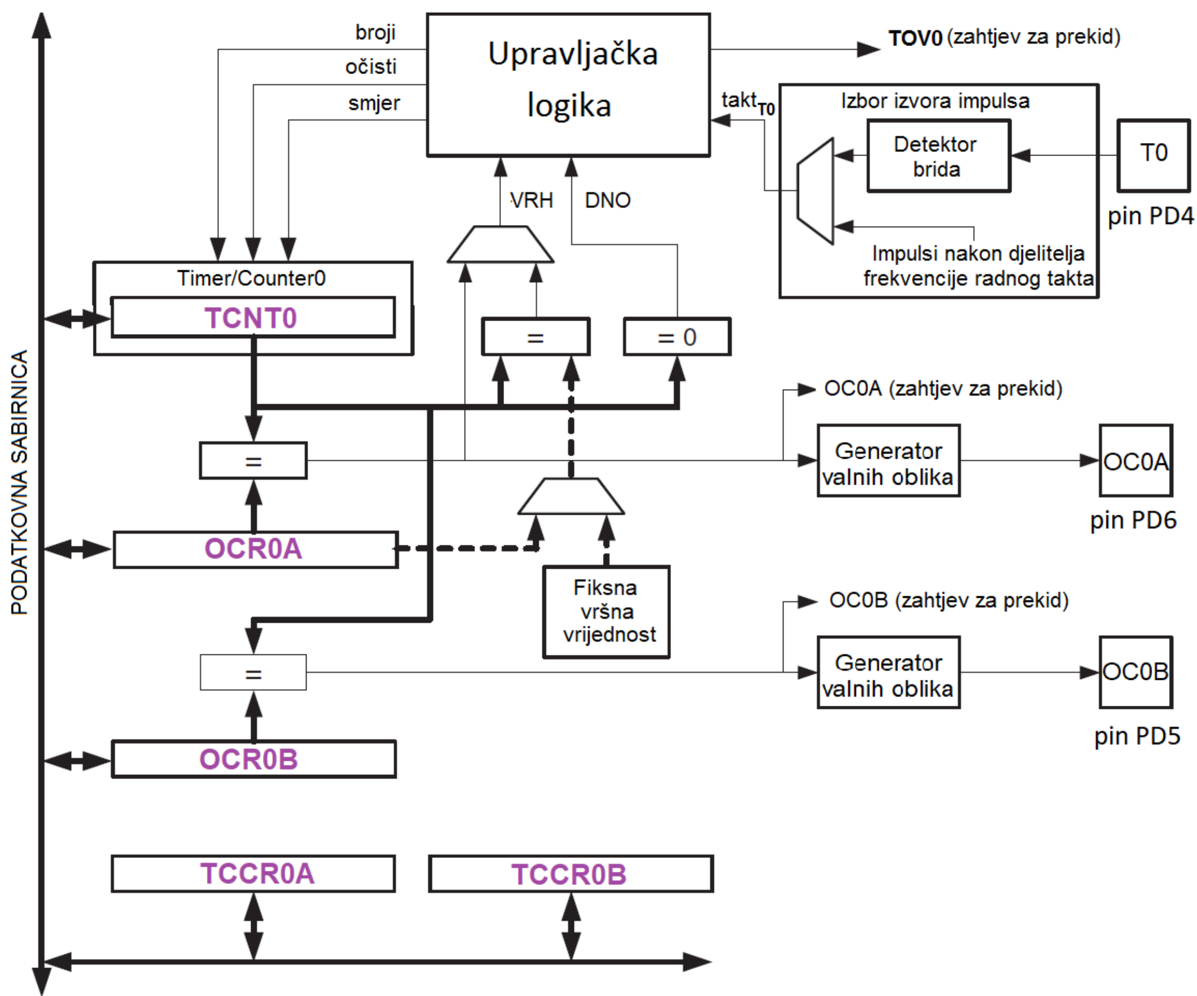
- `TCNT0` - (engl. *Timer/Counter Register*), registar širine 8 bitova u kojem se vrijednost uvećava ili smanjuje za 1. Raspon vrijednosti u registru jest od [0, 255]. Moguće ga je programski čitati i u njega zapisivati vrijednosti putem podatkovne sabirnice.
- `OCR0A`, `OCR0B` - (engl. *Output Compare Register*), registri širine 8 bitova čije se vrijednosti mogu uspoređivati s registrom `TCNT0`. Raspon vrijednosti u registru jest od [0, 255]. Moguće ga je programski čitati i u njega zapisivati vrijednosti putem podatkovne sabirnice.
- `TCCR0A`, `TCCR0B` - (engl. *Timer/Counter Control Register*), registri kojim se konfigurira rad sklopa *Timer/Counter0*.

Na blokovskom dijagramu sklopa *Timer/Counter0* sa slike 9.1 prikazani su sljedeći signali:

- `broji` - uvećaj ili smanji vrijednost registra `TCNT0` za 1 (ovisi o upravljačkoj logici),
- `očisti` - postavi vrijednost registra `TCNT0` na 0,
- `smjer` - uvećavaj ili smanjuj za 1 (korisnik ne može određivati smjer),
- $\texttt{takt}_{\texttt{T0}}$ - izvor impulsa na temelju kojih se vrijednost registra `TCNT0` uvećava ili smanjuje za 1,
- `DNO` (engl. *bottom*) - signalizira da je vrijednost registra `TCNT0` 0x00, a koristi se za generiranje prekida kada se dogodi preljev,
- `VRH` (engl. *top*) - signalizira da je vrijednost registra `TCNT0` postigla maksimalnu vrijednost koja može biti 255 ili vrijednost koja se nalazi u registru `OCR0A`.

Sklop *Timer/Counter0* generira dvije vrste prekida:

- **TOV0** - prekid koji se generira kada registar `TCNT0` prelazi iz vrijednosti 255 u 0. Ovaj prekid poziva prekidni vektor `TIMER0_OVF_vect`.
- **OC0A, OC0B** - prekidi koji se generiraju kada je vrijednost registra `TCNT0` jednaka vrijednosti registara `OCR0A`, `OCR0B`. Ovi prekidi pozivaju prekidne vektore `TIMER0_COMPA_vect` i `TIMER0_COMPB_vect`.



Slika 9.1: Blokovski dijagram za sklop *Timer/Counter0* rezolucije 8 bitova [1]

Upravljački registri kojima se konfigurira rad sklopa *Timer/Counter0* prikazani su na slikama 9.2 i 9.3. Svaki bit u registrima `TCCR0A` i `TCCR0B` ima svoju funkciju i određuje rad sklopa *Timer/Counter0*.

Bit	7	6	5	4	3	2	1	0
`TCCR0A`	`COM0A1`	`COM0A0`	`COM0B1`	`COM0B0`	——	——	`WGM01`	`WGM00`

Slika 9.2: *Timer/Counter0* upravljački registar `TCCR0A` [1]

Bit	7	6	5	4	3	2	1	0
Registar `TCCR0B`	`FOC0A`	`FOC0B`	——	——	`WGM02`	`CS02`	`CS01`	`CS00`

Slika 9.3: *Timer/Counter0* upravljački registar `TCCR0B` [1]

Na slici 9.1 prikazan je blok **`Izbor izvora impulsa`**. Izvor impulsa na temelju kojeg sklop *Timer/Counter0* broji mogu biti impulsi djelitelja frekvencije ili impulsi dovedeni na pin PD4 (T0). Primijetite da pin PD4 ima alternativnu namjenu uz to što može biti digitalni ulaz i izlaz. Odabir izvora impulsa provodi se pomoću registra `TCCR0B` tako da se konfiguriraju bitovi `CS02`, `CS01` i `CS00` prema tablici 9.1. Prvih šest kombinacija bitova `CS02`, `CS01` i `CS00` u tablici 9.1 konfiguriraju

sklop *Timer/Counter0* kao tajmer, a zadnje dvije kombinacije konfiguriraju sklop *Timer/Counter0* kao brojač. Kako smo već naveli u prethodnim poglavljima, radni takt na razini mikroupravljača definiran je *Fuse* bitovima, a na razini programa konstantom `F_CPU`.

Tablica 9.1: Izbor izvora impulsa sklopa *Timer/Counter0* [2]

`CS02`	`CS01`	`CS00`	**Izvor impulsa**
0	0	0	Nema impulsa, sklop *Timer/Counter0* je zaustavljen
0	0	1	`F_CPU`
0	1	0	`F_CPU`/8
0	1	1	`F_CPU`/64
1	0	0	`F_CPU`/256
1	0	1	`F_CPU`/1024
1	1	0	Padajući brid signala pina T0 uvećava registar TCNT0 za 1
1	1	1	Rastući brid signala pina T0 uvećava registar TCNT0 za 1

Četiri su osnovna načina rada tajmera:

- normalan način rada,
- *CTC* (engl. *Clear Timer on Compare Match*) način rada,
- *Fast* PWM (engl. *Pulse Width Modulation*) način rada,
- *Phase Correct* PWM način rada.

Odabir načina rada tajmera provodi se konfiguracijom bitova `WGM02`, `WGM01` i `WGM00` u registrima `TCCR0A` i `TCCR0B` prema tablici 9.2. Ovu tablicu koristit ćemo za konfiguriranje načina rada tajmera sklopa *Timer/Counter0*. U ovoj vježbi ćemo sklop *Timer/Counter0* koristiti kao tajmer u normalnom načinu rada te kao brojač vanjskih impulsa. Registri `TCCR0A` i `TCCR0B` posjeduju bitove za konfiguriranje načina generiranja valnih oblika u PWM načinu rada, no te bitove ćemo opisati u sljedećem poglavlju.

Tablica 9.2: Odabir načina rada tajmera [2]

`WGM02`	`WGM01`	`WGM00`	**Način rada**	**Maksimalna vrijednost brojača**
0	0	0	Normalni način rada	0xFF
0	0	1	*Phase Correct* način rada	0xFF
0	1	0	*CTC* način rada	`OCR0A`
0	1	1	*Fast* PWM način rada	0xFF
1	0	0	Rezervirano	—
1	0	1	*Phase Correct* način rada	`OCR0A`
1	1	0	Rezervirano	—
1	1	1	*Fast* PWM način rada	`OCR0A`

9.1 Vježbe - tajmeri i brojači u normalnom načinu rada

Normalan način rada tajmera i brojača najjednostavniji je način rada. Mikroupravljač ATmega328P ima tri tajmera i brojača: *Timer/Counter0*, *Timer/Counter1* i *Timer/Counter2*. U normalnom načinu rada, vrijednost registra `TCNTx` ($x = 0, 1, 2$) povećava se za 1 neovisno o tome da li je izvor impulsa vanjski uređaj ili je povezan s radnim taktom mikroupravljača. Registri tajmera i brojača `TCNT0` i `TCNT2` su 8-bitni, dok je registar `TCNT1` 16-bitan. Maksimalna vrijednost koju registri `TCNT0` i `TCNT2` mogu doseći jest 255, dok maksimalna vrijednost koju registar `TCNT1` može doseći jest 65535. Što se događa s navedenim registrima kada je njihov iznos maksimalan i kada se na ulazu tajmera ili brojača pojavi novi impuls kojeg treba brojiti? Događa se preljev registra nakon čega će njegova vrijednost biti 0. Pri prijelazu iz maksimalne vrijednosti registra u vrijednost 0 (kada se desi preljev), svi tajmeri i brojači generiraju prekid.

Kako smo već spomenuli u prošlom poglavlju, prekidne se rutine pozivaju makronaredbom `ISR(vector)`. Korištenje prekidnih rutina i općenito prekidnih mehanizama zahtjeva uključenje zaglavlja `#include <avr/interrupt.h>`. Omogućenje prekida na razini mikroupravljača (globalni prekidi) provodi se pozivom makronaredbe `sei()`. Autori su pripremili zaglavlje `#include "Interrupt/interrupt.h"` unutar kojeg se nalazi funkcija za globalno omogućenje prekida `interruptEnable()`. Funkcija kojom se onemogućuje globalni prekid zove se `interruptDisable()`.

Osim što je potrebno omogućiti prekid na globalnoj razini, svaki izvor prekida potrebno je zasebno omogućiti. Tako se generiranje prekida pri preljevu registra `TCNT0` omogućuje u registru `TIMSK0` tako da se bit na poziciji 0 postavi u 1. Naziv tog bita jest `TOIE0`. Na isti način se prekidi omogućuju za ostale tajmere i brojače mikroupravljača ATmega328P. Primjer omogućenja prekida za sve tajmera i brojače mikroupravljača ATmega328P te globalnog omogućenja prekida prikazan je programskim kodom 9.1.

Programski kod 9.1: Omogućenje prekida za tajmere i brojače mikroupravljača ATmega328P

```
interruptEnable(); //globalno omogućen prekid
TIMSK0 |= (1 << TOIE0); // omogućenje prekida za timer0
TIMSK1 |= (1 << TOIE1); // omogućenje prekida za timer1
TIMSK2 |= (1 << TOIE2); // omogućenje prekida za timer2
```

Normalan način rada tajmera konfigurira se registrima `TCCRxA` i `TCCRxB` pomoću bitova `WGMx0`, `WGMx1` i `WGMx2` ($x = 0, 1, 2$) te dodatno bitom `WGM13` kod sklopa *Timer/Counter1*. Konfiguracija normalnog načina rada tajmera mikroupravljača ATmega328P prikazana je programskim kodom 9.2. Provedena je pomoću tablice 9.2 za sklop *Timer/Counter0* te pomoću tablica 15-4 i 17-8 u literaturi [2] za sklopove *Timer/Counter1* i *Timer/Counter2*. Općenito, za konfiguriranje sklopovlja mikroupravljača ključno je dobro poznavati specifikacije korištenog mikroupravljača (engl. *datasheet*).

Programski kod 9.2: Konfiguracija normalnog načina rada tajmera mikroupravljača ATmega328P

```
// normalna način rada za timer0
TCCR0A |= (0 << WGM01) | (0 << WGM00);
TCCR0B |= (0 << WGM02);
// normalna način rada za timer1
TCCR1A |= (0 << WGM11) | (0 << WGM10);
TCCR1B |= (0 << WGM13) | (0 << WGM12);
// normalna način rada za timer2
TCCR2A |= (0 << WGM21) | (0 << WGM20);
TCCR2B |= (0 << WGM22);
```

Nakon omogućenja prekida pri preljevu pojedinog tajmera te odabira normalnog načina rada, najvažniji dio u konfiguraciji tajmera jest odrediti vrijeme između poziva dvaju prekida tajmera. To vrijeme nam omogućuje da vremenske zadatke provodimo u strogo zajamčenom vremenu (na primjer svakih 100 ms).

Vrijeme između dvaju prekida može se izračunati pomoću sljedeće relacije:

$$t_{Tx} = \frac{PRESCALERx}{F_CPU} \cdot (2^{n_x} - TCNTx_0) \tag{9.1}$$

gdje je:

- t_{Tx} - vrijeme između dvaju prekida tajmera x ($x = 0, 1, 2$) u sekundama,
- $PRESCALERx$ - djelitelj frekvencije radnog takta tajmera x,
- F_CPU - frekvencija radnog takta mikroupravljača,
- 2^{n_x} - maksimalni broj stanja registra tajmera x gdje broj n_x predstavlja rezoluciju tajmera x (za *Timer/Counter0* i *Timer/Counter2* $n_0 = n_2 = 8$, a za *Timer/Counter1* $n_1 = 16$),
- $TCNTx_0$ - početna vrijednost registra `TCNTx`. Razlika 2^{n_x} - $TCNTx_0$ odgovara broju impulsa koji na frekvenciji radnog takta uz zadani djelitelj frekvencije radnog takta traje t_{Tx} vremena.

Frekvencija radnog takta na našem razvojnom okruženju jest 16 MHz. Pretpostavimo da je djelitelj frekvencije namješten tako da propušta radni takt na brojač tajmera ($PRESCALERx = 1$). U ovom slučaju, registar `TCNTx` bi svoju vrijednost uvećao za 1 ukupno 16 milijuna puta u jednoj sekundi. Pri tome bi sklopovi *Timer/Counter0* i *Timer/Counter2* generirali 16000000/256 = 62500 prekida preljevom registra `TCNT0`, odnosno registra `TCNT2` u jednoj sekundi. Sklop *Timer/Counter1* bi generirao 16000000/65536 = 244[1] prekida preljevom registra `TCNT1` u jednoj sekundi. Kada bi se koristio puni opseg brojanja u registrima `TCNT0` i `TCNT2` ($TCNT0_0 = 0$ i $TCNT2_0 = 0$), vrijeme između poziva dvaju prekida bilo bi 16 μs. Puni opseg brojanja u registru `TCNT1` ($TCNT1_0 = 0$) osiguralo bi vrijeme poziva između dvaju prekida u iznosu od 4.096 ms. No, što ako želimo povećati vrijeme između dva prekida? Složit ćemo se da je 16 μs vrlo kratko vrijeme. Tajmeri imaju djelitelje frekvencije radnog takta koji mogu smanjiti frekvenciju brojanja, a time i povećati vrijeme između dva prekida. Djelitelj frekvencije radnog takta konfigurira se u registru `TCCRxB` pomoću bitova `CSx0`, `CSx1` i `CSx2` ($x = 0, 1, 2$). Odabir djelitelja frekvencije radnog takta za tajmere mikroupravljača ATmega328P prikazan je programskim kodom 9.3. Djelitelji frekvencije radnog takta odabrani su pomoću tablice 9.1 za sklop *Timer/Counter0* te pomoću tablica 15-5 i 17-9 u literaturi [2] za sklopove *Timer/Counter1* i *Timer/Counter2*. Primijetite da sklop *Timer/Counter2* nema mogućnost brojanja impulsa iz vanjskog izvora.

Programski kod 9.3: Odabir djelitalja frekvencije radnog takta za tajmere mikroupravljača ATmega328P

```
// timer0 - F_CPU/256
TCCR0B |= ((1 << CS02) | (0 << CS01) | (0 << CS00));
// timer1 - F_CPU/64
TCCR1B |= ((0 << CS12) | (1 << CS11) | (1 << CS10));
// timer2 - F_CPU/128
TCCR2B |= ((1 << CS22) | (0 << CS21) | (1 << CS20));
```

U nastavku ćemo napraviti analizu postavljanja djelitelja frekvencije za sklop *Timer/Counter1*. Djelitelj frekvencije radnog takta postavljen je na vrijednost 64. To znači da će

[1] Cjelobrojno dijeljenje. Pravi rezultat dijeljenja jest 244.141, no prekid se može pozvati samo cijeli broj puta.

se vrijednost u registru TCNT1 povećati za 1 ukupno 16000000/64 = 250000 puta u jednoj sekundi. S ovim postavkama bi sklop *Timer/Counter1* prosječno generirao prekide 16000000/64/65536 = 3.8147 puta u jednoj sekundi. Puni opseg brojanja u registru TCNT1 ($TCNT1_0 = 0$) osiguralo bi vrijeme poziva između dvaju prekida u iznosu od 262.144 ms. Možemo primijetiti da smo uspješno povećali vrijeme između dvaju prekida tajmera. No, što ako želimo ugoditi vrijeme na 100 ms? Primijetite da puni opseg brojanja registra TCNT1 daje samo nekoliko diskretnih vrijednosti vremena između prekida koja se mogu postići. Da bismo ugodili vrijeme između dvaju prekida na zadano (npr. 100 ms), potrebno je pri svakom pozivu prekida postaviti početnu vrijednost registra TCNT1. Ako je početna vrijednost registra TCNT1 jednaka 50000, tada će sklop *Timer/Counter1* izazvati prekid nakon 65536 - 50000 = 15536 impulsa djelitelja frekvencije jer registar ne prolazi kroz svoj puni opseg. S ovim postavkama bi sklop *Timer/Counter1* prosječno generirao prekide 16000000/64/15536 = 16.09 puta u jednoj sekundi, odnosno vrijeme između dva prekida iznosilo bi 62.144 ms.

Vrijeme između dvaju prekida pojedinačno za svaki tajmer može se izračunati pomoću sljedećih relacija:

$$t_{T0} = \frac{PRESCALER0}{F_CPU} \cdot (256 - TCNT0_0) \tag{9.2}$$

$$t_{T1} = \frac{PRESCALER1}{F_CPU} \cdot (65536 - TCNT1_0) \tag{9.3}$$

$$t_{T2} = \frac{PRESCALER2}{F_CPU} \cdot (256 - TCNT2_0) \tag{9.4}$$

Kako smo u prethodnom izlaganju vidjeli, da bismo ostvarili zadano vrijeme između dvaju prekida potrebno je odabrati djelitelj frekvencije te izračunati početnu vrijednost registra tajmera. Frekvencija radnog takta mikroupravljača je u pravilu konstanta i ona se ne mijenja tijekom programiranja. Početnu vrijednost registara TCNTx za zadano vrijeme t_{Tx} može se dobiti temeljem relacija (9.2), (9.3) i (9.4) na sljedeći način:

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F_CPU}{PRESCALER0}. \tag{9.5}$$

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_CPU}{PRESCALER1}. \tag{9.6}$$

$$TCNT2_0 = 256 - t_{T2} \cdot \frac{F_CPU}{PRESCALER2}. \tag{9.7}$$

U relacijama (9.5) - (9.7) potrebno je pronaći najmanji djelitelj frekvencije radnog takta za koji će vrijediti da početna vrijednost registra TCNTx nije manja od 0. Frekvencija radnog takta jest 16 MHz, a vrijeme između dva prekida koju moramo ostvariti jest 100 ms. Pokušajmo za sklop *Timer/Counter1* u relaciju (9.6) uvrstiti $PRESCALER1 = 8$ te vrijeme $t_{T0} = 100$ ms = 0.1 s:

$$TCNT1_0 = 65536 - 0,1 \cdot \frac{16000000}{8} = -134464. \tag{9.8}$$

Početna vrijednost registra TCNT1 prema (9.8) manja je od 0 (izlazi iz opsega registra TCNT1) pa je potrebno pokušati sa sljedećim većim djeliteljem frekvencije radnog takta ($PRESCALER1 = 64$):

$$TCNT1_0 = 65536 - 0,1 \cdot \frac{16000000}{64} = 40536. \tag{9.9}$$

Nova početna vrijednost registra TCNT1 prema (9.9) pozitivna je i manja od 65535 pa možemo završiti s proračunom. I za djelitelje frekvencije $PRESCALER1 = 256$ i $PRESCALER1 = 1024$, početna vrijednost registra TCNT1 bit će unutar opsega registra, no s najmanjim mogućim djeliteljem frekvencije dobije se najbolja razlučivost vremena. Ako se pri proračunu početne vrijednosti dobije broj koji ne stane u registar (npr. za registar TCNT0 početna vrijednost je veća od 255, a za registar TCNT1 početna vrijednost je veća od 65535), tada u jednom pozivu prekida nije moguće ostvariti zadano vrijeme.

Ako početna vrijednost registra tajmera prema relacijama (9.5) - (9.7) nije cijeli broj, tada je početnu vrijednost potrebno zaokružiti na najbliži cijeli broj. U tom slučaju stvarno vrijeme između dvaju prekida neće biti jednako zadanom vremenu (na primjer, razlikovat će se 0.1%). Nakon što smo odredili sve parametre za konfiguraciju sklopa *Timer/Counter1*, preostaje nam napisati prekidnu rutinu za prekidni vektor TIMER1_OVF_vect. Prekidna rutina za prekidni vektor TIMER1_OVF_vect prikazana je programskim kodom 9.4. Navedena prekidna rutina poziva se svakih 100 ms s obzirom na odabrani djelitelj frekvencije $PRESCALER1 = 64$.

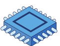
Programski kod 9.4: Prekidna rutina za prekidni vektor TIMER1_OVF_vect koja se poziva svakih 100 ms

```
ISR(TIMER1_OVF_vect){
    TCNT1 = 40536;
    // niz naredbi
}
```

Prekidna rutina TIMER1_OVF_vect poziva se onog trenutka kada dođe do preljeva u registru TCNT1, odnosno kada vrijednost registra TCNT1 prelazi iz 65535 u 0. U trenutku kada se dogodio prekid, počinje se izvršavati prekidna rutina TIMER1_OVF_vect u kojoj se najprije postavlja početna vrijednost registra TCNT1 na vrijednost 40536. Registar TCNT1 prilikom brojanja impulsa u ovom slučaju poprima sljedeće vrijednosti 40536, 40537 ,.., 65533, 65534, 65535, (0) $\rightarrow$ 40536, 40537 , ...

U prethodnim programskim kodovima prikazana je konfiguracija tajmera pomoću registara. Kroz vježbe ćemo dodatno prikazati konfiguriranje tajmera pomoću funkcija čije se definicije nalaze u biblioteci `timer.h`.

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `Timer.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `Timer.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\Timer\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Tajmeri i brojači`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

 Vježba 9.1

Napišite program koji će mijenjati stanje žute LED diode na razvojnom okruženju s mikroupravljačem ATmega328P svakih 16 ms u strogo zajamčenom vremenu pomoću sklopa *Timer/Counter0*. Istovremeno je potrebno napisati dio programskog koda kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini i ispisivati je u °C na LCD displeju jednom u sekundi. Prema shemi na slici 4.1, žuta LED dioda spojena je na digitalni izlaz PB2 mikroupravljača ATmega328P. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba91.cpp`. Omogućite prevođenje datoteke `vjezba91.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba91.cpp` prikazan je programskim kodom 9.5.

Programski kod 9.5: Početni sadržaj datoteke `vjezba91.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"

// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    DDRB |= (1 << PB2); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

U ovoj vježbi koristit ćemo tajmere u svrhu obavljanja vremenski zadataka u strogo zajamčenom vremenu. To znači da se neki zadatak mora obaviti u točno zadanim vremenskim intervalima bez obzira na to kakav se programski kod izvodi u glavnom programu (funkcija `main()`). Do sada smo za vremenske zadatke koristili funkciju `_delay_ms()`. Već smo spomenuli da takvo rješenje nije zgodno jer ova funkcija doslovno zaustavlja program na određeno vrijeme. Nadalje, korištenjem funkcija poput `lcdprintf()`, `adcRead()` i drugih, u programski kod se unosi kašnjenje koje korisnik ne vidi jer se unutar definicija tih funkcija nalaze pozivi funkcije `_delay_ms()`, a vrijeme izvođenja pojedinih naredbi unutar funkcija zahtjeva potrošnju procesorskog vremena. Prema tome, ako u `while` petlji unutar funkcije `main()` pozovemo funkciju `_delay_ms(1000)`, kašnjenje će biti sigurno veće od 1000 ms, a prema tome i vremenski zadatak neće biti izvršen u strogo zajamčenom vremenu. Strogo zajamčeno vrijeme osigurat ćemo pomoću tajmera koji radi u prekidnom načinu rada. Nakon što istekne zadano vrijeme

koje se mjeri tajmerom, prekida se glavni program, a izvršava se dio programskog koda u strogo zajamčenom vremenu.

U programskom kodu 9.5 priikazan je dio programa (prema primjeru u vježbi 7.1) koji mjeri temperaturu te ju ispisuje na LCD displej. Unutar `while` petlje nalazi se kašnjenje iznosa 1000 ms. Naš zadatak je osigurati promjenu stanja žute diode svakih 16 ms. Uz kašnjenje u iznosu 1000 ms, promjena stanja žute LED diode svakih 16 ms nije moguće. U tu svrhu koristiti ćemo tajmer, odnosno kako je vježbom zadano sklop *Timer/Counter0* kojim će se generirati prekid na preljev registra `TCNT0`. Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT0` prikazana je programskim kodom 9.6. Ovu prekidnu rutinu dodajte u programski kod 9.5 iznad definicije inicijalizacijske funkcije `init()`. Korištenje prekida u programskom kodu zahtjeva uključenje zaglavlja `interrupt.h` pomoću naredbe `#include <avr/interrupt.h>`. Autori su pripremili zaglavlje `interrupt.h` koje u programski kod 9.5 uključite pomoću naredbe `#include "Interrupt/interrupt.h"`. Unutar ovog zaglavlja nalazi se poziv naredbe `#include <avr/interrupt.h>`. Za globalno omogućenje prekida u funkciju `init()` upišite naredbu `interruptEnable()`.

Programski kod 9.6: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT0`

```
ISR(TIMER0_OVF_vect) {

}
```

Da bismo pomoću tajmera osigurali izvršenje vremenskog zadatka svakih 16 ms, potrebno je tajmer konfigurirati u normalnom načinu rada, namjestiti djelitelj frekvencije radnog takta te odrediti početnu vrijednost registra `TCNT0` pomoću relacije (9.5). Djelitelj frekvencije za sklop *Timer/Counter0* može se izabrati pomoću tablice 9.1. Prema proceduri koju smo opisali u teorijskom djelu ovog poglavlja, za djelitelj frekvencije odabrat ćemo 1024 (najveći mogući). Prema relaciji (9.5), početna vrijednost registra `TCNT0` bit će:

$$TCNT0_0 = 256 - 0,016 \cdot \frac{16000000}{1024} = 6. \tag{9.10}$$

Konfiguriranje sklopa *Timer/Counter0* za normalan način rada provodi se u registru `TCCR0A` i registru `TCCR0B` pomoću bitova `WGM00`, `WGM01` i `WGM02`, a prema tablici 9.2. Teoretski, s obzirom da svi navedeni bitovi u normalnom načinu rada imaju vrijednost 0, konfiguracija registara `TCCR0A` i `TCCR0B` može se ispustiti jer početno svi bitovi registara imaju vrijednost 0. Preporuka jest da se ipak provede konfiguracija kako bi se osobi koja čita programski kod dalo do znanja da se koristi normalan način rada tajmera. Djelitelj frekvencije radnog takta konfigurira se u registru `TCCR0B` pomoću bitova `CS00`, `CS01` i `CS02`, a prema tablici 9.1. Za djelitelj frekvencije iznosa 1024 bit `CS00` mora imati vrijednost 1, bit `CS01` vrijednost 0, a bit `CS02` vrijednost 1. Omogućenje prekida sklopa *Timer/Counter0* postiže se registrom `TIMSK0` tako da se bit `TOIE0` postavi u 1. Onog trenutka kada se omogući prekid pomoću bita `TOIE0` i postavi djelitelj frekvencije, sklop *Timer/Counter0* će početi izazivati prekide. Iz tog je razloga i za prvi pozvani prekid potrebno namjestiti izračunatu početnu vrijednost registra `TCNT0`. Konfiguracija sklopa *Timer/Counter0* prema navedenom opisu prikazana je programskim kodom 9.7.

Programski kod 9.7: Konfiguracija sklopa *Timer/Counter0*: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 1024, omogućenje prekida i postavljanje početne vrijednosti registra `TCNT0`

```
// postavljanje normalnog načina rada
TCCR0A |= (0 << WGM01) | (0 << WGM00);
TCCR0B |= (0 << WGM02);
// djelitelj frekvencije F_CPU / 1024
TCCR0B |= ((1 << CS02) | (0 << CS01) | (1 << CS00));
TIMSK0 |= (1 << TOIE0); // omogućenje prekida preljevom za timer0
```

```
TCNT0 = 6; // početna vrijednost TCNT0 za 16 ms
```

Programski kod 9.7 potrebno je upisati u inicijalizacijsku funkciju `init()`. Preostaje nam još realizirati izmjenu stanja žute LED diode. U prekidnu rutinu prikazanu programskim kodom 9.6 potrebno je upisati sljedeće naredbe:

- `TCNT0 = 6;` - postavljanje početne vrijednosti registra `TCNT0` na vrijednost izračunatu relacijom (9.10) kako bi se prekid sklopa *Timer/Counter0* dogodio svakih 16 ms.
- `PORTB ^= (1 << PB2);` - bitovni operator `^` mijenja stanje bita na poziciji pina PB2, odnosno mijenja stanje žute LED diode.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba91.cpp` treba biti ista kao programski kod 9.8.

Programski kod 9.8: Program kojim se ispisuje temperatura te mijenja stanje žute LED diode svakih 16 ms u strogo zajamčenom vremenu - prvi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

// prekidna rutina za timer 0
ISR(TIMER0_OVF_vect) {
    TCNT0 = 6; // početna vrijednost TCNT0 za 16 ms
    PORTB ^= (1 << PB2); // promjena stanja na žute LED diode
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    DDRB |= (1 << PB2); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    TCCR0A |= (0 << WGM01) | (0 << WGM00);
    TCCR0B |= (0 << WGM02);
    // djelitelj frekvencije F_CPU / 1024
    TCCR0B |= ((1 << CS02) | (0 << CS01) | (1 << CS00));
    TIMSK0 |= (1 << TOIE0); // omogućenje prekida preljevom za timer0
    TCNT0 = 6; // početna vrijednost TCNT0 za 16 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba91.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili

navedene korake, na LCD displeju ispisivat će se temperatura okoline razvojnog okruženja, a žuta LED dioda izmjenjivati će svoje stanje svakih 16 ms u strogo zajamčenom vremenu.

Radi jednostavnosti konfiguracije tajmera, autori udžbenika napisali su funkcije kojima se konfigurira normalan način rada tajmera, podešava se djelitelj frekvencije i omogućuje se prekidni način rada sklopa *Timer/Counter0*. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. Funkcije kojima se konfigurira sklop *Timer/Counter0* su:

- `timer0NormalMode()` - funkcija koja sklop *Timer/Counter0* konfigurira za normalan način rada,
- `timer0InterruptOVFEnable()` - funkcija kojom se omogućuje prekidni način rada sklopa *Timer/Counter0*,
- `timer0InterruptOVFDisable()` - funkcija kojom se onemogućuje prekidni način rada sklopa *Timer/Counter0*,
- `timer0SetPrescaler(uint8_t prescaler)` - funkcija kojom se konfigurira djelitelj frekvencije radnog takta, a kao argument prima konstantu koje su definirane u zaglavlju `timer.h`,
- `timer0SetValue(uint8_t value)` - funkcija kojom se postavlja vrijednost registra `TCNT0` pomoću argumenta `value`,
- `timer0GetValue()` - funkcija kojom se čita vrijednost registra `TCNT0`. Širina podatka jest 8 bitova.

Funkcija `timer0SetPrescaler(uint8_t prescaler)` kao argument prima predefinirane konstante za sklop *Timer/Counter0* koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 9.9.

Programski kod 9.9: Definirane konstante za djelitelje frekvencije radnog takta za sklop *Timer/Counter0* u datoteci `timer.h`

```
//djelitelji frekvencije za sklop Timer/Counter0
#define TIMER0_PRESCALER_OFF         ((0 << CS02) | (0 << CS01) | (0 << CS00))
#define TIMER0_PRESCALER_1           ((0 << CS02) | (0 << CS01) | (1 << CS00))
#define TIMER0_PRESCALER_8           ((0 << CS02) | (1 << CS01) | (0 << CS00))
#define TIMER0_PRESCALER_64          ((0 << CS02) | (1 << CS01) | (1 << CS00))
#define TIMER0_PRESCALER_256         ((1 << CS02) | (0 << CS01) | (0 << CS00))
#define TIMER0_PRESCALER_1024        ((1 << CS02) | (0 << CS01) | (1 << CS00))
#define TIMER0_EXTERNAL_FALL_EDGE    ((1 << CS02) | (1 << CS01) | (0 << CS00))
#define TIMER0_EXTERNAL_RISI_EDGE    ((1 << CS02) | (1 << CS01) | (1 << CS00))
```

Pokažimo nekoliko primjera korištenja funkcija za sklop *Timer/Counter0* koje se nalaze u zaglavlju `timer.h`:

- `timer0SetPrescaler(TIMER0_PRESCALER_OFF)` - sklop *Timer/Counter0* je zaustavljen,
- `timer0SetPrescaler(TIMER0_PRESCALER_8)` - djelitelj frekvencije za sklop *Timer/Counter0* iznosi 8,
- `timer0SetPrescaler(TIMER0_PRESCALER_256)` - djelitelj frekvencije za sklop *Timer/Counter0* iznosi 256,
- `timer0SetPrescaler(TIMER0_EXTERNAL_RISI_EDGE)` - sklop *Timer/Counter0* broji rastući brid signala iz vanjskog izvora koji je spojen na pin T0 (PD4),

- `timer0SetValue(120)` - postavljanje vrijednosti registra `TCNT0` na iznos 120,
- `timer0Value = timer0GetValue()` - čitanje vrijednosti registra `TCNT0` i zapisivanje u varijablu `timer0Value`.

Konfiguracija sklopa *Timer/Counter0*, korištenjem funkcijskog pristupa, prikazana je programskim kodom 9.10. Na ovaj način konfiguracija je postignuta bez da se postavljaju bitovi u registre za konfiguraciju sklopa *Timer/Counter0*.

Programski kod 9.10: Konfiguracija sklopa *Timer/Counter0* funkcijskim pristupom: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 1024, omogućenje prekida i postavljanje početne vrijednosti registra `TCNT0`

```
// postavljanje normalnog načina rada
timer0NormalMode();
// djelitelj frekvencije F_CPU / 1024
timer0SetPrescaler(TIMER0_PRESCALER_1024);
timer0InterruptOVFEnable(); // omogućenje prekida preljevom za timer0
timer0SetValue(6); // početna vrijednost TCNT0 za 16 ms
```

U datoteku `vjezba91.cpp` najprije dodajte naredbu `#include "Timer/timer.h"`. Programski kod 9.10 potrebno je upisati u inicijalizacijsku funkciju `init()` umjesto konfiguracije prikazane programskim kodom 9.7. U prekidnu rutinu prikazanu programskim kodom 9.6 potrebno je upisati sljedeće naredbe umjesto naredbi koje se trenutno nalaze u prekidnoj rutini:

- `timer0SetValue(6);` - postavljanje početne vrijednosti registra `TCNT0` na vrijednost izračunatu relacijom (9.10) kako bi se prekid sklopa *Timer/Counter0* dogodio svakih 16 ms.
- `toggle_pin(PORTB, PB2);` - makronaredba koja mijenja stanje bita na poziciji pina PB2, odnosno mijenja stanje žute LED diode.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba91.cpp` treba biti ista kao programski kod 9.11.

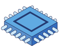
Programski kod 9.11: Program kojim se ispisuje temperatura te mijenja stanje žute LED diode svakih 16 ms u strogo zajamčenom vremenu - drugi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

// prekidna rutina za timer 0
ISR(TIMER0_OVF_vect) {
    timer0SetValue(6); // početna vrijednost TCNT0 za 16 ms
    toggle_pin(PORTB, PB2); // promjena stanja na žute LED diode
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    config_output(DDRB, PB2);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    timer0NormalMode();
```

```
    // djelitelj frekvencije F_CPU / 1024
    timer0SetPrescaler(TIMER0_PRESCALER_1024);
    timer0InterruptOVFEnable(); // omogućenje prekida preljevom za timer0
    timer0SetValue(6); // početna vrijednost TCNT0 za 16 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba91.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, funkcionalnost razvojnog sklopa ostala je ista kao i u prethodnom testiranju.

Zatvorite datoteku `vjezba91.cpp` i onemogućite prevođenje ove datoteke.



Vježba 9.2

Napišite program koji će mijenjati stanje zelene LED diode na razvojnom okruženju s mikroupravljačem ATmega328P svakih 250 ms u strogo zajamčenom vremenu pomoću sklopa *Timer/Counter1*. Istovremeno je potrebno napisati dio programskog koda kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini i ispisivati je u °C na LCD displeju jednom u sekundi. Prema shemi na slici 4.1, zelena LED dioda spojena je na digitalni izlaz PB3 mikroupravljača ATmega328P. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba92.cpp`. Omogućite prevođenje datoteke `vjezba92.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba92.cpp` prikazan je programskim kodom 9.12.

Programski kod 9.12: Početni sadržaj datoteke `vjezba92.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"

// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    config_output(DDRB, PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
}

int main(void) {
```

```
    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Naš zadatak je osigurati promjenu stanja zelene diode svakih 250 ms pomoću sklopa *Timer/Counter1* u strogo zajamčenom vremenu. Kao i u prethodnoj vježbi, u programskom kodu 9.12 prikazan je dio programa koji mjeri temperaturu pomoću senzora LM35 te ju ispisuje na LCD displej. Unutar `while` petlje nalazi se kašnjenje iznosa 1000 ms.

Za ostvarivanje vremenskog zadatka svakih 250 ms u strogo zajamčenom vremenu koristiti ćemo tajmer, odnosno kako je vježbom zadano sklop *Timer/Counter1* kojim će se generirati prekid na preljev registra `TCNT1`. Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT1` prikazana je programskim kodom 9.13. Ovu prekidnu rutinu dodajte u programski kod 9.12 iznad definicije inicijalizacijske funkcije `init()`. U programski kod 9.12 uključite zaglavlje `interrupt.h` pomoću naredbe `#include "Interrupt/interrupt.h"`.

Programski kod 9.13: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT1`

```
ISR(TIMER1_OVF_vect) {

}
```

Da bismo pomoću tajmera osigurali izvršenje vremenskog zadatka svakih 250 ms, potrebno je tajmer konfigurirati u normalnom načinu rada, namjestiti djelitelj frekvencije radnog takta te odrediti početnu vrijednost registra `TCNT1` pomoću relacije (9.6). Djelitelj frekvencije za sklop *Timer/Counter1* može se izabrati pomoću tablice 9.1. Odaberimo najprije djelitelj frekvencije iznosa 8 i uvrstimo ga u relaciju (9.6):

$$TCNT1_0 = 65536 - 0,25 \cdot \frac{16000000}{8} = -434464. \tag{9.11}$$

Dobivena početna vrijednost registra `TCNT1` jest negativna, što znači da zadano vrijeme nije moguće realizirati s odabranim djeliteljem frekvencije. U sljedećoj iteraciji proračuna početne vrijednosti registra `TCNT1` odabrat ćemo djelitelj frekvencije iznosa 64 te ga uvrstiti u relaciju (9.6):

$$TCNT1_0 = 65536 - 0,25 \cdot \frac{16000000}{64} = 3036. \tag{9.12}$$

U ovom slučaju dobiveno je pozitivno rješenje koje se može zapisati u registra širine 16 bitova. Prema tome, početna vrijednost registra `TCNT1` kojom će se ostvariti promjena zelene LED diode svakih 250 ms iznosi 3036.

Konfiguriranje sklopa *Timer/Counter1* za normalan način rada provodi se u registru `TCCR1A` i registru `TCCR1B` pomoću bitova `WGM10`, `WGM11` i `WGM12`, a prema tablici 15-4 u literaturi [2]. Djelitelj frekvencije radnog takta konfigurira se u registru `TCCR1B` pomoću bitova `CS10`, `CS11` i `CS12`, a prema tablici 15-5 u literaturi [2]. Za djelitelj frekvencije iznosa 64 bit `CS10` mora imati vrijednost 1, bit `CS11` vrijednost 1, a bit `CS12` vrijednost 0. Omogućenje prekida sklopa *Timer/Counter1* postiže se registrom `TIMSK1` tako da se bit `TOIE1` postavi u 1. Onog trenutka kada se omogući prekid pomoću bita `TOIE1` i postavi djelitelj frekvencije, sklop *Timer/Counter1*

će početi izazivati prekide. Iz tog je razloga za prvi pozvani prekid potrebno namjestiti izračunatu početnu vrijednost registra TCNT1. Konfiguracija sklopa *Timer/Counter1* prema navedenom opisu prikazana je programskim kodom 9.14.

Programski kod 9.14: Konfiguracija sklopa *Timer/Counter1*: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 64, omogućenje prekida i postavljanje početne vrijednosti registra TCNT1

```
// postavljanje normalnog načina rada
TCCR1A |= (0 << WGM11) | (0 << WGM10);
TCCR1B |= (0 << WGM13) | (0 << WGM12);
// djelitelj frekvencije F_CPU / 64
TCCR1B |= ((0 << CS12) | (1 << CS11) | (1 << CS10));
TIMSK1 |= (1 << TOIE1); // omogućenje prekida preljevom za timer1
TCNT1 = 3036; // početna vrijednost TCNT1 za 250 ms
```

Programski kod 9.14 potrebno je upisati u inicijalizacijsku funkciju `init()`. Preostaje nam još realizirati izmjenu stanja zelene LED diode. U prekidnu rutinu prikazanu programskim kodom 9.13 potrebno je upisati sljedeće naredbe:

- `TCNT1 = 3036;` - postavljanje početne vrijednosti registra TCNT1 na vrijednost izračunatu relacijom (9.12) kako bi se prekid sklopa *Timer/Counter1* dogodio svakih 250 ms.
- `toggle_pin(PORTB, PB3);` - makronaredba koja mijenja stanje bita na poziciji pina PB3, odnosno mijenja stanje zelene LED diode.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba92.cpp` treba biti ista kao programski kod 9.15.

Programski kod 9.15: Program kojim se ispisuje temperatura te mijenja stanje zelene LED diode svakih 250 ms u strogo zajamčenom vremenu - prvi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0
// prekidna rutina za timer 1
ISR(TIMER1_OVF_vect) {
    TCNT1 = 3036; // početna vrijednost TCNT1 za 250 ms
    toggle_pin(PORTB, PB3); // promjena stanja na zelene LED diode
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
     // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    config_output(DDRB, PB3);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    TCCR1A |= (0 << WGM11) | (0 << WGM10);
    TCCR1B |= (0 << WGM13) | (0 << WGM12);
    // djelitelj frekvencije F_CPU / 64
    TCCR1B |= ((0 << CS12) | (1 << CS11) | (1 << CS10));
    TIMSK1 |= (1 << TOIE1); // omogućenje prekida preljevom za timer1
    TCNT1 = 3036; // početna vrijednost TCNT1 za 250 ms
}

int main(void) {
```

```
    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba92.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se temperatura okoline razvojnog okruženja, a zelena LED dioda izmjenjivati će svoje stanje svakih 250 ms u strogo zajamčenom vremenu.

Radi jednostavnosti konfiguracije tajmera i brojača, autori udžbenika napisali su funkcije kojima se konfigurira normalan način rada tajmera, podešava se djelitelj frekvencije i omogućuje se prekidni način rada sklopa *Timer/Counter1*. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. Funkcije kojima se konfigurira sklop *Timer/Counter1* su:

- `timer1NormalMode()` - funkcija koja sklop *Timer/Counter1* konfigurira za normalan način rada,
- `timer1InterruptOVFEnable()` - funkcija kojom se omogućuje prekidni način rada sklopa *Timer/Counter1*,
- `timer1InterruptOVFDisable()` - funkcija kojom se onemogućuje prekidni način rada sklopa *Timer/Counter1*,
- `timer1SetPrescaler(uint8_t prescaler)` - funkcija kojom se konfigurira djelitelj frekvencije radnog takta, a kao argument prima konstantu koje su definirane u zaglavlju `timer.h`,
- `timer1SetValue(uint16_t value)` - funkcija kojom se postavlja vrijednost registra `TCNT1` pomoću argumenta `value`,
- `timer1GetValue()` - funkcija kojom se čita vrijednost registra `TCNT1`. Širina podatka jest 16 bitova.

Funkcija `timer1SetPrescaler(uint8_t prescaler)` kao argument prima predefinirane konstante za sklop *Timer/Counter1* koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 9.16.

Programski kod 9.16: Definirane konstante za djelitelje frekvencije radnog takta za sklop *Timer/Counter1* u datoteci `timer.h`

```
//djelitelji frekvencije za sklop Timer/Counter1
#define TIMER1_PRESCALER_OFF        ((0 << CS12) | (0 << CS11) | (0 << CS10))
#define TIMER1_PRESCALER_1          ((0 << CS12) | (0 << CS11) | (1 << CS10))
#define TIMER1_PRESCALER_8          ((0 << CS12) | (1 << CS11) | (0 << CS10))
#define TIMER1_PRESCALER_64         ((0 << CS12) | (1 << CS11) | (1 << CS10))
#define TIMER1_PRESCALER_256        ((1 << CS12) | (0 << CS11) | (0 << CS10))
#define TIMER1_PRESCALER_1024       ((1 << CS12) | (0 << CS11) | (1 << CS10))
#define TIMER1_EXTERNAL_FALL_EDGE   ((1 << CS12) | (1 << CS11) | (0 << CS10))
#define TIMER1_EXTERNAL_RISI_EDGE   ((1 << CS12) | (1 << CS11) | (1 << CS10))
```

Pokažimo nekoliko primjera korištenja funkcija za sklop *Timer/Counter1* koje se nalaze u zaglavlju `timer.h`:

- `timer1SetPrescaler(TIMER1_PRESCALER_1)` - djelitelj frekvencije za sklop *Timer/Counte1* iznosi 1 (direktno brojanje impulsa),
- `timer1SetPrescaler(TIMER1_PRESCALER_64)` - djelitelj frekvencije za sklop *Timer/Counter1* iznosi 64,
- `timer1SetPrescaler(TIMER1_EXTERNAL_FALL_EDGE)` - sklop *Timer/Counter1* broji padajući brid signala iz vanjskog izvora koji je spojen na pin T1 (PD5),
- `timer1SetValue(25000)` - postavljanje vrijednosti registra `TCNT1` na iznos 25000,
- `timer1Value = timer1GetValue()` - čitanje vrijednosti registra `TCNT1` i zapisivanje u varijablu `timer1Value`.

Konfiguracija sklopa *Timer/Counter1*, korištenjem funkcijskog pristupa, prikazana je programskim kodom 9.17. Na ovaj način konfiguracija je postignuta bez da se postavljaju bitovi u registre za konfiguraciju sklopa *Timer/Counter1*.

Programski kod 9.17: Konfiguracija sklopa *Timer/Counter1* funkcijskim pristupom: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 64, omogućenje prekida i postavljanje početne vrijednosti registra `TCNT1`

```
// postavljanje normalnog načina rada
timer1NormalMode();
// djelitelj frekvencije F_CPU / 64
timer1SetPrescaler(TIMER1_PRESCALER_64);
timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
timer1SetValue(3036); // početna vrijednost TCNT1 za 250 ms
```

Programski kod 9.17 potrebno je upisati u inicijalizacijsku funkciju `init()` umjesto konfiguracije prikazane programskim kodom 9.14. U prekidnu rutinu prikazanu programskim kodom 9.13 potrebno je naredbu `TCNT1 = 3036;` zamijeniti sa sljedećom naredbom:

- `timer1SetValue(3036);` - postavljanje početne vrijednosti registra `TCNT1` na vrijednost izračunatu relacijom (9.12) kako bi se prekid sklopa *Timer/Counter1* dogodio svakih 250 ms.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba92.cpp` treba biti ista kao programski kod 9.18.

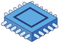
Programski kod 9.18: Program kojim se ispisuje temperatura te mijenja stanje zelene LED diode svakih 250 ms u strogo zajamčenom vremenu - drugi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0
// prekidna rutina za timer 1
ISR(TIMER1_OVF_vect) {
    timer1SetValue(3036); // početna vrijednost TCNT1 za 250 ms
    toggle_pin(PORTB, PB3); // promjena stanja na zelene LED diode
}
```

```
void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    config_output(DDRB, PB3);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    timer1NormalMode();
    // djelitelj frekvencije F_CPU / 64
    timer1SetPrescaler(TIMER1_PRESCALER_64);
    timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
    timer1SetValue(3036); // početna vrijednost TCNT1 za 250 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja + home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba92.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, funkcionalnost razvojnog sklopa ostala je ista kao i u prethodnom testiranju.

Zatvorite datoteku `vjezba92.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 9.3

Napišite program koji će mijenjati stanje crvene LED diode na razvojnom okruženju s mikroupravljačem ATmega328P svakih 120 ms u strogo zajamčenom vremenu pomoću sklopa *Timer/Counter2*. Istovremeno je potrebno napisati dio programskog koda kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini i ispisivati je u °C na LCD displeju jednom u sekundi. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni izlaz PB1 mikroupravljača ATmega328P. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba93.cpp`. Omogućite prevođenje datoteke `vjezba93.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba93.cpp` prikazan je programskim kodom 9.19.

Programski kod 9.19: Početni sadržaj datoteke `vjezba93.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"

// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0
```

```
void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    config_output(DDRB, PB1);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Naš zadatak je osigurati promjenu stanja crvene LED diode svakih 120 ms pomoću sklopa *Timer/Counter2* u strogo zajamčenom vremenu uz prikaz temperature na LCD displeju. Za ostvarivanje strogo zajamčenog vremena koristiti ćemo sklop *Timer/Counter2* kojim će se generirati prekid na preljev registra `TCNT2`. Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT2` prikazana je programskim kodom 9.20. Ovu prekidnu rutinu dodajte u programski kod 9.19 iznad definicije inicijalizacijske funkcije `init()`. U programski kod 9.19 uključite zaglavlje `interrupt.h` pomoću naredbe `#include "Interrupt/interrupt.h"`.

Programski kod 9.20: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT2`

```
ISR(TIMER2_OVF_vect) {

}
```

Da bismo pomoću tajmera osigurali izvršenje vremenskog zadatka svakih 120 ms, potrebno je tajmer konfigurirati u normalnom načinu rada, namjestiti djelitelj frekvencije radnog takta te odrediti početnu vrijednost registra `TCNT2` pomoću relacije (9.7). Djelitelj frekvencije za sklop *Timer/Counter2* može se izabrati pomoću tablice 9.1. Prema iskustvu iz prve vježbe, odaberimo najveći djelitelj frekvencije iznosa 1024 i uvrstimo ga u relaciju (9.7):

$$TCNT2_0 = 256 - 0,12 \cdot \frac{16000000}{1024} = -1619. \tag{9.13}$$

Dobivena početna vrijednost registra `TCNT2` jest negativna, što znači da zadano vrijeme nije moguće realizirati s odabranim djeliteljem frekvencije. S obzirom da smo već odabrali najveći djelitelj frekvencije, nismo u mogućnosti povećavati ga. Zaključak jest da vrijeme iznosa 120 ms između dva prekida nije moguće realizirati pomoću sklopa *Timer/Counter2*. Ipak, postoji rješenje u ovoj situaciji. Iskustvo iz prve vježbe jest da početna vrijednost registra `TCNT0` iznosi 6 za vrijeme 16 ms što ukazuje da je 16 ms najveće cjelobrojno vrijeme u ms koje se može realizirati sklopom *Timer/Counter2*. Registar `TCNT2` iste je širine (8 bitova) kao i registar `TCNT0`. Vrijeme iznosa 120 ms mogli bismo realizirati tako da ga podijelimo na manja vremena na neki od sljedećih načina (nisu navedeni svi mogući načini):

- vrijeme iznosa 120 ms može se realizirati tako da se prekidna rutina sklopa *Timer/Counter2* pozove 10 puta s vremenskim razmacima iznosa 12 ms,
- vrijeme iznosa 120 ms može se realizirati tako da se prekidna rutina sklopa *Timer/Counter2* pozove 12 puta s vremenskim razmacima iznosa 10 ms,

- vrijeme iznosa 120 ms može se realizirati tako da se prekidna rutina sklopa *Timer/Counter2* pozove 15 puta s vremenskim razmacima iznosa 8 ms.

U prethodnim primjerima, vremenski zadatak koji se izvršava svakih 120 ms provest ćemo samo u zadnjem pozivu prekidne rutine koja se poziva nakon svakih 12, 10 i 8 ms. Ostali pozivi prekidne rutine će biti pozivi u praznom hodu.

Uz djelitelj frekvencije iznosa 1024, za navedena vremena 12, 10 i 8 ms prema relaciji (9.7) izračunat ćemo početne vrijednosti registra TCNT2.

Za vrijeme između dva poziva prekidne rutine iznosa 12 ms vrijedi:

$$TCNT2_0 = 256 - 0,012 \cdot \frac{16000000}{1024} = 68.5. \tag{9.14}$$

Za vrijeme između dva poziva prekidne rutine iznosa 10 ms vrijedi:

$$TCNT2_0 = 256 - 0,01 \cdot \frac{16000000}{1024} = 99.75. \tag{9.15}$$

Za vrijeme između dva poziva prekidne rutine iznosa 8 ms vrijedi:

$$TCNT2_0 = 256 - 0,008 \cdot \frac{16000000}{1024} = 131. \tag{9.16}$$

Jedino vrijeme između dva poziva prekidne rutine koje rezultira cjelobrojnom početnom vrijednosti registra TCNT2 jest 8 ms pa ćemo vremenski zadatak koji se izvršava svakih 120 ms realizirati pomoću 15 poziva prekidne rutine sklopa *Timer/Counter2* svakih 8 ms. Ako bismo uzastopne pozive prekidnih rutina realizirali s početnim vrijednostima koje nisu cjelobrojne, akumulirali bismo pogrešku. Na primjer, za početnu vrijednost danu relacijom (9.14), realizirano vrijeme bi prema relaciji 9.4 bilo 119.68 ms umjesto zadanog 120 ms jer bismo početnu vrijednost morali zaokružiti na najbliže cijelo:

$$10 \cdot t_{T2} = 10\frac{1024}{16000000} \cdot (256 - 69) = 0.11968. \tag{9.17}$$

Primijetite da smo u relaciji 9.17 vrijeme između dva poziva prekidne rutine pomnožili s brojem koraka kojim se ostvaruje zadano vrijeme iznosa 120 ms.

Konfiguriranje sklopa *Timer/Counter2* za normalan način rada provodi se u registru TCCR2A i registru TCCR2B pomoću bitova WGM20, WGM21 i WGM22, a prema tablici 17-8 u literaturi [2]. Djelitelj frekvencije radnog takta konfigurira se u registru TCCR2B pomoću bitova CS20, CS21 i CS22, a prema tablici 17-9 u literaturi [2]. Za djelitelj frekvencije iznosa 1024, bitovi CS20, CS21 i CS22 moraju imati vrijednost 1. Omogućenje prekida sklopa *Timer/Counter2* postiže se registrom TIMSK2 tako da se bit TOIE2 postavi u 1. Onog trenutka kada se omogući prekid pomoću bita TOIE2 i postavi djelitelj frekvencije, sklop *Timer/Counter2* će početi izazivati prekide. Iz tog je razloga i za prvi pozvani prekid potrebno namjestiti izračunatu početnu vrijednost registra TCNT2. Konfiguracija sklopa *Timer/Counter2* prema navedenom opisu prikazana je programskim kodom 9.21.

Programski kod 9.21: Konfiguracija sklopa *Timer/Counter2*: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 1024, omogućenje prekida i postavljanje početne vrijednosti registra TCNT2

```
// postavljanje normalnog načina rada
TCCR2A |= (0 << WGM21) | (0 << WGM20);
TCCR2B |= (0 << WGM22);
// djelitelj frekvencije F_CPU / 1024
TCCR2B |= ((1 << CS22) | (1 << CS21) | (1 << CS20));
```

```
TIMSK2 |= (1 << TOIE2); // omogućenje prekida preljevom za timer2
TCNT2 = 131; // početna vrijednost TCNT2 za 8 ms
```

Programski kod 9.21 potrebno je upisati u inicijalizacijsku funkciju `init()`. Preostaje nam još realizirati izmjenu stanja crvene LED diode. U tu svrhu potrebno je deklarirati jednu globalnu varijablu te dopuniti prekidnu rutinu 9.20 na sljedeći način:

- `volatile uint8_t timer2PrazanHod = 0;` - deklaracija globalne cjelobrojne varijable kojom će se brojati broj poziva prekidne rutine. Ovu deklaraciju dodajte iznad prekidne rutine za sklop *Timer/Counter2*. Primijetite da smo pri deklaraciji globalne varijable koristili ključnu riječ `volatile`. Kako smo objasnili u prethodnom poglavlju, ključna riječ `volatile` koristi se u slučaju kada se globalna varijabla mijenja unutar prekidne rutine.

- u prekidnu rutinu za sklop *Timer/Counter2* dodajte sljedeće naredbe:
  - `TCNT2 = 131;` - postavljanje početne vrijednosti registra `TCNT2` na vrijednost izračunatu relacijom (9.16) kako bi se prekid sklopa *Timer/Counter2* dogodio svakih 8 ms.
  - `timer2PrazanHod++;` - povećanje broj poziva prekidne rutine sklopa *Timer/Counter2* za 1.
  - `if (timer2PrazanHod == 15 ){}` - uvjetovani blok naredbi koji se izvodi svaki 15. poziv prekidne rutine sklopa *Timer/Counter2* jer je 15 * 8 ms = 120 ms. Unutar uvjetovanog `if` bloka dodajte sljedeće naredbe:
    - `timer2PrazanHod = 0;` - resetiranje brojača poziva prekidne rutine sklopa *Timer/Counter2* kako bi se ponovno izvršio ciklus od 15 * 8 ms = 120 ms.
    - `toggle_pin(PORTB, PB1);` - makronaredba koja mijenja stanje bita na poziciji pina PB1, odnosno mijenja stanje crvene LED diode.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba93.cpp` treba biti ista kao programski kod 9.22.

Programski kod 9.22: Program kojim se ispisuje temperatura te mijenja stanje crvene LED diode svakih 120 ms u strogo zajamčenom vremenu - prvi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

volatile uint8_t timer2PrazanHod = 0;
// prekidna rutina za timer 2
ISR(TIMER2_OVF_vect) {
    TCNT2 = 131; // početna vrijednost TCNT2 za 8 ms
    timer2PrazanHod++; // povecaj broj poziva za 1
    if (timer2PrazanHod == 15 ) { // ako je timer2PrazanHod = 15*8ms=120ms
        timer2PrazanHod = 0; // vrati broj poziva na 0
        toggle_pin(PORTB, PB1); // promjena stanja na crvene LED diode
    }
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
```

```
    // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    config_output(DDRB, PB1);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    TCCR2A |= (0 << WGM21) | (0 << WGM20);
    TCCR2B |= (0 << WGM22);
    // djelitelj frekvencije F_CPU / 1024
    TCCR2B |= ((1 << CS22) | (1 << CS21) | (1 << CS20));
    TIMSK2 |= (1 << TOIE2); // omogućenje prekida preljevom za timer2
    TCNT2 = 131; // početna vrijednost TCNT2 za 8 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja + home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba93.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se temperatura okoline razvojnog okruženja, a crvena LED dioda izmjenjivati će svoje stanje svakih 120 ms u strogo zajamčenom vremenu.

Radi jednostavnosti konfiguracije tajmera i brojača, autori udžbenika napisali su funkcije kojima se konfigurira normalan način rada tajmera, podešava se djelitelj frekvencije i omogućuje se prekidni način rada sklopa *Timer/Counter2*. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. Funkcije kojima se konfigurira sklop *Timer/Counter2* su:

- `timer2NormalMode()` - funkcija koja sklop *Timer/Counter2* konfigurira za normalan način rada,
- `timer2InterruptOVFEnable()` - funkcija kojom se omogućuje prekidni način rada sklopa *Timer/Counter2*,
- `timer2InterruptOVFDisable()` - funkcija kojom se onemogućuje prekidni način rada sklopa *Timer/Counter2*,
- `timer2SetPrescaler(uint8_t prescaler)` - funkcija kojom se konfigurira djelitelj frekvencije radnog takta, a kao argument prima konstantu koje su definirane u zaglavlju `timer.h`,
- `timer2SetValue(uint8_t value)` - funkcija kojom se postavlja vrijednost registra `TCNT2` pomoću argumenta `value`,
- `timer2GetValue()` - funkcija kojom se čita vrijednost registra `TCNT2`. Širina podatka jest 8 bitova.

Funkcija `timer2SetPrescaler(uint8_t prescaler)` kao argument prima predefinirane konstante za sklop *Timer/Counter2* koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 9.23.

Programski kod 9.23: Definirane konstante za djelitelje frekvencije radnog takta za sklop *Timer/Counter2* u datoteci `timer.h`

```
//djelitelji frekvencije za sklop Timer/Counter2
#define TIMER2_PRESCALER_OFF        ((0 << CS22) | (0 << CS21) | (0 << CS20))
#define TIMER2_PRESCALER_1          ((0 << CS22) | (0 << CS21) | (1 << CS20))
#define TIMER2_PRESCALER_8          ((0 << CS22) | (1 << CS21) | (0 << CS20))
#define TIMER2_PRESCALER_32         ((0 << CS22) | (1 << CS21) | (1 << CS20))
#define TIMER2_PRESCALER_64         ((1 << CS22) | (0 << CS21) | (0 << CS20))
#define TIMER2_PRESCALER_128        ((1 << CS22) | (0 << CS21) | (1 << CS20))
#define TIMER2_PRESCALER_256        ((1 << CS22) | (1 << CS21) | (0 << CS20))
#define TIMER2_PRESCALER_1024       ((1 << CS22) | (1 << CS21) | (1 << CS20))
```

Primijetite da sklop *Timer/Counter2* nema mogućnost brojanja rastućih i padajućih impulsa iz vanjskog izvora. Pokažimo nekoliko primjera korištenja funkcija za sklop *Timer/Counter2* koje se nalaze u zaglavlju `timer.h`:

- `timer2SetPrescaler(TIMER2_PRESCALER_32)` - djelitelj frekvencije za sklop *Timer/Counte2* iznosi 32,
- `timer2SetPrescaler(TIMER2_PRESCALER_128)` - djelitelj frekvencije za sklop *Timer/Counter2* iznosi 128,
- `timer2SetValue(69)` - postavljanje vrijednosti registra `TCNT2` na iznos 69,
- `timer2Value = timer2GetValue()` - čitanje vrijednosti registra `TCNT2` i zapisivanje u varijablu `timer2Value`.

Konfiguracija sklopa *Timer/Counter2*, korištenjem funkcijskog pristupa, prikazana je programskim kodom 9.24. Na ovaj način konfiguracija je postignuta bez da se postavljaju bitovi u registre za konfiguraciju sklopa *Timer/Counter2*.

Programski kod 9.24: Konfiguracija sklopa *Timer/Counter2* funkcijskim pristupom: postavljanje u normalan način rada, postavljanje djelitelja frekvencije na 1024, omogućenje prekida i postavljanje početne vrijednosti registra `TCNT2`

```
// postavljanje normalnog načina rada
timer2NormalMode();
// djelitelj frekvencije F_CPU / 1024
timer2SetPrescaler(TIMER2_PRESCALER_1024);
timer2InterruptOVFEnable(); // omogućenje prekida preljevom za timer2
timer2SetValue(131); // početna vrijednost TCNT2 za 8 ms
```

Programski kod 9.24 potrebno je upisati u inicijalizacijsku funkciju `init()` umjesto konfiguracije prikazane programskim kodom 9.21. U prekidnu rutinu prikazanu programskim kodom 9.20 potrebno je naredbu `TCNT2 = 131;` zamijeniti sa sljedećom naredbom:

- `timer2SetValue(131);` - postavljanje početne vrijednosti registra `TCNT2` na vrijednost izračunatu relacijom (9.16) kako bi se prekid sklopa *Timer/Counter2* dogodio svakih 8 ms.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba93.cpp` treba biti ista kao programski kod 9.25.

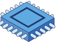
Programski kod 9.25: Program kojim se ispisuje temperatura te mijenja stanje crvene LED diode svakih 120 ms u strogo zajamčenom vremenu - drugi način

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
```

```
#include "Senzori/lm35.h
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

uint8_t timer2PrazanHod = 0;
// prekidna rutina za timer 2
ISR(TIMER2_OVF_vect) {
    timer2SetValue(131); // početna vrijednost TCNT2 za 8 ms
    timer2PrazanHod++; // povećaj broj poziva za 1
    if (timer2PrazanHod == 15 ) { // ako je timer2PrazanHod =15*8ms=120ms
        timer2PrazanHod = 0; // vrati broj poziva na 0
        toggle_pin(PORTB, PB1); // promjena stanja na crvene LED diode
    }
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    config_output(DDRB, PB1);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    timer2NormalMode();
    // djelitelj frekvencije F_CPU / 1024
    timer2SetPrescaler(TIMER2_PRESCALER_1024);
    timer2InterruptOVFEnable(); // omogućenje prekida preljevom za timer2
    timer2SetValue(131); // početna vrijednost TCNT2 za 8 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba93.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, funkcionalnost razvojnog sklopa ostala je ista kao i u prethodnom testiranju.

Zatvorite datoteku `vjezba93.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 9.4

Napišite program koji će na razvojnom okruženju s mikroupravljačem ATmega328P mijenjati stanje crvene LED diode svakih 400 ms pomoću sklopa *Timer/Counter0*, stanje žute LED diode svakih 400 ms pomoću sklopa *Timer/Counter1* te stanje zelene LED diode svakih 400 ms pomoću sklopa *Timer/Counter2* u strogo zajamčenom vremenu. Istovremeno je potrebno napisati dio programskog koda kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini i ispisivati je u °C na LCD displeju jednom u sekundi. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena

je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba94.cpp`. Omogućite prevođenje datoteke `vjezba94.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba94.cpp` prikazan je programskim kodom 9.26.

Programski kod 9.26: Početni sadržaj datoteke `vjezba94.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

// prekidna rutina za timer 0
ISR(TIMER0_OVF_vect) {

}

// prekidna rutina za timer 1
ISR(TIMER1_OVF_vect) {

}

// prekidna rutina za timer 2
ISR(TIMER2_OVF_vect) {

}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B1, OUTPUT);
    // PB2 konfiguriran kao izlazni pin (zuta LED dioda)
    pinMode(B2, OUTPUT);
    // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    pinMode(B3, OUTPUT);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Naš zadatak je osigurati promjenu stanja crvene, žute i zelene LED diode svakih 400 ms pomoću sklopova *Timer/Counter0*, *Timer/Counter1* i *Timer/Counter2* u strogo zajamčenom vremenu uz prikaz temperature na LCD displeju. Prekidne rutine koje se pozivaju pri preljevu registara `TCNT0`, `TCNT1` i `TCNT2` definirane su programskom kodu 9.26.

Da bismo pomoću tri tajmera osigurali izvršenje vremenskih zadataka svakih 400 ms, tajmere je potrebno konfigurirati u normalnom načinu rada, namjestiti djelitelj frekvencije radnog takta te odrediti početnu vrijednost registara `TCNT0`, `TCNT1` i `TCNT2` pomoću relacija (9.5) - (9.7). S obzirom da sklopovi *Timer/Counter0* i *Timer/Counter2* imaju istu rezoluciju, konfigurirat ćemo ih na isti način. Temeljem iskustva iz prethodnih vježbi, odaberimo najveći djelitelj frekvencije iznosa 1024 i uvrstimo ga u relacije (9.5) i (9.7):

$$TCNT0_0 = TCNT2_0 = 256 - 0,4 \cdot \frac{16000000}{1024} = -5994. \tag{9.18}$$

Dobivena početna vrijednost registara `TCNT0` i `TCNT2` jest negativna, što znači da zadano vrijeme nije moguće realizirati s odabranim djeliteljem frekvencije. S obzirom da smo već odabrali najveći djelitelj frekvencije, nismo u mogućnosti povećavati ga. Prema tome, moramo odabrati manje vrijeme poziva između dva prekida sklopova *Timer/Counter0* i *Timer/Counter2* te ga izvršiti n puta kako bismo ostvarili zadano vrijeme iznosa 400 ms. Prema prvoj vježbi, vrijeme između poziva prekida sklopova *Timer/Counter0* i *Timer/Counter2* neka je 16 ms, a broj poziva do izvršenja promjene stanja LED dioda će biti 25 (16 ms * 25 = 400 ms).

$$TCNT0_0 = TCNT2_0 = 256 - 0,016 \cdot \frac{16000000}{1024} = 6. \tag{9.19}$$

Prema tome, početna vrijednost registara `TCNT0` i `TCNT2` kojom će se ostvariti poziv prekidne rutine svakih 16 ms iznosi 6. Svaki 25. poziv prekidne rutine sklopa *Timer/Counter0* i *Timer/Counter2* će napraviti promjenu stanja crvene i zelene LED diode (16 ms * 25 = 400 ms).

Djelitelj frekvencije za sklop *Timer/Counter1* neka prema iskustvu iz druge vježbe ima iznos 64. Taj djelitelj frekvencije uvrstimo u relaciju (9.6):

$$TCNT1_0 = 65536 - 0,4 \cdot \frac{16000000}{64} = -34464. \tag{9.20}$$

Dobivena početna vrijednost registra `TCNT1` jest negativna, što znači da zadano vrijeme nije moguće realizirati s odabranim djeliteljem frekvencije. U sljedećoj iteraciji proračuna početne vrijednosti registra `TCNT1` odabrat ćemo djelitelj frekvencije iznosa 256 te ga uvrstiti u relaciju (9.6):

$$TCNT1_0 = 65536 - 0,4 \cdot \frac{16000000}{256} = 40536. \tag{9.21}$$

U ovom slučaju dobiveno je pozitivno rješenje koje se može zapisati u registra širine 16 bitova. Prema tome, početna vrijednost registra `TCNT1` kojom će se ostvariti promjena žute LED diode svakih 400 ms iznosi 40536.

Konfiguraciju sklopova *Timer/Counter0*, *Timer/Counter1* i *Timer/Counter2* potrebno je provesti unutar inicijalizacijske funkcije `init()` tako da napišete sljedeće naredbe:

- `timer0NormalMode();` - konfiguriranje sklopa *Timer/Counter0* za normalan način rada,
- `timer1NormalMode();` - konfiguriranje sklopa *Timer/Counter1* za normalan način rada,
- `timer2NormalMode();` - konfiguriranje sklopa *Timer/Counter2* za normalan način rada,

- `timer0SetPrescaler(TIMER0_PRESCALER_1024);` - djelitelj frekvencije sklopa *Timer/Counter0* namješten na iznos 1024,
- `timer1SetPrescaler(TIMER1_PRESCALER_256);` - djelitelj frekvencije sklopa *Timer/Counter1* namješten na iznos 256,
- `timer2SetPrescaler(TIMER2_PRESCALER_1024);` - djelitelj frekvencije sklopa *Timer/Counter2* namješten na iznos 1024,
- `timer0InterruptOVFEnable();` - omogućenje prekida za sklop *Timer/Counter0*,
- `timer1InterruptOVFEnable();` - omogućenje prekida za sklop *Timer/Counter1*,
- `timer2InterruptOVFEnable();` - omogućenje prekida za sklop *Timer/Counter2*,
- `timer0SetValue(6);` - inicijalizacija registra `TCNT0` na iznos 6 za vrijeme 16 ms,
- `timer1SetValue(40536);` - inicijalizacija registra `TCNT1` na iznos 40536 za vrijeme 400 ms,
- `timer2SetValue(6);` - inicijalizacija registra `TCNT2` na iznos 6 za vrijeme 16 ms.

Preostaje nam još realizirati izmjenu stanja LED dioda pomoću prekidnih rutina.

Za prekidnu rutinu `ISR(TIMER0_OVF_vect)` deklarirajte i inicijalizirajte varijablu kojom će se brojati broj poziva prekidnu rutinu tako da iznad prekidne rutine za sklop *Timer/Counter0* upišete naredbu `volatile uint8_t timer0PrazanHod = 0;`.

U prekidnu rutinu `ISR(TIMER0_OVF_vect)` dodajte sljedeće naredbe:

- `timer0SetValue(6);` - postavljanje početne vrijednosti registra `TCNT0` na vrijednost izračunatu relacijom (9.19) kako bi se prekid sklopa *Timer/Counter0* dogodio svakih 16 ms.
- `timer0PrazanHod++;` - povećanje broj poziva prekidne rutine sklopa *Timer/Counter0* za 1.
- `if (timer0PrazanHod == 25 ){}` - uvjetovani blok naredbi koji se izvodi svaki 25. poziv prekidne rutine sklopa *Timer/Counter0* jer je 25 * 16 ms = 400 ms. Unutar uvjetovanog `if` bloka dodajte sljedeće naredbe:
  - `timer0PrazanHod = 0;` - resetiranje brojača poziva prekidne rutine sklopa *Timer/Counter0* kako bi se ponovno izvršio ciklus od 25 * 16 ms = 400 ms.
  - `digitalToggle(B1);` - funkcija koja mijenja stanje crvene LED diode spojene na pin PB1.

U prekidnu rutinu `ISR(TIMER1_OVF_vect)` dodajte sljedeće naredbe:

- `timer1SetValue(40536);` - postavljanje početne vrijednosti registra `TCNT1` na vrijednost izračunatu relacijom (9.21) kako bi se prekid sklopa *Timer/Counter1* dogodio svakih 400 ms.
- `digitalToggle(B2);` - funkcija koja mijenja stanje žute LED diode spojene na pin PB2.

Za prekidnu rutinu `ISR(TIMER2_OVF_vect)` deklarirajte i inicijalizirajte varijablu kojom će se brojati broj poziva prekidnu rutinu tako da iznad prekidne rutine za sklop *Timer/Counter2* upišete naredbu `volatile uint8_t timer2PrazanHod = 0;`.

U prekidnu rutinu `ISR(TIMER2_OVF_vect)` dodajte sljedeće naredbe:

- `timer2SetValue(6);` - postavljanje početne vrijednosti registra `TCNT2` na vrijednost izračunatu relacijom (9.19) kako bi se prekid sklopa *Timer/Counter2* dogodio svakih 16 ms.
- `timer2PrazanHod++;` - povećanje broj poziva prekidne rutine sklopa *Timer/Counter2* za 1.
- `if (timer2PrazanHod == 25 ){}` - uvjetovani blok naredbi koji se izvodi svaki 25. poziv prekidne rutine sklopa *Timer/Counter2* jer je 25 * 16 ms = 400 ms. Unutar uvjetovanog `if` bloka dodajte sljedeće naredbe:
  - `timer2PrazanHod = 0;` - resetiranje brojača poziva prekidne rutine sklopa *Timer/Counter2* kako bi se ponovno izvršio ciklus od 25 * 16 ms = 400 ms.
  - `digitalToggle(B3);` - funkcija koja mijenja stanje zelene LED diode spojene na pin PB3.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba94.cpp` treba biti ista kao programski kod 9.27.

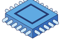
Programski kod 9.27: Program kojim se ispisuje temperatura te mijenja stanje crvene, žute i zelene LED diode svakih 400 ms u strogo zajamčenom vremenu

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

volatile uint8_t timer0PrazanHod = 0;

// prekidna rutina za timer 0
ISR(TIMER0_OVF_vect) {
    timer0SetValue(6); // početna vrijednost TCNT0 za 16 ms
    timer0PrazanHod++; // povećaj broj poziva za 1
    if (timer0PrazanHod == 25)  { // ako je timer0PrazanHod = 25*16ms=400ms
        timer0PrazanHod = 0; // vrati broj poziva na 0
        digitalToggle(B1); // promjena stanja na crvene LED diode
    }
}

// prekidna rutina za timer 1
ISR(TIMER1_OVF_vect) {
    timer1SetValue(40536); // početna vrijednost TCNT1 za 400 ms
    digitalToggle(B2); // promjena stanja na žute LED diode
}

volatile uint8_t timer2PrazanHod = 0;

// prekidna rutina za timer 2
ISR(TIMER2_OVF_vect) {
    timer2SetValue(6); // početna vrijednost TCNT2 za 8 ms
    timer2PrazanHod++; // povećaj broj poziva za 1
    if (timer2PrazanHod == 25)  { // ako je timer2PrazanHod = 25*16ms=400ms
        timer2PrazanHod = 0; // vrati broj poziva na 0
        digitalToggle(B3); // promjena stanja na zelene LED diode
```

```
    }
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B1, OUTPUT);
    // PB2 konfiguriran kao izlazni pin (žute LED dioda)
    pinMode(B2, OUTPUT);
    // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    pinMode(B3, OUTPUT);
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    timer0NormalMode();
    timer1NormalMode();
    timer2NormalMode();
    // djelitelji frekvencije
    timer0SetPrescaler(TIMER0_PRESCALER_1024); // F_CPU / 1024
    timer1SetPrescaler(TIMER1_PRESCALER_256); //  F_CPU / 256
    timer2SetPrescaler(TIMER2_PRESCALER_1024); //  F_CPU / 1024
    timer0InterruptOVFEnable(); // omogućenje prekida preljevom za timer0
    timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
    timer2InterruptOVFEnable(); // omogućenje prekida preljevom za timer2
    timer0SetValue(6); // početna vrijednost TCNT0 za 16 ms
    timer1SetValue(40536); // početna vrijednost TCNT1 za 400 ms
    timer2SetValue(6); // početna vrijednost TCNT2 za 16 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba94.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se temperatura okoline razvojnog okruženja, a crvena, žuta i zelena LED dioda izmjenjivati će svoje stanje svakih 400 ms u strogo zajamčenom vremenu.

Zatvorite datoteku `vjezba94.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 9.5

Napišite program kojim će se pomoću temperaturnog senzora LM35 uzimati uzorak temperature svakih 1000 ms u strogo zajamčenom vremenu pomoću sklopa *Timer/Counter1*. Temperaturu je potrebno ispisivati u °C na LCD displeju samo onda kada je dostupan novi uzorak temperature. Temperaturni senzor LM35 je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 1 i 2. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba95.cpp`. Omogućite prevođenje datoteke `vjezba95.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba95.cpp` prikazan je programskim kodom 9.28.

Programski kod 9.28: Početni sadržaj datoteke `vjezba95.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"

// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        T = readTempLM35(LM35); // čitaj temperaturu na LM35
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("T = %.2f%c", T, 223);
        _delay_ms(1000);
    }
}
```

Uzimanje uzoraka u strogo zajamčenom vremenom često se susreće u sustavima automatskog upravljanja gdje je takav postupak nužan. S druge strane, prezentacija mjerenih podataka (prikaz na displeju, logiranje podataka u datoteku i drugo) često zahtjeva vremensko uzorkovanje. U dosadašnjim je vježbama ispis temperature na LCD displeju bio vremenski određen funkcijom kašnjenja koja je program zakasnila za 1000 ms. Već smo spomenuli da funkcije kašnjenja nisu najbolji i prvi izbor pri programiranju mikroupravljača kada je potrebno izvesti složeniji projekt. U ovoj vježbi ćemo pokazati kako iskoristiti tajmer u svrhu uzimanja uzoraka temperature u strogo zajamčenom vremenu svakih 1000 ms pomoću sklopa *Timer/Counter1*. Provedimo sada konfiguraciju sklopa *Timer/Counter1*.

Djelitelj frekvencije za sklop *Timer/Counter1* neka prema iskustvu iz četvrte vježbe ima iznos 256. Taj djelitelj frekvencije uvrstimo u relaciju (9.6):

$$TCNT1_0 = 65536 - 1,0 \cdot \frac{16000000}{256} = 3036. \qquad (9.22)$$

Početna vrijednost registra `TCNT1` kojom će se ostvariti poziv prekidne rutine svakih 1000 ms iznosi 3036.

Konfiguraciju sklopa *Timer/Counter1* potrebno je provesti unutar inicijalizacijske funkcije `init()` tako da napišete sljedeće naredbe:

- `timer1NormalMode();` - konfiguriranje sklopa *Timer/Counter1* za normalan način rada,
- `timer1SetPrescaler(TIMER1_PRESCALER_256);` - djelitelj frekvencije sklopa *Timer/Counter1* namješten na iznos 256,

- `timer1InterruptOVFEnable();` - omogućenje prekida za sklop *Timer/Counter1*,
- `timer1SetValue(3036);` - inicijalizacija registra `TCNT1` na iznos 3036 za vrijeme 1000 ms.

Uzimanje uzoraka svakih 1000 ms pomoću sklopa *Timer/Counter1* moguće je provesti na sljedeća dva načina:

1. mjerenjem temperature pomoću funkcije `readTempLM35(LM35)` unutar prekidne rutine. Ispis temperature na LCD displej moguće je provesti ili u prekidnoj rutini ili u glavnom programu.
2. korištenjem *Boolean* varijable koja se postavlja u vrijednost `true` svaki puta kada se pozove prekidna rutina (svakih 1000 ms). Ova varijabla može biti uvjet u glavnom programu za naredbu grananja `if` unutar koje će se provesti mjerenje temperature pomoću funkcije `readTempLM35(LM35)` te ispis temperature na LCD displej.

Opredijelit ćemo se za drugi pristup. Razlog tome jest taj da i AD pretvorba koja se izvršava pri mjerenju temperature i ispis na LCD displej unose kašnjenje u prekidnu rutinu, a kašnjenje se u prekidnim rutinama preporučuje strogo izbjegavati. *Boolean* varijablu koju ćemo postavljati u vrijednost `true` unutar prekidne rutine `ISR(TIMER1_OVF_vect)` deklarirat ćemo kao globalnu tako da u programski kod 9.28 dodamo sljedeću naredbu: `volatile bool uzmiUzorak = false;`. Primijetite da smo globalnu varijablu koja se mijenja u prekidnoj rutini deklarirali na način da smo tipu podatka dodali ključnu riječ `volatile`. Kako smo već spomenuli u prethodnom poglavlju, ključna riječ `volatile` ima za cilj spriječiti primjenu optimizacije programskog koda nad objektima koji se mogu promijeniti na načine koje prevoditelj ne može utvrditi.

U prekidnu rutinu `ISR(TIMER1_OVF_vect)` dodajte sljedeće naredbe:

- `timer1SetValue(3036);` - postavljanje početne vrijednosti registra `TCNT1` na vrijednost izračunatu relacijom (9.22) kako bi se prekid sklopa *Timer/Counter1* dogodio svakih 1000 ms.
- `uzmiUzorak = true;` - varijabla koja će omogućiti mjerenje temperature i njen prikaz na LCD displeju jednom u 1000 ms.

Ostaje nam još realizirati mjerenje temperature i njen prikaz na LCD displeju jednom u 1000 ms. U programskom kodu 9.28 potrebno je obrisati poziv funkcije `_delay_ms(1000);` u beskonačnoj `while` petlji. Preostali dio programskog koda u beskonačnoj `while` petlji potrebno je uvjetno izvoditi na sljedeći način:

- otvorite uvjetovani blok `if (uzmiUzorak) { }`. Ovim blokom provjerava se da li je varijabla `uzmiUzorak` poprimila vrijednost `true` čime se osigurava izvođenje uvjetovanog bloka jednom u sekundi u strogo zajamčenom vremenu.
- unutar uvjetovanog bloka kopirajte prethodni sadržaj beskonačne `while` petlje.
- na kraju uvjetovanog bloka dodajte naredbu `uzmiUzorak = false;` kako bi se onemogućilo mjerenje temperature i ispis na LCD displej do ponovnog poziva prekidne rutine.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba95.cpp` treba biti ista kao programski kod 9.29.

Programski kod 9.29: Program kojim se uzima uzorak temperature te je ispisuje na LCD displej svakih 1000 ms u strogo zajamčenom vremenu

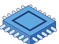
```
#include "AVR/avrlib.h"
```

```
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Senzori/lm35.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
// konstanta koja se koristi umjesto ADC0
#define LM35 ADC0

volatile bool uzmiUzorak = false;

// prekidna rutina za timer 1
ISR(TIMER1_OVF_vect) {
    timer1SetValue(3036); // početna vrijednost TCNT1 za 1000 ms
    uzmiUzorak = true; // omogući mjerenje temperature
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    interruptEnable(); // globalno omogućenje prekida
    // postavljanje normalnog načina rada
    timer1NormalMode();
    // djelitelj frekvencije F_CPU / 256
    timer1SetPrescaler(TIMER1_PRESCALER_256);
    timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
    timer1SetValue(3036); // početna vrijednost TCNT1 za 1000 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float T; // temperatura u okolini senzora LM35
    while (1) {
        if (uzmiUzorak) {
            T = readTempLM35(LM35); // čitaj temperaturu na LM35
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("T = %.2f%c", T, 223);
            // onemogući mjerenje temperature sljedećih 1000 ms
            uzmiUzorak = false; // onemogući mjerenje temperature
        }
    }
}
```

Prevedite datoteku `vjezba95.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se temperatura okoline razvojnog okruženja svakih 1000 ms u strogo zajamčenom vremenu.

Izbrišite ključnu riječ `volatile` u deklaraciji `volatile bool uzmiUzorak = false;`. Prevedite datoteku `vjezba95.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Primijetite da LCD ništa ne ispisuje. Razlog tomu jest taj što je prevoditelj pri optimiranju programskog koda pretpostavio da se uvjetovani blok `if (uzmiUzorak) { }` nikad neće izvesti jer je varijabla `uzmiUzorak` inicijalizirana na vrijednost `false`. S obzirom da varijablu `uzmiUzorak` mijenja prekidna rutina koja se ne poziva iz glavnog programa, već je poziva hardver, prevoditelj će pretpostaviti da se varijabla `uzmiUzorak` nikada ne mijenja tijekom izvođenja programskog koda.

Zatvorite datoteku `vjezba95.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 9.6

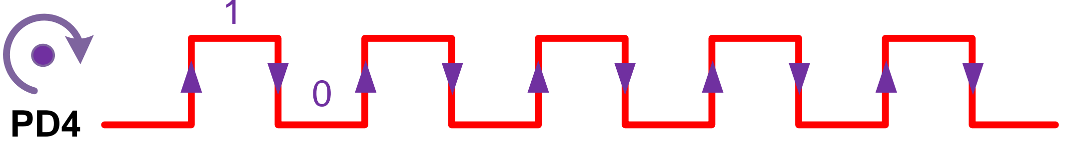
Napišite program kojim će brojati impulsi s rotacijskog enkodera čiji je jedan kanal spojen na pin PD4 (T0) mikroupravljača ATmega328P. Program mora ispisivati broj impulsa na LCD displej samo onda kada se promijeni broj impulsa. Pretpostavimo da mogući broj impulsa tijekom jednog uključenja razvojnog okruženja neće premašiti 2000000. Rotacijski enkoder zahtjeva postavljanje kratkospojnika JP2 između trnova 2 i 3 (`ENK`). Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba96.cpp`. Omogućite prevođenje datoteke `vjezba96.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba96.cpp` prikazan je programskim kodom 9.30.

Programski kod 9.30: Početni sadržaj datoteke `vjezba96.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

// prekidna rutina za brojac 0
ISR(TIMER0_OVF_vect) {

}

void init() {
    lcdInit(); // inicijalizacija LCD displeja

}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    while (1) {

    }
}
```

U ovoj vježbi cilj nam je prikazati rad sklopa *Timer/Counter0* kao brojač koji broji impulse iz vanjskoj izvora. Sklop *Timer/Counter0* konfiguriran kao brojač može brojati rastuće ili padajuće bridove signala na pinu T0 (PD4). Niz impulsa rotacijskog enkodera čiji je jedan kanal spojen na digitalni ulaz PD4 prikazan je na slici 9.4.



Slika 9.4: Niz impulsa rotacijskog enkodera čiji je jedan kanal spojen na digitalni ulaz PD4 (T0)

Pin PD4 ima alternativnu namjenu kojom se mogu brojati impulsi pomoću sklopa *Timer/Counter0* iz vanjskog izvora. U dosadašnjim vježbama pin PD4 smo koristili kao digitalni ulaz na koji je spojeno tipkalo T1 (vidi shemu na slici 5.2). U ovoj vježbi potrebno je kratkospojnik JP2 postaviti između trnova 2 i 3 (`ENK`) na razvojnom okruženju s mikroupravljačem ATmega328P. Impulse na pinu PD4 moguće je generirati i pomoću tipkala T1, no zbog istitravanja tipkala često jedan pritisak na tipkalo može povećati brojač impulsa za

nasumični broj (1 ili 2 ili 3 ili 4 ili ...).

Da bi sklop *Timer/Counter0* mogao brojati impulse iz vanjskog izvora potrebno ga je konfigurirati u normalnom načinu rada. Izvor impulsa odabire se u registru `TCCR0B` pomoću bitova `CS00`, `CS01` i `CS02`, a prema tablici 9.1. Na primjer, za povećanje vrijednosti u registru `TCNT0` za 1 na padajući brid signala na digitalnom ulazu PD4 bit `CS00` mora imati vrijednost 0, bit `CS01` vrijednost 1, a bit `CS02` vrijednost 1. Vrijednosti koje mogu biti pohranjene u registar `TCNT0` kreću se od 0 do 255. Naš zadatak je omogućenje brojanja impulsa do vrijednosti najmanje 2000000. Jedini način da to uspijemo jest da iskoristimo prekidnu rutinu sklopa *Timer/Counter0* kojom ćemo brojati koliko puta je brojač izbrojio 256 impulsa prema relaciji:

$$\texttt{brojImpulsa} = 256 \cdot \texttt{brojPrekida} + TCNT0. \tag{9.23}$$

S obzirom da ćemo koristiti prekidnu rutinu sklopa *Timer/Counter0*, potrebno je omogućiti prekide za ovaj sklop. Početna vrijednost registra `TCNT0` mora biti 0. Konfiguraciju sklopa *Timer/Counter0* potrebno je provesti unutar inicijalizacijske funkcije `init()` tako da napišete sljedeće naredbe:

- `timer0NormalMode();` - konfiguriranje sklopa *Timer/Counter0* za normalan način rada,
- `timer0SetPrescaler(TIMER0_EXTERNAL_FALL_EDGE);` - konfiguracija vanjskog izvora impulsa na digitalnom ulazu PD4 (T0). Registar `TCNT0` uvećat će se za 1 na svaki padajući brid impulsa signala na digitalnom ulazu PD4 (T0).
- `timer0InterruptOVFEnable();` - omogućenje prekida za sklop *Timer/Counter0* kako bismo mogli izbrojiti više od 256 impulsa,
- `timer0SetValue(0);` - inicijalizacija registra `TCNT0` na iznos 0. Ova se naredba može preskočiti s obzirom da su početne vrijednosti registara jednake 0.

Nakon što smo konfigurirali sklop *Timer/Counter0*, potrebno je osigurati dohvat broja impulsa te prezentaciju na LCD displeju. S obzirom da nam je cilj ispisivati broj impulsa samo onda kada se njegova vrijednost promijeni, ispod poziva funkcije `init()` u programskom kodu 9.30 deklarirajte sljedeće varijable:

- `int brojImpulsa = 0;` - deklaracija i inicijalizacija cjelobrojne varijable u koju će se spremati broj padajućih bridova signala na digitalnom ulazu PD4.
- `int brojImpulsaOld = 0;` - deklaracija i inicijalizacija cjelobrojne varijable u koju će se spremati stara vrijednost broja padajućih bridova signala na digitalnom ulazu PD4.

Kako bi LCD diplej prikazivao početnu vrijednost broja impulsa (0) prije nego što dođe do promjene broja impulsa, prije beskonačne petlje napišite sljedeće naredbe:

- `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
- `lcdprintf("BrImp = %d", brojImpulsa);` - ispis teksta `BrImp = 0` jer početna vrijednost varijable `brojImpulsa` iznosi 0.

U beskonačnu petlju `while` napišite sljedeće naredbe:

- `brojImpulsa = brojPrekida * 256 + timer0GetValue();` - izračunavanje broja padajućih bridova signala na digitalnom ulazu PD4 prema relaciji (9.23). Za dohvat vrijednosti registra `TCNT0` u kojem se broje padajući bridovi signala na digitalnom ulazu PD4 korištena je funkcija `timer0GetValue()`. Umjesto ove funkcije mogli smo navesti ime

registra `TCNT0`.

- `if ((brojImpulsa != brojImpulsaOld)) { }` - uvjetovani blok kojim se provjerava da li je novi broj padajućih bridova signala na digitalnom ulazu PD4 različit od starog broja. Ako je došlo do promjene, potrebno je provesti ispis na LCD displej. Unutar uvjetovanog bloka napišite sljedeće naredbe:
  - `lcdClrScr();` - brisanje prethodnog teksta na LCD displeju i postavljanje kursora na početak reda.
  - `lcdprintf("BrImp = %d", brojImpulsa);` - ispis teksta `BrImp =` te broj padajućih bridova signala na digitalnom ulazu PD4 koji se nalazi u varijabli `brojImpulsa`.
  - `brojImpulsaOld = brojImpulsa;` - spremanje aktualnog broja padajućih bridova signala na digitalnom ulazu PD4 u staru vrijednost kako bi se blokirao ispis na LCD displej do promjene broja impulsa.

Varijabla `brojPrekida` uvećavat će se unutar prekidne rutine sklopa *Timer/Counter0*. Tu varijablu potrebno je deklarirati kao globalnu te joj dodijeliti ključnu riječ `volatile` jer se mijenja unutar prekidne rutine. U globalnom području deklaracije varijable napišite naredbu `volatile uint16_t brojPrekida = 0;`. Ova varijabla inicijalizirana je na vrijednost 0. U prekidnu rutinu `ISR(TIMER0_OVF_vect)` dodajte sljedeću naredbu:

- `brojPrekida++;` - povećanje broja prekida za vrijednost 1 svaki puta kada se desio prekid, odnosno kada je registar `TCNT0` izbrojio 256 impulsa.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba96.cpp` treba biti ista kao programski kod 9.31.

Programski kod 9.31: Program kojim se ispisuje broj impulsa koji su generirani rotacijskim enkoderom na digitalnom ulazu PD4

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

volatile uint16_t brojPrekida = 0;
// prekidna rutina za brojač 0
ISR(TIMER0_OVF_vect) {
    brojPrekida++;
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    config_input(DDRD, PD4); // konfiguriranje ulaza PD4
    pull_up_on(PORTD, PD4); // pull up uključen na ulazu PD4
    interruptEnable(); // globalno omogućenje prekida
    // postaviti Normalna način rada
    timer0NormalMode();
    // djeliteli frekvencije F_CPU / 1024
    timer0SetPrescaler(TIMER0_EXTERNAL_FALL_EDGE);
    timer0InterruptOVFEnable(); // omogućenje prekida preljevom za timer0
    timer0SetValue(0); // početna vrijednost TCNT0
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    // aktualni i stari broj impulsa rotacijskog enkodera
```

```
    int brojImpulsa = 0;
    int brojImpulsaOld = 0;
    // brisanje znakova LCD displeja +  home pozicija kursora
    lcdClrScr();
    // ispis na LCD displej (sintaksa funkcije printf())
    lcdprintf("BrImp = %d", brojImpulsa);
    while (1) {
            //izračun ukupnog broja impulsa
        brojImpulsa = brojPrekida * 256 + timer0GetValue();
        if (brojImpulsa != brojImpulsaOld) {
            // brisanje znakova LCD displeja +  home pozicija kursora
            lcdClrScr();
            // ispis na LCD displej (sintaksa funkcije printf())
            lcdprintf("BrImp = %d", brojImpulsa);
            // postavljanje stare vrijednosti broja impulsa
            brojImpulsaOld = brojImpulsa;
        }
    }
}
```

Prevedite datoteku `vjezba96.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se broj impulsa koji su generirani rotacijskim enkoderom na digitalnom ulazu PD4.

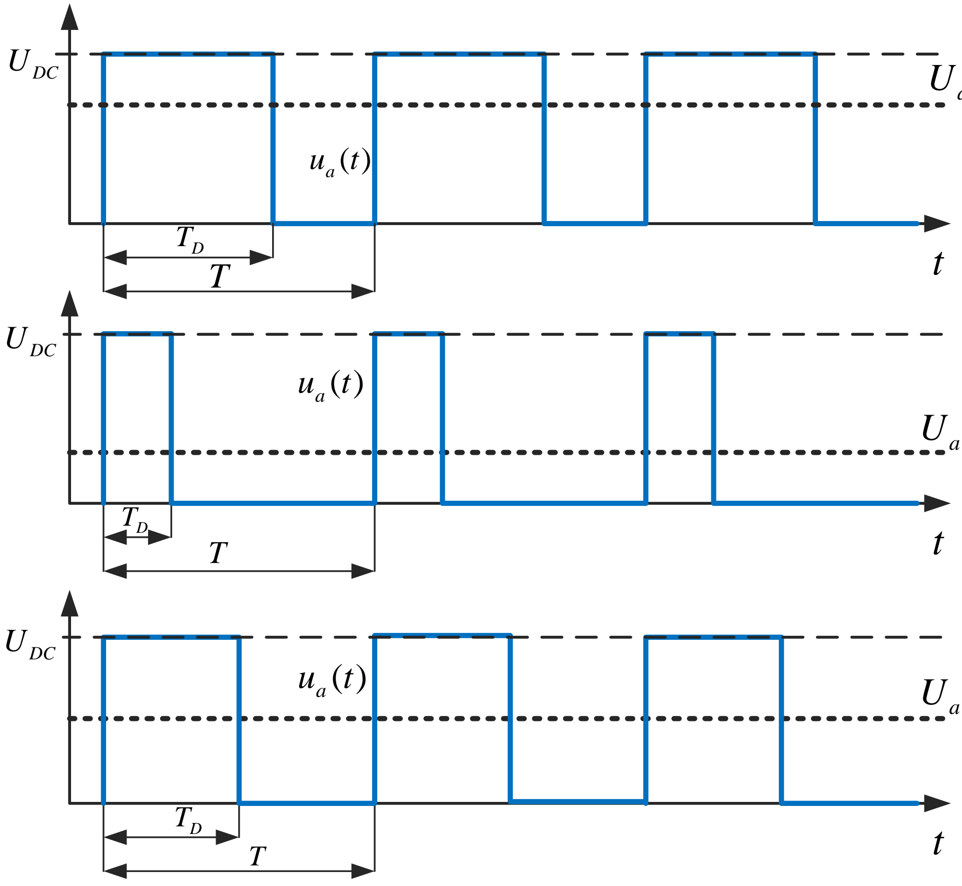
Na razvojnom okruženju s mikroupravljačem ATmega328P vratite kratkospojnik JP2 između trnova 1 (`TIPK`) i 2 kako bi se u sljedećim vježbama mogla koristiti tipkala. Testirajte sada prethodni program. Primijetit ćete da se pritiskom na tipkalo (padajući brid signala na digitalnom ulazu PD4) broj impulsa ne povećava uvijek za 1. Kako smo rekli, dolazi do istitravanja tipkala pa se svaki padajući dio titrajnog signala detektira kao padajući brid signala, odnosno pritisak na tipkalo.

Zatvorite datoteku `vjezba96.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 10

Pulsno širinska modulacija

U prethodnom poglavlju tajmere smo koristili za obavljanje vremenskih zadataka u strogo zajamčenom vremenu. Fokus ovog poglavlja bit će na korištenju tajmera u svrhu pulsno širinske modulacije (engl. *Pulse width modulation*), skraćeno PWM. PWM je tehnika modulacije signala kojom se omogućuje promjena prosječne snage električnog signala promjenom širine naponskog signala (slika 10.1). Modulacijom se širine impulsa digitalni signal pretvara u analogni.



Slika 10.1: Modulacija širine impulsa naponskog signala - PWM

PWM signal je periodičan signal, a može se koristiti za različite svrhe kao što je promjena brzine vrtnje DC motora, promjena intenziteta svjetla itd. Na slici 10.1 prikazan je PWM signal

koji je periodičan pravokutni signal visoke frekvencije koji se može na periodu T opisati relacijom:

$$u_a(t) = \begin{cases} U_{DC} & za \quad 0 \leq t < T_D \\ 0 & za \quad T_D \leq t < T \end{cases} \tag{10.1}$$

gdje su:

- U_{DC} - napon istosmjernog izvora napajanja, [V]
- T - period PWM signala, [s]
- T_D - vrijeme visokog stanja PWM signala, [s].

PWM signal na slici 10.1 jest neinvertirajući. U daljnjem tekstu pod PWM signalom podrazumijevat će se neinvertirajući PWM signal. Modulaciju širine impulsa objasnit ćemo na primjeru istosmjernog motora. Ako istosmjerni motor spojimo na aktuator koji generira PWM signal malog perioda (visoke frekvencije), struja istosmjernog motora, zbog induktiviteta armaturnog namota, ne može slijediti brze promjene napona armature $u_a(t)$ sa slike 10.1. Zbog toga se u ovom slučaju istosmjerni motor ponaša kao da je spojen na istosmjerni napon U_a, odnosno kroz armaturu istosmjernog motora teče istosmjerna struja. Napon U_a prikazan je na slici 10.1 isprekidanom linijom i jednak je srednjoj vrijednosti signala $u_a(t)$:

$$U_a = U_{DC}\frac{T_D}{T} \tag{10.2}$$

Srednja vrijednost napona armature U_a proporcionalna je s vremenom visokog stanja PWM signala T_D. Prema tome, promjenom vremena visokog stanja PWM signala T_D, mijenjamo i srednju vrijednost napona, a time i brzinu vrtnje istosmjernog motora [6].

Relacija (10.2) može se napisati i na sljedeći način:

$$U_a = U_{DC} \cdot D \tag{10.3}$$

gdje je $D = \frac{T_D}{T}$ faktor vođenja PWM signala (engl. *Duty Cycle*). Ako je $U_{DC} = 12$ V, a faktor vođenja $D = 30\%$, tada je srednja vrijednost napona jednaka $U_a = 3,6$ V.

Frekvencija PWM signala jednaka je:

$$f = \frac{1}{T} \tag{10.4}$$

Mikroupravljači pomoću tajmera mogu generirati PWM signal prikazan na slici 10.1 različite frekvencije i s različitim faktorom vođenja. PWM signal generira se na neki od digitalnih pinova sukladno tehničkim specifikacijama proizvođača. AVR mikroupravljači podržavaju dva PWM načina rada:

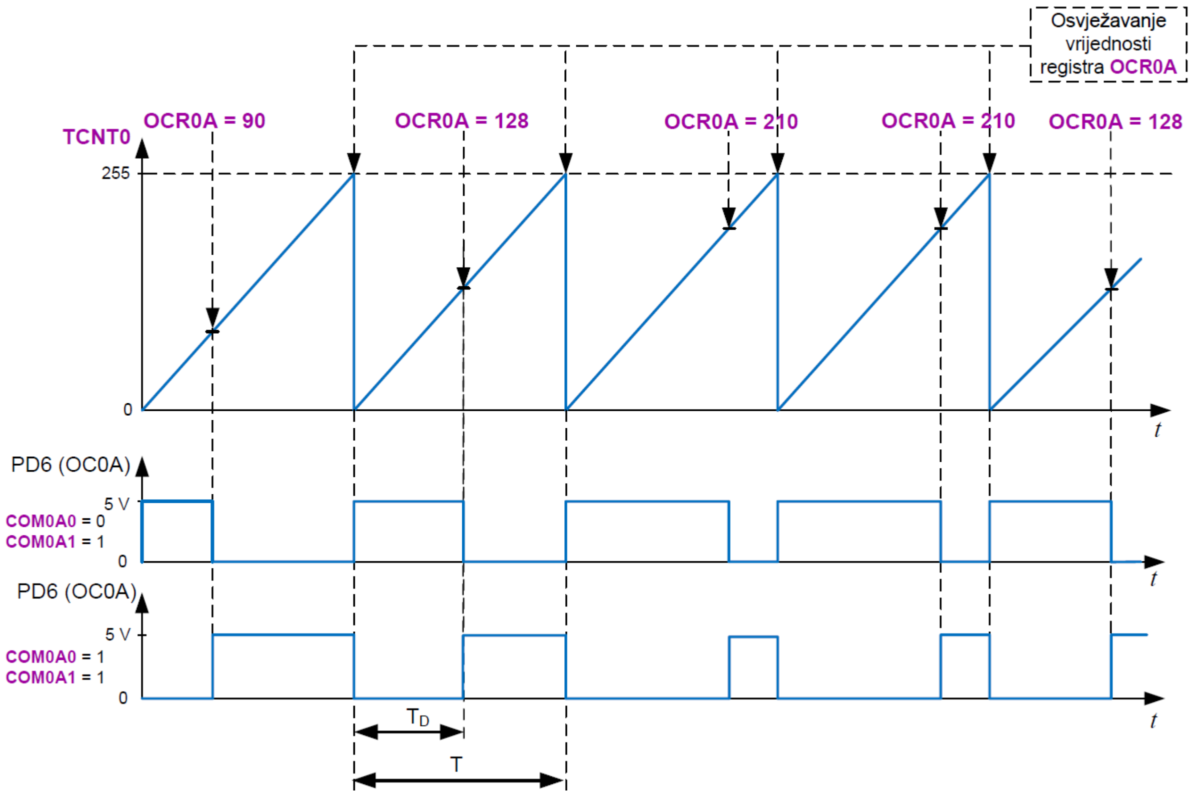
- *Fast* PWM način rada
- *Phase Correct* PWM način rada

Ove ćemo načine rada opisati u nastavku poglavlja. Mikroupravljač ATmega328P PWM signale može generirati pomoću sva tri tajmera na 6 digitalnih pinova. Svaki tajmer generira PWM signal na dva kanala: A i B kanal.

10.1 Vježbe - PWM načini rada tajmera

Fast PWM način rada tajmera

Fast PWM način rada objasnit ćemo na sklopu *Timer/Counter0* za kanal A. U *Fast* PWM načinu rada tajmera, PWM signal se generira na temeljem usporede vrijednosti u registru `TCNT0` s vrijednostima u registrima `OCR0A` (za kanal A) i `OCR0B` (za kanal B). Sklop *Timer/Counter0* generira PWM signal visoke frekvencije na pinu PD6 (OC0A), odnosno pinu PD5 (OC0B). Vremenski dijagram *Fast* PWM načina rada sklopa *Timer/Counter0* prikazan je na slici 10.2. Vrijednost registra `TCNT0` cijelo se vrijeme uvećava za 1 impulsima koji dolaze nakon djelitelja frekvencije radnog takta i kreće se od 0 (DNO) do 255 (VRH). Nakon što registar `TCNT0` postigne vrijednost 255, dolazi do preljeva te ponovno broji od 0 do 255. U registar `OCR0A` možemo upisati vrijednosti u rasponu od [0, 255]. Vrijednost u registru `OCR0A` osvježava se svaki put kada vrijednost registra `TCNT0` prelazi iz 255 u 0 (kad se dogodi preljev).



Slika 10.2: Vremenski dijagram *Fast* PWM načina rada sklopa *Timer/Counter0*

Vrijednosti registara `TCNT0` i `OCR0A` cijelo se vrijeme uspoređuju. U trenutku kada se vrijednosti ovih dvaju registara izjednače (`TCNT0` == `OCR0A`) stanja signala na pinu PD6 (OC0A) mijenja se iz visokog u nisko ili obratno, što ovisi o konfiguraciji bitova `COM0A1` i `COM0A0`. Ovisno o stanju bitova `COM0A1` i `COM0A0`, prema tablici 10.1 na pinu PD6 (OC0A) generira se PWM signal prikazan na slici 10.2.

Ako je konfiguracija bitova `COM0A1` = 1 i `COM0A0` = 0, tada se pin PD6 (OC0A) postavlja u nisko stanje kada je vrijednost registra `TCNT0` jednaka vrijednosti registra `OCR0A`, a u visoko stanje kada vrijednost registra `TCNT0` prijeđe iz 255 u 0 (preljev registra `TCNT0`). Ovakva vrsta konfiguracije generira neinvertirajući PWM signal (slika 10.2).

Ako je konfiguracija bitova COM0A1 = 1 i COM0A0 = 1, tada se pin PD6 (OC0A) postavlja u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0, a u nisko stanje kada vrijednost registra TCNT0 prijeđe iz 255 u 0 (preljev registra TCNT0). Ovakva vrsta konfiguracije generira invertirajući PWM signal (slika 10.2) [1].

Tablica 10.1: Postavke bitova COM0A1 i COM0A0 u *Fast* PWM načina rada (kanal A) [2]

COM0A1	COM0A0	**Postavke *Fast* PWM načina rada**
0	0	Pin PD6 (OC0A) isključen
0	1	Ako je WGM02 = 0, tada je pin PD6 (OC0A) isključen. Ako je WGM02 = 1, promjena stanja pina PD6 (OC0A) događa se kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A.
1	0	Pin PD6 (OC0A) postavlja se u nisko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A. Pin PD6 (OC0A) postavlja se u visoko stanje kada vrijednost registra TCNT0 prijeđe iz 255 (VRH) u 0 (DNO).
1	1	PD6 (OC0A) postavlja se u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A. PD6 (OC0A) postavlja se u nisko stanje kada vrijednost registra TCNT0 prijeđe iz 255 (VRH) u 0 (DNO).

Na blokovskoj shemi prikazanoj na slici 9.1 nalazi se dio sklopa *Timer/Counter0* koji provjerava da li su vrijednosti registara TCNT0 i OCR0A jednake te dio koji generira PWM signal na pinu PD6 (OC0A). PWM signal može se generirati i na pinu PD5 (OC0B, kanal B) pomoću sklopa *Timer/Counter0*, no tada je potrebno pogledati konfiguraciju bitova COM0B1 i COM0B0 u literaturi [2] u tablici 14-6. PWM signal mogu generirati i ostala dva sklopa: *Timer/Counter1* i *Timer/Counter2*. Konfiguraciju PWM sklopova *Timer/Counter1* i *Timer/Counter2* možete pronaći u literaturi [2] u tablicama 15-1 do 15-6 i 16-1 do 16-6.

Frekvencija PWM signala u *Fast* PWM načinu rada za sklop *Timer/Counter0* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER0 \cdot (VRH0 + 1)}. \tag{10.5}$$

Vrijednosti varijable $PRESCALER0$ su: 1, 8, 64, 256 ili 1024. Iznos varijable $VRH0$ jest 255 ili vrijednost zapisana u registru OCR0A, ovisno o konfiguraciji sklopa *Timer/Counter0*.

Frekvencija PWM signala u *Fast* PWM načinu rada za sklop *Timer/Counter1* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER1 \cdot (VRH1 + 1)}. \tag{10.6}$$

Vrijednosti varijable $PRESCALER1$ su: 1, 8, 64, 256 ili 1024. Iznos varijable $VRH1$ može biti 255, 511, 1023 ili vrijednost zapisana u registre ICR1 ili OCR1A, ovisno o konfiguraciji sklopa *Timer/Counter1*.

Frekvencija PWM signala u *Fast* PWM načinu rada za sklop *Timer/Counter2* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER2 \cdot (VRH2 + 1)}. \tag{10.7}$$

Vrijednosti varijable $PRESCALER2$ su: 1, 8, 32, 64, 128, 256 ili 1024. Iznos varijable $VRH2$ jest 255 ili vrijednost zapisana u registru OCR2A, ovisno o konfiguraciji sklopa *Timer/Counter2*.

Širina impulsa PWM signala na kanalu A određena je vremenom visokog stanja T_D u odnosu na vrijeme perioda T (slika 10.2) prema relaciji:

$$\frac{T_D}{T} = D = \frac{\texttt{OCR0A}}{VRH0}. \quad (10.8)$$

Omjer $\frac{T_D}{T} = D$ naziva se faktor vođenja (engl. *duty cycle*) i izražava se u % s rasponom vrijednosti [0, 100]%. Varijabla $VRH0$ uobičajeno ima iznos 255 za sklop *Timer/Counter0*. Širinu PWM signala na pinu PD6 (OC0A) mijenjamo promjenom vrijednosti registra `OCR0A` prema relaciji ($VRH0 = 255$):

$$\texttt{OCR0A} = \frac{T_D}{T} \cdot VRH0 = \frac{T_D}{T} \cdot 255 = D \cdot 255. \quad (10.9)$$

Slučajevi u kojima registar `OCR0A` poprima vrijednost 0 (DNO) ili 255 (VRH) specijalni su slučajevi. Kod neinvertirajućeg PWM signala (`COM0A1` $= 1$, `COM0A0` $= 0$) za specijalne slučajeve vrijedi:

- ako je `OCR0A` $= 255$, tada je pin PD6 (OC0A) konstantno u visokom stanju,
- ako je `OCR0A` $= 0$, tada će se se na pinu PD6 (OC0A) generirati uski šiljak.

Na primjer, ako upravljamo intezitetom LED diode u slučaju kada je `OCR0A` $= 0$, LED dioda će sjajiti najmanjim mogućim intezitetom zbog uskog šiljka PWM signala iako bismo očekivali da će LED dioda biti isključena. Ovo svakako spada u nedostatak *Fast* PWM načina rada. Taj nedostatak nema *Phase Correct* PWM način rada koji će biti opisan u nastavku [1].

Ako se koristi kanal B (uz `WGM02` $= 0$), tada širinu PWM signala na pinu PD5 (OC0B) mijenjamo promjenom vrijednosti registra `OCR0B` prema relaciji ($VRH0 = 255$):

$$\texttt{OCR0B} = \frac{T_D}{T} \cdot VRH0 = \frac{T_D}{T} \cdot 255 = D \cdot 255. \quad (10.10)$$

Pretpostavimo da PWM signal koristimo u silaznom pretvaraču za koji vrijedi da se istosmjerni napon U_{DC} (npr. $U_{DC} = 48$ V) pretvara u raspon od $[0, U_{DC}]$ V. PWM signal na pinu PD6 (OC0A) upravlja radom sklopke u izvedbi unipolarnog tranzistora. Označimo izlazni napon na silaznom pretvaraču U_i. Izlazni napon može se izračunati prema relaciji:

$$U_i = D \cdot U_{DC} = \frac{\texttt{OCR0A}}{255} \cdot U_{DC}. \quad (10.11)$$

Napon na izlazu iz silaznog pretvarača mijenjamo promjenom vrijednosti registra `OCR0A` prema relaciji:

$$\texttt{OCR0A} = \frac{U_i}{U_{DC}} \cdot 255. \quad (10.12)$$

U programskom kodu 10.1 prikazana je konfiguracija tajmera za *Fast* PWM način rada.

Programski kod 10.1: Konfiguracija sklopa *Timer/Counter0* za *Fast* PWM način rada

```
TCCR0B |= (0 << CS02) | (1 << CS01) | (0 << CS00);  //F_CPU/8
TCCR0A |= (1 << WGM01) | (1 << WGM00);  // Fast PWM način rada
TCCR0B |= (0 << WGM02);  // VRH je 0x0FF
TCCR0A |= (1 << COM0A1) | (0 << COM0A0); // Neinvertirajući PWM
OCR0A = 150;
```

Naredbom `TCCR0B |= (0 << CS02)| (1 << CS01)| (0 << CS00);` frekvenciju radnog takta podijelili smo s 8, prema tablici 9.1. Iznos varijable $VRH0$ jest 255. Frekvencija PWM signala prema relaciji 10.5 jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER0 \cdot (VRH0 + 1)} = \frac{16000000}{8 \cdot 256} = 7812,5Hz. \tag{10.13}$$

Fast PWM način rada tajmera odabran je naredbom `TCCR0A |= (1 << WGM01)| (1 << WGM00);` prema tablici 9.2. Neinvertirajući način rada odabran je naredbom `TCCR0A |= (1 << COM0A1)| (0 << COM0A0);`. U tom načinu rada vrijednost registra `OCR0A` postavljena je na 150. Ukoliko je ovo konfiguracija PWM signala koji upravlja silaznim pretvaračem koji je prethodno spomenut, izlazni napon za konfiguraciju prikazanu programskim kodom 10.1 može se izračunati prema relaciji (10.11):

$$U_i = \frac{\texttt{OCR0A}}{255} \cdot U_{DC} = \frac{150}{255} \cdot 48 = 28.235V. \tag{10.14}$$

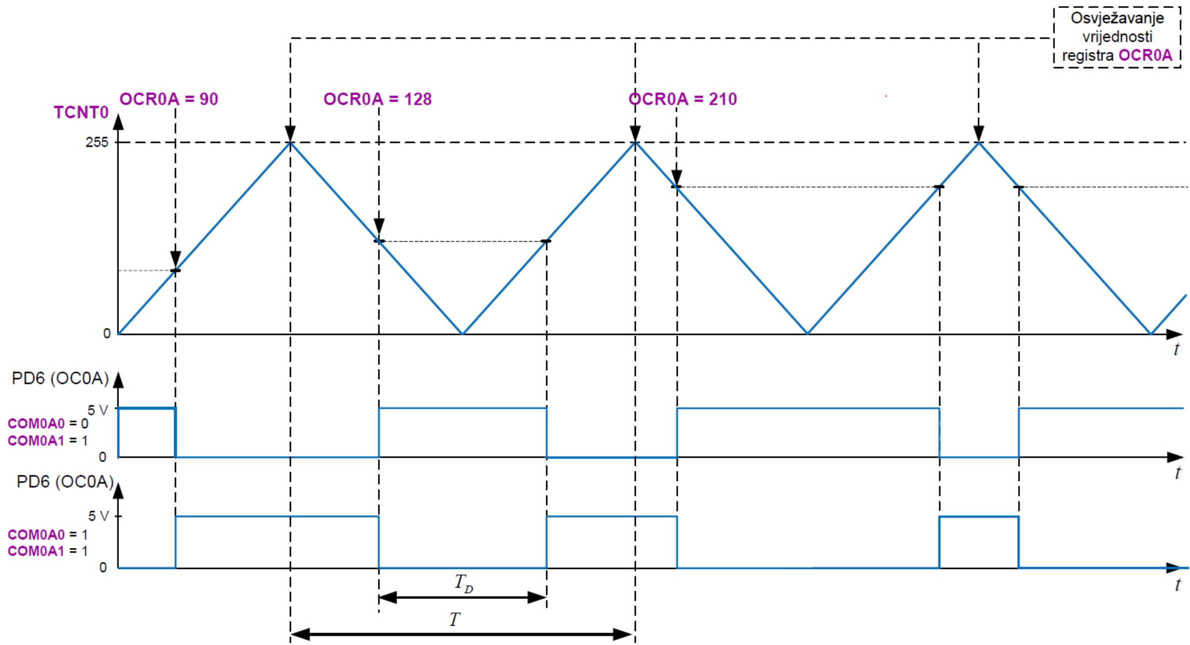
Ako na izlazu želimo postići vrijednost izlaznog napona 12 V, tada je u registar `OCR0A` potrebno upisati vrijednost dobivenu relacijom (10.12):

$$\texttt{OCR0A} = \frac{U_i}{U_{DC}} \cdot 255 = \frac{12}{48} \cdot 255 = 63. \tag{10.15}$$

Za sklopove *Timer/Counter1* i *Timer/Counter2* konfiguracija *Fast* PWM načina rada slična je prethodno provedenoj konfiguraciji sklopa *Timer/Counter0*. Za sve detalje o konfiguraciji sklopova *Timer/Counter1* i *Timer/Counter2* pogledajte literaturu [2].

Phase Correct PWM način rada tajmera

Drugi način na koji se može generirati PWM signal jest konfiguracija sklopa *Timer/Counter0* u *Phase Correct* PWM načinu rada. Kao i u prethodnom PWM načinu rada i ovaj ćemo objasniti na primjeru kanala A. *Phase Correct* PWM način rada ima dvostruko manju frekvenciju od *Fast* PWM načinu rada. Kod *Phase Correct* PWM načina rada registar `TCNT0` uvećava svoju vrijednost za 1 od 0 (DNO) do 255 (VRH), a kada dođe do vrijednosti 255 počinje brojati unazad, odnosno smanjuje vrijednost registra `TCNT0` za 1. Na temelju vrijednosti u registrima `TCNT0` i `OCR0A` (kanal A), odnosno `OCR0B` (kanal B) generira se PWM signal na pinu PD6 (OC0A), odnosno pinu PD5 (OC0B) (vidjeti sliku 10.3). U registar `OCR0A` možemo upisati vrijednosti u rasponu od [0, 255]. Vrijednost u registru `OCR0A` osvježava se svaki put kada vrijednost registra `TCNT0` dosegne 255 (vidjeti sliku 10.3).



Slika 10.3: Vremenski dijagram *Phase Correct* PWM načina rada

Vrijednosti registara TCNT0 i OCR0A cijelo se vrijeme uspoređuju. U trenutku kada se vrijednosti ovih dvaju registara izjednače (TCNT0 == OCR0A) stanja signala na pinu PD6 (OC0A) mijenja se iz visokog u nisko ili obratno, što ovisi o konfiguraciji bitova COM0A1 i COM0A0. Ovisno o stanju bitova COM0A1 i COM0A0, prema tablici 10.1 na pinu PD6 (OC0A) generira se PWM signal prikazan na slici 10.3.

Tablica 10.2: Postavke bitova COM0A1 i COM0A0 u *Phase Correct* PWM načina rada (kanal A) [2]

COM0A1	COM0A0	**Postavke *Phase Correct* PWM načina rada**
0	0	Pin PD6 (OC0A) isključen
0	1	Ako je WGM02 = 0, tada je pin PD6 (OC0A) isključen. Ako je WGM02 = 1, promjena stanja pina PD6 (OC0A) događa se kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A.
1	0	Pin PD6 (OC0A) postavlja se u nisko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema gore. Pin PD6 (OC0A) postavlja se u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema dolje.
1	1	Pin PD6 (OC0A) postavlja se u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema gore. Pin PD6 (OC0A) postavlja se u nisko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema dolje.

Ako je konfiguracija bitova COM0A1 = 1 i COM0A0 = 0, tada se pin PD6 (OC0A) postavlja u nisko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema gore, a u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema dolje. Ovakva vrsta konfiguracije generira neinvertirajući PWM signal (slika 10.3).

Ako je konfiguracija bitova COM0A1 = 1 i COM0A0 = 1, tada se pin PD6 (OC0A) postavlja u visoko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A pri brojanju prema gore, a u nisko stanje kada je vrijednost registra TCNT0 jednaka vrijednosti registra OCR0A

pri brojanju prema dolje. Ovakva vrsta konfiguracije generira invertirajući PWM signal (slika 10.3).

PWM signal može se generirati i na pinu PD5 (OC0B, kanal B) pomoću sklopa *Timer/Counter0*, no tada je potrebno pogledati konfiguraciju bitova `COM0B1` i `COM0B0` u literaturi [2] u tablici 14-4. PWM signal mogu generirati i ostala dva sklopa: *Timer/Counter1* i *Timer/Counter2*. Konfiguraciju PWM sklopova *Timer/Counter1* i *Timer/Counter2* za *Phase Correct* PWM način rada možete pronaći u literaturi [2] u tablicama 15-1 do 15-6 i 16-1 do 16-6.

Frekvencija PWM signala u *Phase Correct* PWM načinu rada za sklop *Timer/Counter0* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER0 \cdot 2 \cdot VRH0}. \tag{10.16}$$

Vrijednosti varijable $PRESCALER0$ su: 1, 8, 64, 256 ili 1024. Iznos varijable $VRH0$ jest 255 ili vrijednost zapisana u registru `OCR0A`, ovisno o konfiguraciji sklopa *Timer/Counter0*.

Frekvencija PWM signala u *Phase Correct* PWM načinu rada za sklop *Timer/Counter1* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot VRH1}. \tag{10.17}$$

Vrijednosti varijable $PRESCALER1$ su: 1, 8, 64, 256 ili 1024. Iznos varijable $VRH1$ može biti 255, 511, 1023 ili vrijednost zapisana u registre `ICR1` ili `OCR1A`, ovisno o konfiguraciji sklopa *Timer/Counter1*.

Frekvencija PWM signala u *Phase Correct* PWM načinu rada za sklop *Timer/Counter2* jest:

$$F_fPWM = \frac{F_CPU}{PRESCALER2 \cdot 2 \cdot VRH2}. \tag{10.18}$$

Vrijednosti varijable $PRESCALER2$ su: 1, 8, 32, 64, 128, 256 ili 1024. Iznos varijable $VRH2$ jest 255 ili vrijednost zapisana u registru `OCR2A`, ovisno o konfiguraciji sklopa *Timer/Counter2*.

Faktor vođenja D se u *Phase Correct* PWM načinu rada računa na isti način kao i kod *Fast* PWM načina rada prema relaciji (10.8). Vrijednost registra `OCR0A` za željeni faktor vođenja D može se dobiti relacijom (10.9).

Slučajevi u kojima registar `OCR0A` poprima vrijednost 0 (DNO) ili 255 (VRH) specijalni su slučajevi. Kod neinvertirajućeg PWM signala (`COM0A1` $= 1$, `COM0A0` $= 0$) za specijalne slučajeve vrijedi:

- ako je `OCR0A` $= 255$, tada je pin PD6 (OC0A) konstantno u visokom stanju,
- ako je `OCR0A` $= 0$, tada je pin PD6 (OC0A) konstantno u niskom stanju.

Za razliku od *Fast* PWM načina rada, kod *Phase Correct* PWM načina rada ne pojavljuje se uski šiljak u PWM signalu kada je `OCR0A` $= 0$.

U programskom kodu 10.2 prikazana je konfiguracija tajmera za *Phase Correct* PWM način rada.

Programski kod 10.2: Konfiguracija sklopa *Timer/Counter0* za *Phase Correct* PWM način rada

```
TCCR0B |= (0 << CS02) | (1 << CS01) | (0 << CS00);  //F_CPU/8
TCCR0A |= (0 << WGM01) | (1 << WGM00);  // Phase Correct PWM način rada
TCCR0B |= (0 << WGM02);  // VRH je 0x0FF
TCCR0A |= (1 << COM0A1) | (0 << COM0A0); // Neinvertirajući PWM
OCR0A = 80;
```

Djelitelj frekvencije namješten je kao i kod prethodne konfiguracije na iznos 8. *Phase Correct* PWM način rada tajmera odabran je naredbom

`TCCR0A |= (0 << WGM01)| (1 << WGM00);`, prema tablici 9.2. Neinvertirajući način rada odabran je naredbom `TCCR0A |= (1 << COM0A1)| (0 << COM0A0);`. U tom načinu rada vrijednost registra `OCR0A` postavljena je na 80. Srednja vrijednost napona na pinu PD6 (OC0A) za konfiguraciju prikazanu programskim kodom 10.2 može se izračunati prema relaciji (10.11):

$$U_i = \frac{\texttt{OCR0A}}{255} \cdot 5 = \frac{80}{255} \cdot 5 = 1.569V. \tag{10.19}$$

Ako na pinu PD6 (OC0A) želimo postići srednju vrijednost napona iznosa U_{PD6} =4,0 V, tada je u registar `OCR0A` potrebno upisati vrijednost dobivenu sljedećom relacijom (napon napajanja mikroupravljača ATmega328P iznosi U_{VCC} =5 V):

$$\texttt{OCR0A} = \frac{U_{PD6}}{U_{VCC}} \cdot 255 = \frac{4}{5} \cdot 255 = 204. \tag{10.20}$$

Za sklopove *Timer/Counter1* i *Timer/Counter2* konfiguracija *Phase Correct* PWM načina rada slična je prethodno provedenoj konfiguraciji sklopa *Timer/Counter0*. Za sve detalje o konfiguraciji sklopova *Timer/Counter1* i *Timer/Counter2* pogledajte literaturu [2].

U prethodnim programskim kodovima prikazana je konfiguracija tajmera za PWM način rada pomoću registara. Kroz vježbe ćemo dodatno prikazati konfiguriranje tajmera za PWM način rada pomoću funkcija čije se definicije nalaze u biblioteci `timer.h`. Mikroupravljač ATmega328P ima šest PWM izlaza, od kojih ćemo mi u vježbama koristiti tri: PB1 (OC1A), PB2 (OC1B) i PB3 (OC2A). Razlog tome jest taj što na razvojnom okruženju sa slike 2.1 na ovim digitalnim izlazima imamo spojene LED diode. U nastavku ćemo prikazati sve definirane funkcije za konfiguriranje PWM načina rada na svih šest PWM izlaza (funkcije su definirane u zaglavlju `timer.h`).

Konfiguriranje PWM načina rada za sklop *Timer/Counter0*

- `timer0FastPWM()` - funkcija za konfiguraciju sklopa *Timer/Counter0* za *Fast* PWM način rad.
- `timer0PhaseCorrectPWM()` - funkcija za konfiguraciju sklopa *Timer/Counter0* za *Phase Correct* PWM način rad.
- `timer0OC0AEnableNonInvertedPWM()` - funkcija kojom se na kanalu A (OC0A) sklopa *Timer/Counter0* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer0OC0AEnableInvertedPWM()` - funkcija kojom se na kanalu A (OC0A) sklopa *Timer/Counter0* omogućuje generiranja invertirajućeg PWM signala.
- `timer0OC0ADisable()` - funkcija kojom se na kanalu A (OC0A) sklopa *Timer/Counter0* onemogućuje generiranja PWM signala.
- `timer0OC0BEnableNonInvertedPWM()` - funkcija kojom se na kanalu B (OC0B) sklopa *Timer/Counter0* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer0OC0BEnableInvertedPWM()` - funkcija kojom se na kanalu B (OC0B) sklopa *Timer/Counter0* omogućuje generiranja invertirajućeg PWM signala.
- `timer0OC0BDisable()` - funkcija kojom se na kanalu B (OC0B) sklopa *Timer/Counter0* onemogućuje generiranja PWM signala.
- `timer0OC0ADutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu A (OC0A) za sklop *Timer/Counter0*. Ova funkcija prima realan argument `D` čiji je raspon

vrijednosti iz intervala [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).

- `timer0OC0BDutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu B (OC0B) za sklop *Timer/Counter0*. Ova funkcija prima realan argument `D` čiji je raspon vrijednosti iz intervala [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).

Uz navedene funkcije, koristi ćemo i funkciju `timer0SetPrescaler()` koju smo opisali u prošlom poglavlju za namještanje frekvencije PWM signala. Pokažimo nekoliko primjera korištenja navedenih funkcija za sklop *Timer/Counter0* koje se nalaze u zaglavlju `timer.h`:

- `timer0FastPWM();` - sklop *Timer/Counter0* konfiguriran za *Fast* PWM način rad.
- `timer0PhaseCorrectPWM();` - sklop *Timer/Counter0* konfiguriran za *Phase Correct* PWM način rad.
- `timer0OC0AEnableNonInvertedPWM();` - funkcija kojom se na kanalu A (OC0A) sklopa *Timer/Counter0* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer0OC0ADutyCycle(25.4)` - faktor vođenja D na kanalu A (OC0A) postavljen na 25.4%.

Konfiguriranje PWM načina rada za sklop *Timer/Counter1*

- `timer1FastPWM8bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Fast* PWM način rad s rasponom brojanja [0, 255]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_8BIT_TOP`).
- `timer1FastPWM9bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Fast* PWM način rad s rasponom brojanja [0, 511]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_9BIT_TOP`).
- `timer1FastPWM10bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Fast* PWM način rad s rasponom brojanja [0, 1023]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_10BIT_TOP`).
- `timer1FastPWMICR1(uint16_t top)` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Fast* PWM način rad s rasponom brojanja [0, *top*]. Ova funkcija prima cjelobrojni argument `top` čiji je raspon vrijednosti iz intervala [0, 65535]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_ICR1_TOP`).
- `timer1PhaseCorrectPWM8bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Phase Correct* PWM način rad s rasponom brojanja [0, 255]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_8BIT_TOP`).
- `timer1PhaseCorrectPWM9bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Phase Correct* PWM način rad s rasponom brojanja [0, 511]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_9BIT_TOP`).
- `timer1PhaseCorrectPWM10bit()` - funkcija za konfiguraciju sklopa *Timer/Counter1* za *Phase Correct* PWM način rad s rasponom brojanja [0, 1023]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_10BIT_TOP`).
- `timer1PhaseCorrectPWMICR1(uint16_t top)` - funkcija za konfiguraciju sklopa

Timer/Counter1 za *Phase Correct* PWM način rad s rasponom brojanja [0, *top*]. Ova funkcija prima cjelobrojni argument `top` čiji je raspon vrijednosti iz intervala [0, 65535]. Maksimalna vrijednost navedenog raspona definirana je konstantom `PWM_ICR1_TOP`).

- `timer1OC1AEnableNonInvertedPWM()` - funkcija kojom se na kanalu A (OC1A) sklopa *Timer/Counter1* omogućuje generiranja neinvertirajućeg PWM signala.

- `timer1OC1AEnableInvertedPWM()` - funkcija kojom se na kanalu A (OC1A) sklopa *Timer/Counter1* omogućuje generiranja invertirajućeg PWM signala.

- `timer1OC1ADisable()` - funkcija kojom se na kanalu A (OC1A) sklopa *Timer/Counter1* onemogućuje generiranja PWM signala.

- `timer1OC1BEnableNonInvertedPWM()` - funkcija kojom se na kanalu B (OC1B) sklopa *Timer/Counter1* omogućuje generiranja neinvertirajućeg PWM signala.

- `timer1OC1BEnableInvertedPWM()` - funkcija kojom se na kanalu B (OC1B) sklopa *Timer/Counter1* omogućuje generiranja invertirajućeg PWM signala.

- `timer1OC1BDisable()` - funkcija kojom se na kanalu B (OC1B) sklopa *Timer/Counter1* onemogućuje generiranja PWM signala.

- `timer1OC1ADutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu A (OC1A) za sklop *Timer/Counter1*. Ova funkcija prima realan argument `D` čiji je raspon [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).

- `timer1OC1BDutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu B (OC1B) za sklop *Timer/Counter1*. Ova funkcija prima realan argument `D` čiji je raspon [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).

Preddefinirane konstante za maksimalne vrijednosti raspona brojanja sklopa *Timer/Counter1* u PWM načinima rada prikazane su programskim kodom 10.3. Ove konstante koriste funkcije za promjenu faktora vođenja.

Programski kod 10.3: Preddefinirane konstante za maksimalne vrijednosti raspona brojanja sklopa *Timer/Counter1* u PWM načinima rada

```
//VRH vrijednost za PWM sklopa Timer/Counter1
#define PWM_8BIT_TOP 255
#define PWM_9BIT_TOP 511
#define PWM_10BIT_TOP 1023
#define PWM_ICR1_TOP ICR1
```

Uz navedene funkcije, koristi ćemo i funkciju `timer1SetPrescaler()` koju smo opisali u prošlom poglavlju za namještanje frekvencije PWM signala. Pokažimo nekoliko primjera korištenja navedenih funkcija za sklop *Timer/Counter1* koje se nalaze u zaglavlju `timer.h`:

- `timer1FastPWM8bit();` - sklop *Timer/Counter0* konfiguriran za *Fast* PWM način rad s rasponom brojanja od [0, 255].

- `timer1PhaseCorrectPWM10bit();` - sklop *Timer/Counter0* konfiguriran za *Phase Correct* PWM način rad s rasponom brojanja od [0, 1023].

- `timer1PhaseCorrectPWMICR1(9999);` - sklop *Timer/Counter0* konfiguriran za *Phase Correct* PWM način rad s rasponom brojanja od [0, 9999]. U ovom slučaju imamo mogućnost odabira 10090 diskretnih i različitih faktora vođenja D.

- `timer1OC1BEnableNonInvertedPWM();` - funkcija kojom se na kanalu B (OC1B) sklopa *Timer/Counter1* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer1OC1BDutyCycle(50.8)` - faktor vođenja D na kanalu B (OC1B) postavljen na 50.8%.

Konfiguriranje PWM načina rada za sklop *Timer/Counter2*

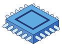
- `timer2FastPWM()` - funkcija za konfiguraciju sklopa *Timer/Counter2* za *Fast* PWM način rad.
- `timer2PhaseCorrectPWM()` - funkcija za konfiguraciju sklopa *Timer/Counter2* za *Phase Correct* PWM način rad.
- `timer2OC2AEnableNonInvertedPWM()` - funkcija kojom se na kanalu A (OC2A) sklopa *Timer/Counter2* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer2OC2AEnableInvertedPWM()` - funkcija kojom se na kanalu A (OC2A) sklopa *Timer/Counter2* omogućuje generiranja invertirajućeg PWM signala.
- `timer2OC2ADisable()` - funkcija kojom se na kanalu A (OC2A) sklopa *Timer/Counter2* onemogućuje generiranja PWM signala.
- `timer2OC2BEnableNonInvertedPWM()` - funkcija kojom se na kanalu B (OC2B) sklopa *Timer/Counter2* omogućuje generiranja neinvertirajućeg PWM signala.
- `timer2OC2BEnableInvertedPWM()` - funkcija kojom se na kanalu B (OC2B) sklopa *Timer/Counter2* omogućuje generiranja invertirajućeg PWM signala.
- `timer2OC2BDisable()` - funkcija kojom se na kanalu B (OC2B) sklopa *Timer/Counter2* onemogućuje generiranja PWM signala.
- `timer2OC2ADutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu A (OC2A) za sklop *Timer/Counter2*. Ova funkcija prima realan argument `D` čiji je raspon [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).
- `timer2OC2BDutyCycle(float D)` - funkcija kojom se mijenja faktor vođenja D na kanalu B (OC2B) za sklop *Timer/Counter2*. Ova funkcija prima realan argument `D` čiji je raspon [0.0, 100.0] (faktor vođenja, odnosno širina impulsa PWM signala od 0 do 100%).

Uz navedene funkcije, koristi ćemo i funkciju `timer2SetPrescaler()` koju smo opisali u prošlom poglavlju za namještanje frekvencije PWM signala. Pokažimo nekoliko primjera korištenja navedenih funkcija za sklop *Timer/Counter2* koje se nalaze u zaglavlju `timer.h`:

- `timer2FastPWM();` - sklop *Timer/Counter2* konfiguriran za *Fast* PWM način rad.
- `timer2PhaseCorrectPWM();` - sklop *Timer/Counter2* konfiguriran za *Phase Correct* PWM način rad.
- `timer2OC2BEnableInvertedPWM();` - funkcija kojom se na kanalu B (OC2B) sklopa *Timer/Counter2* omogućuje generiranja invertirajućeg PWM signala.
- `timer2OC2BDutyCycle(75.0)` - faktor vođenja D na kanalu B (OC2B) postavljen na 75.0%.

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `PWM.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom

dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `PWM.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\PWM\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Pulsno širinska modulacija`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

 Vježba 10.1

Napišite program koji će mijenjati intenzitet zelene LED diode na razvojnom okruženju s mikroupravljačem ATmega328P pomoću neinvertirajućeg PWM signala koji ćete generirati na digitalnom izlazu PB3 (OC2A). Intenzitet zelene LED diode mijenjajte promjenom faktora vođenja PWM signala s korakom promjene iznosa 10% svakih 1000 ms (0% → 10% → 20% ... → 90% → 100% → 0% ...). Na LCD displeju ispišite faktor vođenja i frekvenciju PWM signala. Prema shemi na slici 4.1, zelena LED dioda spojena je na digitalni izlaz PB3 (OC2A) mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba101.cpp`. Omogućite prevođenje datoteke `vjezba101.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba101.cpp` prikazan je programskim kodom 10.4.

Programski kod 10.4: Početni sadržaj datoteke `vjezba101.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D = 0.0;
    while (1) {

        _delay_ms(1000);
    }
}
```

Cilj ove vježbe jest mijenjati intenzitet zelene LED diode prema zadanom obrascu pomoću PWM signala. U vježbi nije eksplicitno navedeno na koji način je potrebno ostvariti navedenu funkcionalnost. Zelena LED dioda spojena je na digitalni izlaz PB3. Prema rasporedu pinova mikroupravljača ATmega328P na slici 2.4, alternativna primjena pina PB3 omogućuje generiranje PWM signala na kanalu A pomoću sklopa *Timer/Counter2*. Alternativni naziv za pin PB3 jest OC2A.

Alternativni naziv pina ključan je parametar pri konfiguraciji PWM načina rada. Cijeli postupak konfiguracije provest ćemo za sklop *Timer/Counter2* i kanal A. Kroz postupak konfiguriranja sklopa *Timer/Counter2* prikazat ćemo *Fast* PWM i *Phase Correct* PWM način rada.

Prikažimo sada postupak konfiguriranja sklopa *Timer/Counter2* za *Fast* PWM način rada.

U prvom koraku namjestit ćemo frekvenciju PWM signala za *Fast* PWM način rada. Frekvencija PWM signala u *Fast* PWM načinu rada može se dobiti prema relaciji 10.7, a ovisi o frekvenciji radnog takta, djelitelju frekvencije radnog takta i vršnoj vrijednosti registra `TCNT2`. Odaberimo djelitelj frekvencije iznosa 8 (frekvencija radnog takta jest 16 MHz). Prema relaciji (10.7), frekvencija PWM signala u *Fast* PWM načinu rada za sklop *Timer/Counter2* bit će:

$$F_fPWM = \frac{F_CPU}{PRESCALER2 \cdot (VRH2 + 1)} = \frac{16000000}{8 \cdot (255 + 1)} = 7812,5\ \mathtt{Hz}. \qquad (10.21)$$

Dobivena frekvencija je dovoljno velika da naše oko (zbog tromosti) ne vidi uključenje i isključenje LED diode. U relaciju (10.21) pokušajte uvrstiti i druge djelitelje frekvencije kako biste imali uvid o mogućim frekvencijama PWM signala za sklop *Timer/Counter2*.

Konfiguriranje sklopa *Timer/Counter2* za *Fast* PWM način rada provodi se u registru `TCCR2A` i registru `TCCR2B` pomoću bitova `WGM20`, `WGM21` i `WGM22`, a prema tablici 17-8 u literaturi [2]. Za *Fast* PWM način rada, bit `WGM20` mora imati vrijednost 1, bit `WGM21` vrijednost 1, a bit `WGM22` vrijednost 0. Djelitelj frekvencije radnog takta konfigurira se u registru `TCCR2B` pomoću bitova `CS20`, `CS21` i `CS22`, a prema tablici 17-9 u literaturi [2]. Za djelitelj frekvencije iznosa 8 bit `CS20` mora imati vrijednost 0, bit `CS21` vrijednost 1, a bit `CS22` vrijednost 0. Omogućenje generiranje neinvertirajućeg PWM signala sklopa *Timer/Counter2* na kanalu A postiže se registrom `TCCR2A` tako da se bit `COM2A0` postavi u vrijednost 0, a bit `COM2A1` postavi u vrijednost 1 (prema tablici 17-3 u literaturi [2]). Ovime je konfiguracija sklopa *Timer/Counter2* za *Fast* PWM način rada završena. Konfiguracija sklopa *Timer/Counter2* prema navedenom opisu prikazana je programskim kodom 10.5. Navedenu konfiguraciju napišite u inicijalizacijsku funkciju `init()`.

Programski kod 10.5: Konfiguracija sklopa *Timer/Counter2*: *Fast* PWM način rada, djelitelj frekvencije postavljen na iznos 8, omogućeno generiranje neinvertirajućeg PWM signala

```
// postavljanje Fast PWM načina rada za timer2
TCCR2A |= (1 << WGM21) | (1 << WGM20);
TCCR2B |= (0 << WGM22);
// djelitelj frekvencije F_CPU / 8
TCCR2B |= ((0 << CS22) | (1 << CS21) | (0 << CS20));
// neinvertirajući PWM signal na PB3 (OC2A)
TCCR2A |= (1 << COM2A1) | (0 << COM2A0);
```

Nakon konfiguracije sklopa *Timer/Counter2* za *Fast* PWM način rada potrebno je odrediti vrijednosti registra `OCR2A` za koju će faktor vođenja PWM signala biti 0%, 10%, 20%, ..., 90% i 100%. Izračunat ćemo širinu impulsa za nekoliko faktora vođenja. Prema relaciji (10.9) vrijednost `OCR2A` registra za širinu PWM signala 20 % perioda T bit će:

$$\mathtt{OCR2A} = D \cdot 255 = 0.2 \cdot 255 = 51. \qquad (10.22)$$

Prema relaciji (10.9) vrijednost `OCR2A` registra za širinu PWM signala 50 % perioda T bit će:

$$\mathtt{OCR2A} = D \cdot 255 = 0.5 \cdot 255 = 127. \qquad (10.23)$$

Primijetimo da je vrijednost `OCR2A` registra uvijek cjelobrojna. Da bismo realizirali sve faktore vođenja od 0 do 100% s korakom promjene 10%, koristit ćemo sljedeću relaciju:

$$\mathtt{OCR2A} = D \cdot 255 \qquad (10.24)$$

U programski kod10.4 u beskonačnu `while` petlju napišite sljedeće naredbe:

- `lcdClrScr();` - brisanje LCD displeja,
- `lcdprintf("D = %.1f%%\n", D * 100.0);` - ispis faktora vođenja D u prvom retku.

- `lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 256);` - ispis frekvencije PWM signala u drugom retku na dva decimalna mjesta. Drugi argument funkcije `lcdprintf()` jest izraz za frekvenciju PWM signala dan relacijom (10.21).
- `OCR2A = D * 255;` - određivanje vrijednosti `OCR2A` registra s obzirom na faktor vođenja D,
- `D += 0.1;` - uvećavanje faktora vođenja za 0.1 što je ekvivalent povećanju za 10%,
- `if (D > 1.0) { }` - uvjet kojim se provjerava da li je faktor vođenja veći od 1.0, odnosno 100%:
  - unutar vitičastih zagrada upišite `D = 0.0;` kako bi osigurali novi ciklus promjene faktora vođenja od 0 do 100%.

Prilikom generiranja PWM signala, obavezno je PWM izlaz koji koristite konfigurirati kao digitalni izlaz (u programskom kodu 10.5 pogledati naredbu `DDRB |= (1 << PB3);`).

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba101.cpp` treba biti ista kao programski kod 10.6.

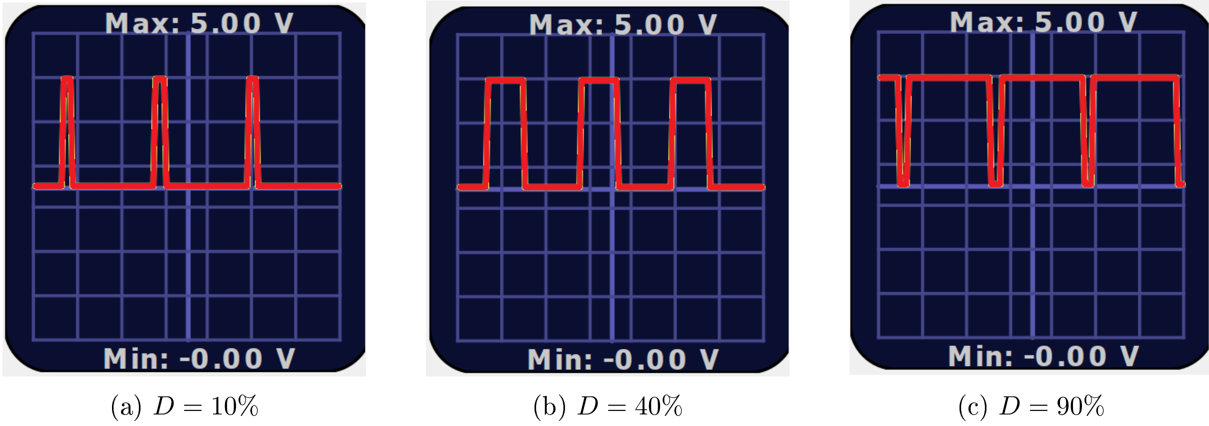
Programski kod 10.6: Promjena intenziteta zelene LED diode pomoću PWM signala sa sklopom *Timer/Counter2* u *Fast* PWM načinu rada

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    // postavljanje Fast PWM načina rada za timer2
    TCCR2A |= (1 << WGM21) | (1 << WGM20);
    TCCR2B |= (0 << WGM22);
    // djelitelj frekvencije F_CPU / 8
    TCCR2B |= ((0 << CS22) | (1 << CS21) | (0 << CS20));
    // neinvertirajući PWM signal na PB3 (OC2A)
    TCCR2A |= (1 << COM2A1) | (0 << COM2A0);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D = 0.0;
    while (1) {
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("D = %.1f%%\n", D * 100.0);
        lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 256);
        // osvježavanje faktora vođenja kroz registar OCR2A
        OCR2A = D * 255;
        // uvećaj faktor za 0.1 (10%)
        D += 0.1;
        // ako je faktor vođenja veći od 1.0 (100%)
        if (D > 1.0) {
            D = 0.0; // vrati faktor vođenja na 0.0 (0%)
        }
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba101.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Ako ste slijedili navedene korake, na LCD displeju ispisivat će se faktor vođenja i frekvencija PWM sgnala, a zelena LED dioda mijenjat će intenzitet.

Na slici 10.4 je prikazan PWM signal za tri različita faktora vođenja D pomoću osciloskopa. Kako možete primijetiti, širina impulsa PWM signala razmjerna je faktoru vođenju D.



(a) $D = 10\%$ (b) $D = 40\%$ (c) $D = 90\%$

Slika 10.4: Prikaz širine impulsa na pinu PB3 (OC2A) pomoću osciloskopa

Promatrajte zelenu LED diodu za faktor vođenja $D = 0$. Zelena LED dioda za faktor vođenja $D = 0$ trebala bi biti isključena, no primijetit ćete da svijetli najmanjim mogućim intenzitetom ("tinja"). Kada je vrijednost registra `OCR2A` jednaka 0, na PWM kanalu generira se impuls duljine $\frac{1}{256}T$, gdje je T period PWM signala. Navedeni problem može se riješiti korištenjem sklopa *Timer/Counter2* u *Phase Correct* PWM načinu rada.

Prikažimo sada postupak konfiguriranja sklopa *Timer/Counter2* za *Phase Correct* PWM način rada. U prvom koraku namjestit ćemo frekvenciju PWM signala za *Phase Correct* PWM način rada. Frekvencija PWM signala u *Phase Correct* PWM načinu rada može se dobiti prema relaciji 10.18, a ovisi o frekvenciji radnog takta, djelitelju frekvencije radnog takta i dvostrukoj vršnoj vrijednosti registra `TCNT2`. Odaberimo djelitelj frekvencije iznosa 8 (frekvencija radnog takta jest 16 MHz). Prema relaciji (10.18), frekvencija PWM signala u *Phase Correct* PWM načinu rada bit će:

$$F_fPWM = \frac{F_CPU}{PRESCALER2 \cdot 2 \cdot VRH2} = \frac{16000000}{8 \cdot 510} = 3906,25\ \mathtt{Hz} = 3,906\ \mathtt{kHz}. \quad (10.25)$$

Dobivena frekvencija je približno dva puta manja, nego u slučaju kada je sklop *Timer/Counter2* konfiguriran u *Fast* PWM načinu rada.

Konfiguriranje sklopa *Timer/Counter2* za *Phase Correct* PWM način rada provodi se u registru `TCCR2A` i registru `TCCR2B` pomoću bitova `WGM20`, `WGM21` i `WGM22`, a prema tablici 17-8 i literaturi [2]. Za *Phase Correct* PWM način rada bit `WGM20` mora imati vrijednost 1, bit `WGM21` vrijednost 0, a bit `WGM22` vrijednost 0. Preostali bitovi u registrima `TCCR2A` i `TCCR2B` ostaju nepromijenjeni u odnosu na *Fast* PWM način rada. U programskom kodu 10.6, naredbu `TCCR2A |= (1 << WGM21)| (1 << WGM20);` zamijenite s naredbom `TCCR2A |= (0 << WGM21)| (1 << WGM20);`.

S obzirom na promjenu frekvencije PWM signala, u programskom kodu 10.6 u beskonačnoj `while` petlji zamijenite naredbu:

- `lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 256);`

s naredbom:

- `lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 510);` - ispis frekvencije PWM signala u

drugom retku na dva decimalna mjesta. Drugi argument funkcije `lcdprintf()` jest izraz za frekvenciju PWM signala dan relacijom (10.25).

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba101.cpp` treba biti ista kao programski kod 10.7.

Programski kod 10.7: Promjena intenziteta zelene LED diode pomoću PWM signala sa sklopom *Timer/Counter2* u *Phase Correct* PWM načinu rada

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    // postavljanje Phase Correct PWM načina rada za timer2
    TCCR2A |= (0 << WGM21) | (1 << WGM20);
    TCCR2B |= (0 << WGM22);
    // djelitelj frekvencije F_CPU / 8
    TCCR2B |= ((0 << CS22) | (1 << CS21) | (0 << CS20));
    // neinvertirajući PWM signal na PB3 (OC2A)
    TCCR2A |= (1 << COM2A1) | (0 << COM2A0);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D = 0.0;
    while (1) {
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("D = %.1f%%\n", D * 100.0);
        lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 510);
        // osvježavanje faktora vođenja kroz registar OCR2A
        OCR2A = D * 255;
        // uvećaj faktor za 0.1 (10%)
        D += 0.1;
        // ako je faktor vođenja veći od 1.0 (100%)
        if (D > 1.0) {
            D = 0.0; // vrati faktor vođenja na 0.0 (0%)
        }
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba101.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se faktor vođenja i frekvencija PWM sgnala, a zelena LED dioda mijenjat će intenzitet. Primijetite da je za faktor vođenja $D = 0$ zelena LED dioda sada potpuno isključena.

U nastavku ove vježbe za konfiguraciju sklopa *Timer/Counter2* koristit ćemo funkcije za konfiguriranje *Fast* PWM i *Phase Correct* PWM načina rada koje smo opisali u uvodnom djelu ovog poglavlja. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"` (dodajte u programski kod ovu naredbu). Konfiguracija sklopa *Timer/Counter2* za *Fast* PWM način rada prikazana je programskim kodom 10.8.

Programski kod 10.8: Konfiguracija sklopa *Timer/Counter2*: *Fast* PWM način rada, djelitelj frekvencije postavljen na iznos 8, omogućeno generiranje neinvertirajućeg PWM signala - funkcijski pristup konfiguraciji

```
// postavljanje Fast PWM načina rada za timer2
timer2FastPWM();
// djelitelj frekvencije F_CPU / 8
timer2SetPrescaler(TIMER2_PRESCALER_8);
// neinvertirajući PWM signal na PB3 (OC2A)
timer2OC2AEnableNonInvertedPWM();
```

Navedenu konfiguraciju prikazanu programskim kodom 10.8, zamijenite s konfiguracijom pomoću registara u programskom kodu 10.6. U svrhu promjene širine impulsa PWM signala, umjesto naredbe `OCR2A = D * 255;`, koristit ćemo naredbu `timer2OC2ADutyCycle(D);`. Pri tome je važno napomenuti da faktor vođenja D u ovom slučaju promatramo u postocima (D poprima vrijednosti od 0 do 100%). Promijenite programski kod 10.6 s obzirom na faktor vođenja D (varijabla `D` uvećava se za 10.0, a u `if` grananju se varijabla `D` uspoređuje sa 100.0).

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba101.cpp` treba biti ista kao programski kod 10.9.

Programski kod 10.9: Promjena intenziteta zelene LED diode pomoću PWM signala sa sklopom *Timer/Counter2* u *Fast* PWM načinu rada - funkcijski pristup konfiguraciji

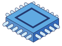
```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    // postavljanje Fast PWM načina rada za timer2
    timer2FastPWM();
    // djelitelj frekvencije F_CPU / 8
    timer2SetPrescaler(TIMER2_PRESCALER_8);
    // neinvertirajući PWM signal na PB3 (OC2A)
    timer2OC2AEnableNonInvertedPWM();
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D = 0.0;
    while (1) {
        // brisanje znakova LCD displeja +  home pozicija kursora
        lcdClrScr();
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdprintf("D = %.1f%%\n", D);
        lcdprintf("f = %.2fHz\n", F_CPU / 8.0 / 256);
        // osvježavanje faktora vođenja kroz registar OCR2A
        timer2OC2ADutyCycle(D);
        // uvećaj faktor za 10%
        D += 10.0;
        // ako je faktor vođenja veći od 100%
        if (D > 100.0) {
            D = 0.0; // vrati faktor vođenja na 0%
        }
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba101.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Ako ste slijedili navedene korake, na LCD displeju ispisivat će se faktor vođenja i frekvencija PWM sgnala, a zelena LED dioda mijenjat će intenzitet.

Za konfiguriranje sklopa *Timer/Counter2* u *Phase Correct* PWM načinu rada, u programskom kodu 10.9 potrebno je naredbu `timer2FastPWM();` zamijeniti naredbom `timer2PhaseCorrectPWM();`. Prevedite datoteku `vjezba101.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba101.cpp` i onemogućite prevođenje ove datoteke.



Vježba 10.2

Napišite program koji će mijenjati intenzitet zelene LED diode na razvojnom okruženju s mikroupravljačem ATmega328P pomoću neinvertirajućeg PWM signala koji ćete generirati na digitalnom izlazu PB3 (OC2A). Intenzitet zelene LED diode mijenjajte pomoću potenciometra spojenog na analogni ulaz ADC0 (PC0). Sklop *Timer/Counter2* konfigurirajte za *Phase Correct* PWM način rada. Na LCD displeju ispišite faktor vođenja D. Prema shemi na slici 4.1, zelena LED dioda spojena je na digitalni izlaz PB3 (OC2A) mikroupravljača ATmega328P. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba102.cpp`. Omogućite prevođenje datoteke `vjezba102.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba102.cpp` prikazan je programskim kodom 10.10.

Programski kod 10.10: Početni sadržaj datoteke `vjezba102.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D;
    while (1) {

        _delay_ms(100);
    }
}
```

U nastavku ove vježbe za konfiguraciju sklopa *Timer/Counter2* koristit ćemo funkcije za konfiguriranje *Fast* PWM i *Phase Correct* PWM načina rada koje smo opisali u uvodnom djelu ovog poglavlja. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. U ovoj vježbi koristit ćemo sklop *Timer/Counter2* u *Phase Correct* PWM načinu rada koji generira neinvertirajući PWM signal. Konfiguracija sklopa *Timer/Counter2* za *Phase Correct* PWM način rada prikazana je programskim kodom 10.11.

Programski kod 10.11: Konfiguracija sklopa *Timer/Counter2*: *Phase Correct* PWM način rada, djelitelj frekvencije postavljen na iznos 8, omogućeno generiranje neinvertirajućeg PWM signala - funkcijski pristup konfiguraciji

```
// postavljanje Phase Correct PWM načina rada za timer2
timer2PhaseCorrectPWM();
// djelitelj frekvencije F_CPU / 8
timer2SetPrescaler(TIMER2_PRESCALER_8);
// neinvertirajući PWM signal na PB3 (OC2A)
timer2OC2AEnableNonInvertedPWM();
```

Navedenu konfiguraciju napišite u inicijalizacijsku funkciju `init()`. Faktor vođenja D potrebno je mijenjati pomoću potenciometra koji je spojen na analogni ulaz ADC0. S obzirom da ćemo u programskom kodu koristiti AD pretvorbu, u programski kod uključite zaglavlje `adc.h` pomoću naredbe `#include "ADC/adc.h"`.

Faktor vođenja D poprima vrijednosti u rasponu od 0 do 100 pa ćemo stanje potenciometra čitati funkcijom `adcReadScale0To100`. U programski kod 10.10 u beskonačnu `while` petlju napišite sljedeće naredbe:

- `D = adcReadScale0To100(ADC0);` - čitanje rezultata AD pretvorbe na kanalu ADC0 (potenciometar), skaliranje rezultata u raspon [0, 100] te spremanje u varijablu `D` (faktor vođenja),
- `lcdHome();` - pozicioniranje kursora u prvi redak i prvi stupac za ispis faktora vođenja,
- `lcdprintf("D = %.1f%%       ", D);` - ispis teksta `D =`, faktora vođenja u %, brisanje sljedećih 6 mjesta pomoću znakova *white space* kako bismo osigurali brisanje starog teksta te pozicioniranje u novi redak.
- `timer2OC2ADutyCycle(D);` - postavljanje faktora vođenja D PWM signala na kanalu A.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba102.cpp` treba biti ista kao programski kod 10.12.

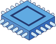
Programski kod 10.12: Promjena intenziteta zelene LED diode pomoću potenciometra

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    DDRB |= (1 << PB3); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    // postavljanje Fast PWM načina rada za timer2
    timer2PhaseCorrectPWM();
    // djelitelj frekvencije F_CPU / 8
    timer2SetPrescaler(TIMER2_PRESCALER_8);
    // neinvertirajući PWM signal na PB3 (OC2A)
    timer2OC2AEnableNonInvertedPWM();
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D;
    while (1) {
        // promjena faktora vođenja pomocu ADC0
        D = adcReadScale0To100(ADC0);
```

```
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdHome();
        lcdprintf("D = %.1f%%       ", D);
        // osvježavanje faktora vođenja kroz registar OCR2A
        timer2OC2ADutyCycle(D);
        _delay_ms(100);
    }
}
```

Prevedite datoteku `vjezba102.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, na LCD displeju ispisivat će se faktor vođenja. Pomoću potenciometra sada možete mijenjati intenzitet zelene LED diode.

Zatvorite datoteku `vjezba102.cpp` i onemogućite prevođenje ove datoteke.



Vježba 10.3

Napišite program koji će mijenjati intenzitet crvene LED diode pomoću neinvertirajućeg PWM signala koji ćete generirati na digitalnom izlazu PB1 (OC1A) i žute LED diode pomoću invertirajućeg PWM signala koji ćete generirati na digitalnom izlazu PB2 (OC2A). Intenzitet crvene i žute LED diode mijenjajte pomoću potenciometra spojenog na analogni ulaz ADC0 (PC0). Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni izlaz PB1 (OC1A), a žuta LED dioda na digitalni izlaz PB2 (OC2A) mikroupravljača ATmega328P. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3.

U projektnom stablu otvorite datoteku `vjezba103.cpp`. Omogućite prevođenje datoteke `vjezba103.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba103.cpp` prikazan je programskim kodom 10.13.

Programski kod 10.13: Početni sadržaj datoteke `vjezba103.cpp`

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)

}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    while (1) {

    }
}
```

Cilj ove vježbe jest mijenjati intenzitet crvene i žute LED diode prema zadanom obrascu pomoću PWM signala. U vježbi nije eksplicitno navedeno na koji način je potrebno ostvariti navedenu funkcionalnost. Crvena LED dioda spojena je na digitalni izlaz PB1, a žuta LED dioda na digitalni izlaz PB2. Prema rasporedu pinova mikroupravljača ATmega328P na slici 2.4, alternativna primjena pina PB1 (OC1A) omogućuje generiranje PWM signala na kanalu A pomoću sklopa *Timer/Counter1*. Alternativna primjena pina PB2 (OC1B)

omogućuje generiranje PWM signala na kanalu B pomoću sklopa *Timer/Counter1*. Cijeli postupak konfiguracije provest ćemo za sklop *Timer/Counter1* te kanale A i B. Kroz postupak konfiguriranja sklopa *Timer/Counter1* prikazat ćemo *Fast* PWM i *Phase Correct* PWM način rada u različitim rezolucijama.

Prikažimo sada postupak konfiguriranja sklopa *Timer/Counter1* za *Fast* PWM način rada rezolucije 10 bitova. Vršna vrijednost koju postiže registar `TCNT1` pri brojanju prema gore kod 10-bitnog *Fast* PWM načina rada jest 1023. U prvom koraku namjestit ćemo frekvenciju PWM signala za *Fast* PWM način rada. Frekvencija PWM signala u *Fast* PWM načinu rada može se dobiti prema relaciji 10.6, a ovisi o frekvenciji radnog takta, djelitelju frekvencije radnog takta i vršnoj vrijednosti registra `TCNT1`. Odaberimo djelitelj frekvencije iznosa 8 (frekvencija radnog takta jest 16 MHz). Prema relaciji (10.6), frekvencija PWM signala u *Fast* PWM načinu rada rezolucije 10 bitova za sklop *Timer/Counter1* bit će:

$$F_fPWM = \frac{F_CPU}{PRESCALER1 \cdot (VRH1 + 1)} = \frac{16000000}{8 \cdot (1023 + 1)} = 1953,13\,\texttt{Hz}. \tag{10.26}$$

Dobivena frekvencija je dovoljno velika da naše oko (zbog tromosti) ne vidi uključenje i isključenje LED diode. U relaciju (10.26) pokušajte uvrstiti i druge djelitelje frekvencije kako biste imali uvid o mogućim frekvencijama PWM signala za sklop *Timer/Counter1* .

Konfiguriranje sklopa *Timer/Counter1* za *Fast* PWM način rada rezolucije 10 bitova provodi se u registru `TCCR1A` i registru `TCCR1B` pomoću bitova `WGM10`, `WGM11`, `WGM12` i `WGM13`, a prema tablici 15-4 u literaturi [2]. Za *Fast* PWM način rada rezolucije 10 bitova, bit `WGM10` mora imati vrijednost 1, bit `WGM11` vrijednost 1, bit `WGM12` vrijednost 1, a bit `WGM13` vrijednost 0. Djelitelj frekvencije radnog takta konfigurira se u registru `TCCR1B` pomoću bitova `CS10`, `CS11` i `CS12`, a prema tablici 15-5 u literaturi [2]. Za djelitelj frekvencije iznosa 8 bit `CS10` mora imati vrijednost 0, bit `CS11` vrijednost 1, a bit `CS12` vrijednost 0. Omogućenje generiranje neinvertirajućeg PWM signala sklopa *Timer/Counter1* na kanalu A postiže se registrom `TCCR1A` tako da se bit `COM1A0` postavi u vrijednost 0, a bit `COM1A1` postavi u vrijednost 1 (prema tablici 15-2 u literaturi [2]). Generiranje invertirajućeg PWM signala sklopa *Timer/Counter1* na kanalu B postiže se registrom `TCCR1A` tako da se bit `COM1B0` postavi u vrijednost 1, a bit `COM1B1` postavi u vrijednost 1 (prema tablici 15-2 u literaturi [2]).Ovime je konfiguracija sklopa *Timer/Counter1* za *Fast* PWM način rada rezolucije 10 bitova završena. Konfiguracija sklopa *Timer/Counter1* prema navedenom opisu prikazana je programskim kodom 10.14. Navedenu konfiguraciju napišite u inicijalizacijsku funkciju `init()`. Primijetite da se *Fast* PWM način rada i dijelitelj frekvencije radnog takta konfigurira na razini cijelog sklopa *Timer/Counter1* te da se konfiguracija odnosi i na kanal A i na kanal B. Za pojedine kanale A i B možemo odvojeno konfigurirati generiranje neinvertirajućeg, odnosno invertirajućeg PWM signala.

Programski kod 10.14: Konfiguracija sklopa *Timer/Counter1*: *Fast* PWM način rada rezolucije 10 bitova, djelitelj frekvencije postavljen na iznos 8, omogućeno generiranje neinvertirajućeg PWM signala na kanalu A i invertirajućeg PWM signala na kanalu B

```
// postavljanje Fast PWM načina rada za timer1 - 10 bita
TCCR1A |= (1 << WGM11) | (1 << WGM10);
TCCR1B |= (0 << WGM13) | (1 << WGM12);
// djelitelj frekvencije F_CPU / 8
TCCR1B |= ((0 << CS12) | (1 << CS11) | (0 << CS10));
// neinvertirajući PWM signal na PB1 (OC1A)
TCCR1A |= (1 << COM1A1) | (0 << COM1A0);
// invertirajući PWM signal na PB2 (OC1B)
TCCR1A |= (1 << COM1B1) | (1 << COM1B0);
```

Faktor vođenja D potrebno je mijenjati pomoću potenciometra koji je spojen na analogni ulaz ADC0. S obzirom da ćemo u programskom kodu koristiti AD pretvorbu, u programski kod uključite zaglavlje `adc.h` pomoću naredbe `#include "ADC/adc.h"`. Promjenu faktora vođenja

na kanalu A i B ostvarit ćemo kroz registre OCR1A i OCR1B. Pri tome se registrom OCR1A mijenja faktor vođenja PWM signala na kanalu A, a registrom OCR1B mijenja faktor vođenja PWM signala na kanalu B. Primijetite da je rezolucija AD pretvorbe mikroupravljača ATmega328P jednaka kao i rezolucija *Fast* PWM načina rada koji koristimo (10 bita). Prema tome, širinu PWM signala, odnosno faktor vođenja možemo mijenjati tako da u registre OCR1A i OCR1B pridružimo upravo rezultat AD pretvorbe. I rezultat AD pretvorbe i vrijednosti registra TCNT1 se kreću u rasponu [0, 1023].

U programski kod10.13 u beskonačnu `while` petlju napišite sljedeće naredbe:

- `OCR1A = adcRead(ADC0);` - rezultat AD pretvorbe na kanalu ADC0 pridružuje se u registar OCR1A s ciljem promjene širine PWM signala (faktora vođenja) na kanalu A (PB1 ili OC1A),

- `OCR1B = adcRead(ADC0);` - rezultat AD pretvorbe na kanalu ADC0 pridružuje se u registar OCR1B s ciljem promjene širine PWM signala (faktora vođenja) na kanalu B (PB2 ili OC1B).

Druga u nizu naredba može se zamijeniti s naredbom `OCR1B = OCR1A;` jer su vrijednosti registra OCR1A i OCR1B, a na ovaj način se AD pretvorba provodi samo jednom.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba103.cpp` treba biti ista kao programski kod 10.15.

Programski kod 10.15: Promjena intenziteta crvene i žute LED diode pomoću potenciometra - *Fast* PWM način rada rezolucije 10 bitova

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    // postavljanje Fast PWM načina rada za timer1 - 10 bita
    TCCR1A |= (1 << WGM11) | (1 << WGM10);
    TCCR1B |= (0 << WGM13) | (1 << WGM12);
    // djelitelj frekvencije F_CPU / 8
    TCCR1B |= ((0 << CS12) | (1 << CS11) | (0 << CS10));
    // neinvertirajući PWM signal na PB1 (OC1A)
    TCCR1A |= (1 << COM1A1) | (0 << COM1A0);
    // invertirajući PWM signal na PB2 (OC1B)
    TCCR1A |= (1 << COM1B1) | (1 << COM1B0);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    while (1) {
        // osvježavanje faktora vođenja kroz registar OCR1A i OCR1B
        OCR1A = adcRead(ADC0);
        OCR1B = adcRead(ADC0);
    }
}
```

Prevedite datoteku `vjezba103.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, pomoću potenciometra možete mijenjati intenzitet crvene i žute LED diode u stvarnom vremenu. Promjena intenziteta crvene i žute LED diode komplementarna je: dok crvena LED dioda povećava intenzitet, žuta ga smanjuje i obrnuto.

Ukoliko želite konfigurirati sklop *Timer/Counter1* za *Fast* PWM način rada rezolucije 8

ili 9 bitova možete to učiniti pomoću konfiguracija prikazanih programskim kodovima 10.16 i 10.17 (prema tablici 15-4 u literaturi [2]). Četvrti način na koji se može konfigurirati sklop *Timer/Counter1* za *Fast* PWM način rada prikazat ćemo u sljedećoj vježbi.

Programski kod 10.16: Konfiguracija sklopa *Timer/Counter1*: *Fast* PWM način rada rezolucije 8 bitova

```
// postavljanje Fast PWM načina rada za timer1 - 8 bita
TCCR1A |= (0 << WGM11) | (1 << WGM10);
TCCR1B |= (0 << WGM13) | (1 << WGM12);
```

Programski kod 10.17: Konfiguracija sklopa *Timer/Counter1*: *Fast* PWM način rada rezolucije 9 bitova

```
// postavljanje Fast PWM načina rada za timer1 - 9 bita
TCCR1A |= (1 << WGM11) | (0 << WGM10);
TCCR1B |= (0 << WGM13) | (1 << WGM12);
```

U nastavku ćemo prikazati postupak konfiguriranja sklopa *Timer/Counter1* za *Phase Correct* PWM način rada rezolucije 9 bitova. Vršna vrijednost koju postiže registar `TCNT1` pri brojanju prema gore kod 9-bitnog *Phase Correct* PWM načina rada jest 511. U prvom koraku namjestit ćemo frekvenciju PWM signala za *Phase Correct* PWM način rada. Frekvencija PWM signala u *Phase Correct* PWM načinu rada može se dobiti prema relaciji 10.17, a ovisi o frekvenciji radnog takta, djelitelju frekvencije radnog takta i dvostrukoj vršnoj vrijednosti registra `TCNT1`. Odaberimo djelitelj frekvencije iznosa 8 (frekvencija radnog takta jest 16 MHz). Prema relaciji (10.17), frekvencija PWM signala u *Phase Correct* PWM načinu rada rezolucije 9 bitova za sklop *Timer/Counter1* bit će:

$$F_fPWM = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot VRH1} = \frac{16000000}{8 \cdot 2 \cdot 511} = 1956,95\ \texttt{Hz}. \tag{10.27}$$

U relaciju (10.27) pokušajte uvrstiti i druge djelitelje frekvencije kako biste imali uvid o mogućim frekvencijama PWM signala za sklop *Timer/Counter1*.

Konfiguriranje sklopa *Timer/Counter1* za *Phase Correct* PWM način rada rezolucije 9 bitova provodi se u registru `TCCR1A` i registru `TCCR1B` pomoću bitova `WGM10`, `WGM11`, `WGM12` i `WGM13`, a prema tablici 15-4 u literaturi [2]. Za *Phase Correct* PWM način rada rezolucije 9 bitova, bit `WGM10` mora imati vrijednost 0, bit `WGM11` vrijednost 1, bit `WGM12` vrijednost 0, a bit `WGM13` vrijednost 0. Preostali dio konfiguracije koji smo napravili u programskom kodu 10.14 ostaje nepromijenjen. Konfiguracija sklopa *Timer/Counter1* za za *Phase Correct* PWM način rada rezolucije 9 bitova prikazana je programskim kodom 10.18. Navedenu konfiguraciju napišite u inicijalizacijsku funkciju `init()` umjesto dijela konfiguracije koji se koristiti za *Fast* PWM način rada rezolucije 10 bitova.

Programski kod 10.18: Konfiguracija sklopa *Timer/Counter1* za *Phase Correct* PWM način rada rezolucije 9 bitova

```
// postavljanje Phase Correct PWM načina rada za timer1 - 9 bit
TCCR1A |= (1 << WGM11) | (0 << WGM10);
TCCR1B |= (0 << WGM13) | (0 << WGM12);
```

Faktor vođenja D potrebno je mijenjati pomoću potenciometra koji je spojen na analogni ulaz ADC0. Rezolucija AD pretvorbe iznosi 10 bitova, a rezolucija konfiguriranog *Phase Correct* PWM načina rada jest 9 bita. Rezultat AD pretvorbe kreće se u rasponu [0, 1023], dok se vrijednosti registra `TCNT1` kreću u rasponu [0, 511]. Ako raspon AD pretvorbe podijelimo s 2, preslikat ćemo ga u raspon vrijednosti registra `TCNT1`.

U programskom kodu 10.15, obrišite sadržaj beskonačne `while` petlje i u nju napišite sljedeće naredbe:

- `OCR1A = adcRead(ADC0)/ 2;` - rezultat AD pretvorbe na kanalu ADC0 skalira se u raspon [0, 511] (dijeli se s dva) te se pridružuje u registar `OCR1A` s ciljem promjene širine PWM signala (faktora vođenja) na kanalu A (PB1 ili OC1A),

- `OCR1B = OCR1A` - vrijednost registra `OCR1A` pridružuje se u registar `OCR1B` s ciljem promjene širine PWM signala (faktora vođenja) na kanalu B (PB2 ili OC1B).

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba103.cpp` treba biti ista kao programski kod 10.19.

Programski kod 10.19: Promjena intenziteta crvene i žute LED diode pomoću potenciometra - *Phase Correct* PWM način rada rezolucije 9 bitova

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    // postavljanje Phase Correct PWM načina rada za timer1 - 9 bit
    TCCR1A |= (1 << WGM11) | (0 << WGM10);
    TCCR1B |= (0 << WGM13) | (0 << WGM12);
    // djelitelj frekvencije F_CPU / 8
    TCCR1B |= ((0 << CS12) | (1 << CS11) | (0 << CS10));
    // neinvertirajući PWM signal na PB1 (OC1A)
    TCCR1A |= (1 << COM1A1) | (0 << COM1A0);
    // invertirajući PWM signal na PB2 (OC1B)
    TCCR1A |= (1 << COM1B1) | (1 << COM1B0);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    while (1) {
        // osvježavanje faktora vođenja kroz registar OCR1A i OCR1B
        OCR1A = adcRead(ADC0) / 2;
        OCR1B = OCR1A;
    }
}
```

Prevedite datoteku `vjezba103.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, pomoću potenciometra možete mijenjati intenzitet crvene i

žute LED diode u stvarnom vremenu.

Ukoliko želite konfigurirati sklop *Timer/Counter1* za *Phase Correct* PWM način rada rezolucije 8 ili 10 bitova možete to učiniti pomoću konfiguracija prikazanih programskim kodovima 10.20 i 10.21 (prema tablici 15-4 u literaturi [2]). Četvrti način na koji se može konfigurirati sklop *Timer/Counter1* za *Phase Correct* PWM način rada prikazat ćemo u sljedećoj vježbi.

Programski kod 10.20: Konfiguracija sklopa *Timer/Counter1*: *Phase Correct* PWM način rada rezolucije 8 bitova

```
// postavljanje Fast PWM načina rada za timer1 - 8 bita
TCCR1A |= (0 << WGM11) | (1 << WGM10);
TCCR1B |= (0 << WGM13) | (0 << WGM12);
```

Programski kod 10.21: Konfiguracija sklopa *Timer/Counter1*: *Phase Correct* PWM način rada rezolucije 10 bitova

```
// postavljanje Fast PWM načina rada za timer1 - 10 bita
TCCR1A |= (1 << WGM11) | (1 << WGM10);
TCCR1B |= (0 << WGM13) | (0 << WGM12);
```

U nastavku ove vježbe za konfiguraciju sklopa *Timer/Counter1* koristit ćemo funkcije za konfiguriranje *Fast* PWM i *Phase Correct* PWM načina rada koje smo opisali u uvodnom djelu ovog poglavlja. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. Konfiguracija sklopa *Timer/Counter1* za *Fast* PWM način rada rezolucije 10 bitova prikazana je programskim kodom 10.22.

Programski kod 10.22: Konfiguracija sklopa *Timer/Counter1*: *Fast* PWM način rada rezolucije 10 bitova, djelitelj frekvencije postavljen na iznos 8, omogućeno generiranje neinvertirajućeg PWM signala na kanalu A i invertirajućeg PWM signala na kanalu B - funkcijski pristup konfiguraciji

```
// postavljanje Fast PWM načina rada za timer1 - 10 bit
timer1FastPWM10bit();
// djelitelj frekvencije F_CPU / 8
timer1SetPrescaler(TIMER1_PRESCALER_8);
// neinvertirajući PWM signal na PB1 (OC1A)
timer1OC1AEnableNonInvertedPWM();
// invertirajući PWM signal na PB2 (OC1B)
timer1OC1BEnableInvertedPWM();
```

Navedenu konfiguraciju prikazanu programskim kodom 10.22, zamijenite s konfiguracijom pomoću registara u programskom kodu 10.14. U programskom kodu 10.19, obrišite sadržaj beskonačne `while` petlje i u nju napišite sljedeće naredbe:

- `D = adcReadScale0To100(ADC0);` - čitanje rezultata AD pretvorbe na kanalu ADC0 (potenciometar), skaliranje rezultata u raspon [0, 100] te spremanje u varijablu `D` (faktor vođenja). Varijablu `D` je potrebno deklarirati izvan beskonačne `while` petlje.
- `timer1OC1ADutyCycle(D);` - postavljanje faktora vođenja D PWM signala na kanalu A.
- `timer1OC1BDutyCycle(D);` - postavljanje faktora vođenja D PWM signala na kanalu B.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba103.cpp` treba biti ista kao programski kod 10.23.

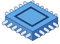
Programski kod 10.23: Promjena intenziteta crvene i žute LED diode pomoću potenciometra - *Fast* PWM način rada rezolucije 10 bitova - funkcijski pristup

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    // postavljanje Fast PWM načina rada za timer1 - 10 bit
    timer1FastPWM10bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);
    // neinvertirajući PWM signal na PB1 (OC1A)
    timer1OC1AEnableNonInvertedPWM();
    // invertirajući PWM signal na PB2 (OC1B)
    timer1OC1BEnableInvertedPWM();
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D;
    while (1) {
        // AD rezultat skalira se u raspon [0, 100.0]
        D = adcReadScale0To100(ADC0);
        // osvježavanje faktora vođenja kroz registar OCR1A i OCR1B
        timer1OC1ADutyCycle(D);
        timer1OC1BDutyCycle(D);
    }
}
```

Prevedite datoteku `vjezba103.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, pomoću potenciometra možete mijenjati intenzitet crvene i žute LED diode u stvarnom vremenu.

Za konfiguriranje sklopa *Timer/Counter1* u *Phase Correct* PWM načinu rada rezolucije 9 bitova, u programskom kodu 10.23 potrebno je naredbu `timer1FastPWM10bit();` zamijeniti naredbom `timer1PhaseCorrectPWM9bit();`. Prevedite datoteku `vjezba103.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba103.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 10.4

Napišite program koji će mijenjati intenzitet crvene LED diode pomoću neinvertirajućeg PWM signala koji ćete generirati na digitalnom izlazu PB1 (OC1A) razvojnog okruženja s mikroupravljačem ATmega328P samo ako nije pritisnuto tipkalo T1. Intenzitet crvene LED diode mijenjajte pomoću potenciometra spojenog na analogni ulaz ADC0 (PC0). Sklop *Timer/Counter1* konfigurirajte za *Phase Correct* PWM način rada, a frekvenciju PWM signala podesite tako da iznosi 1 Hz, 10 Hz, 25 Hz i 50 Hz (svaku frekvenciju posebno testirajte). Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1 (OC1A), a tipkalo T1 spojeno je na digitalni ulaz PD4 mikroupravljača ATmega328P. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba104.cpp`. Omogućite prevođenje datoteke `vjezba104.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba104.cpp` prikazan je programskim kodom 10.24.

Programski kod 10.24: Početni sadržaj datoteke `vjezba104.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulazni pin (tipkalo T1)
    pullUpOn(D4); // uključi pull up na pinu PD4

}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D; // faktor vođenja
    while (1) {

    }
}
```

Cilj ove vježbe jest mijenjati intenzitet crvene LED diode pomoću PWM signala točno definirane frekvencije pomoću sklopa *Timer/Counter1* u *Phase Correct* PWM načinu rada. Kako smo u prošloj vježbi naučili, crvena LED dioda spojena je na digitalni pin PB1 na kojem se PWM signal generira pomoću kanala A. Do sada smo mogli zamijetiti da postojećim načinima konfiguracije sklopa *Timer/Counter1* u *Phase Correct* PWM načinu rada na raspolaganju imamo samo 18 različitih iznosa frekvencije PWM signala (6 djelitelja frekvencije i 3 različite rezolucije: 8, 9 i 10 bitova) koju možemo postići. U ovoj vježbi potrebno je ostvariti točno zadane frekvencije PWM signala: 1 Hz, 10 Hz, 25 Hz i 50 Hz. U teorijskom dijelu ovog poglavlja, spomenuli smo kako se sklop *Timer/Counter1* može konfigurirati tako da je vršna vrijednost registra `TCNT1` proizvoljna i podešava se pomoću registra `ICR1`. Na taj način raspon brojanja sklopa *Timer/Counter1* u registru `TCNT1` može se podesiti od 0 do 65535. U relaciji (10.17) za frekvenciju PWM signala, varijabla $VRH1$ će poprimiti vrijednost koju smo postavili u registar `ICR1`:

$$F_fPWM = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot ICR1} \tag{10.28}$$

Prema tome, ako želimo postiži zadanu frekvenciju, relaciju (10.28) je potrebno izraziti na sljedeći način:

$$ICR1 = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot F_fPWM} \tag{10.29}$$

Vrijednost registra `ICR1` za frekvenciju 1 Hz, uz djelitelj frekvencije 256 prema relaciji (10.29) jest:

$$ICR1 = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot F_fPWM} = \frac{16000000}{256 \cdot 2 \cdot 1} = 31250. \tag{10.30}$$

Izračunata vrijednost 31250 može se pohraniti u registar `ICR1`. Uvjerite se da s manjim djeliteljima frekvencije to neće biti moguće.

Vrijednost registra ICR1 za frekvenciju 10 Hz, uz djelitelj frekvencije 256 prema relaciji (10.29) jest:

$$ICR1 = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot F_fPWM} = \frac{16000000}{256 \cdot 2 \cdot 10} = 3125. \qquad (10.31)$$

Izračunata vrijednost 3125 može se pohraniti u registar ICR1. U slučaju frekvencije 10 Hz, djelitelj frekvencije je mogao biti najmanje 64, a vrijednost registra ICR1 bi tada bila 12500.

Vrijednost registra ICR1 za frekvenciju 25 Hz, uz djelitelj frekvencije 256 prema relaciji (10.29) jest:

$$ICR1 = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot F_fPWM} = \frac{16000000}{256 \cdot 2 \cdot 25} = 1250. \qquad (10.32)$$

Izračunata vrijednost 1250 može se pohraniti u registar ICR1. U slučaju frekvencije 25 Hz, djelitelj frekvencije je mogao biti najmanje 8, a vrijednost registra ICR1 bi tada bila 40000.

Vrijednost registra ICR1 za frekvenciju 50 Hz, uz djelitelj frekvencije 256 prema relaciji (10.29) jest:

$$ICR1 = \frac{F_CPU}{PRESCALER1 \cdot 2 \cdot F_fPWM} = \frac{16000000}{256 \cdot 2 \cdot 50} = 625. \qquad (10.33)$$

Izračunata vrijednost 625 može se pohraniti u registar ICR1. U slučaju frekvencije 50 Hz, djelitelj frekvencije je mogao biti najmanje 8, a vrijednost registra ICR1 bi tada bila 20000.

Iako bi djelitelj frekvencije mogli mijenjati za svaku zadanu frekvenciju, postavit ćemo ga na 256 sukladno proračunu u relacijama (10.30) - (10.33). Prikažimo sada postupak konfiguriranja sklopa *Timer/Counter1* za *Phase Correct* PWM način rada proizvoljne rezolucije definirane registrom ICR1. Za konfiguraciju sklopa *Timer/Counter1* koristit ćemo funkcije za konfiguraciju koje smo opisali u uvodnom djelu ovog poglavlja. Navedene funkcije nalaze se u zaglavlju `timer.h` koje se u programski kod uključuju pomoću naredbe `#include "Timer/timer.h"`. Konfiguriranje sklopa *Timer/Counter1* za *Phase Correct* PWM način rada proizvoljne rezolucije definirane registrom ICR1 provodi se funkcijom `timer1PhaseCorrectPWMICR1(uint16_t top)` koja kao argument prima vrijednost registra ICR1. Vrijednost argumenta navedene funkcije za frekvenciju 1 Hz iznosi 31250 prema relaciji (10.30). Djelitelj frekvencije radnog takta iznosa 256 konfigurira se funkcijom `timer1SetPrescaler(TIMER1_PRESCALER_256);`. Omogućenje generiranje neinvertirajućeg PWM signala sklopa *Timer/Counter1* na kanalu A postiže se funkcijom `timer1OC1AEnableNonInvertedPWM();`. Konfiguracija sklopa *Timer/Counter1* prema navedenom opisu prikazana je programskim kodom 10.25.

Programski kod 10.25: Konfiguracija sklopa *Timer/Counter1*: *Phase Correct* PWM način rada rezolucije definirane registrom ICR1, djelitelj frekvencije postavljen na iznos 256, omogućeno generiranje neinvertirajućeg PWM signala na kanalu A, frekvencija PWM signala je 1 Hz - funkcijski pristup konfiguraciji

```
// postavljanje Phase Correct PWM načina rada za timer1 - ICR1
timer1PhaseCorrectPWMICR1(31250); // f =  1 Hz
// djelitelj frekvencije F_CPU / 8
timer1SetPrescaler(TIMER1_PRESCALER_256);
// neinvertirajući PWM signal na PB1 (OC1A)
timer1OC1AEnableNonInvertedPWM();
```

Konfiguraciju prikazanu programskim kodom 10.25 napišite u inicijalizacijsku funkciju `init()`. PWM signal generira se uvjetno: ako nije pritisnuto tipkalo T1. Ako je tipkalo T1 pritisnuto, tada se PWM signal ne smije generirati na pinu PB1 (OC1A).

U programski kod 10.24 unutar beskonačne `while` petlje provedite sljedeće korake:

- otvorite `if else` uvjetovano grananje,
- kao uvjet `if` grananja postavite `if (digitalRead(D4))` - ovaj uvjet bit će zadovoljen ako tipkalo T1 nije pritisnuto inače neće biti zadovoljen.
- ako je uvjet zadovoljen (`if` blok), pozovite funkciju `timer1OC1AEnableNonInvertedPWM();` kojom se omogućuje generiranje neinvertirajućeg PWM signala na kanalu A,
- ako uvjet nije zadovoljen (`else` blok), pozovite funkciju `timer1OC1ADisable();` kojom se onemogućuje generiranje neinvertirajućeg PWM signala na kanalu A.

Preostaje nam još samo provesti promjenu faktora vođenja pomoću potenciometra. U programski kod 10.24 unutar beskonačne `while` petlje napišite sljedeće naredbe:

- `D = adcReadScale0To100(ADC0);` - čitanje rezultata AD pretvorbe na kanalu ADC0 (potenciometar), skaliranje rezultata u raspon [0, 100] te spremanje u varijablu `D` (faktor vođenja),
- `lcdHome();` - pozicioniranje kursora u prvi redak i prvi stupac za ispis faktora vođenja,
- `lcdprintf("D = %.1f%%      \n", D);` - ispis teksta `D =`, faktora vođenja u %, brisanje sljedećih 6 mjesta pomoću znakova *white space* kako bismo osigurali brisanje starog teksta te pozicioniranje u novi redak.
- `lcdprintf("f = %dHz", F_CPU / 256 / 2 / ICR1);` - ispis frekvencije PWM signala u drugom retku. Drugi argument funkcije `lcdprintf()` jest izraz za frekvenciju PWM signala dan relacijom (10.17). Primijetite da kada ne brišemo tekst na LCD displeju, već ispisujemo fiksni i promjenjivi dio tekst na ispravne pozicije, da tekst na LCD displeju ne treperi.
- `timer1OC1ADutyCycle(D);` - postavljanje faktora vođenja D PWM signala na kanalu A.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba104.cpp` treba biti ista kao programski kod 10.26.

Programski kod 10.26: Promjena intenziteta crvene LED diode pomoću potenciometra - *Phase Correct* PWM način rada proizvoljne rezolucije definirane registrom `ICR1` - funkcijski pristup

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulazni pin (tipkalo T1)
    pullUpOn(D4); // uključi pull up na pinu PD4
    // postavljanje Phase Correct PWM načina rada za timer1 - ICR1
    timer1PhaseCorrectPWMICR1(625); // f = 1 Hz
    // djelitelj frekvencije F_CPU / 256
    timer1SetPrescaler(TIMER1_PRESCALER_256);
    // neinvertirajući PWM signal na PB1 (OC1A)
    timer1OC1AEnableNonInvertedPWM();
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
```

```
    float D; // faktor vođenja
    while (1) {
        // ako nije pritisnuto tipkalo S1
        if (digitalRead(D4)) {
            // neinvertirajući PWM signal na PB1 (OC1A)
            timer1OC1AEnableNonInvertedPWM();
            } else {
            // isključi PWM signal na PB1 (OC1A)
            timer1OC1ADisable();
        }
        // AD rezultat skalira se u raspon [0, 100.0]
        D = adcReadScale0To100(ADC0);
        // ispis na LCD displej (sintaksa funkcije printf())
        lcdHome();
        lcdprintf("D = %.1f%%      \n", D);
        lcdprintf("f = %dHz", F_CPU / 256 / 2 / ICR1);
        // osvježavanje faktora vođenja kroz registar OCR1A
        timer1OC1ADutyCycle(D);
    }
}
```

Prevedite datoteku `vjezba104.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, pomoću potenciometra možete mijenjati faktor vođenja na pinu PB1 na kojem je spojena crvena LED dioda u stvarnom vremenu te ispisivati faktor vođenja i frekvenciju na LCD displej. Zbog toga što je frekvencija PWM signala niska, crvena LED dioda će promjenom stanja potenciometra unutar jedne sekunde ($f = 1$ Hz) mijenjati postotak uključenosti. Ovaj primjer smo pokazali kako takvu frekvenciju ne biste koristili za potrebu promjene intenziteta LED diode.

Testirajmo sada slučaj kada je frekvencija jednaka 10 Hz. U programskom kodu 10.26 poziv funkcije `timer1PhaseCorrectPWMICR1(31250);` zamijenite s pozivom funkcije `timer1PhaseCorrectPWMICR1(3125);` prema relaciji (10.31). Prevedite datoteku `vjezba104.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. I ovaj primjer pokazuje kako se još uvijek preniskom frekvencijom ne može mijenjati intenzitet LED diode.

Testirajmo sada slučaj kada je frekvencija jednaka 25 Hz. U programskom kodu 10.26 poziv funkcije `timer1PhaseCorrectPWMICR1(3125);` zamijenite s pozivom funkcije `timer1PhaseCorrectPWMICR1(1250);` prema relaciji (10.32). Prevedite datoteku `vjezba104.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. I dalje je frekvencija preniska u odnosu na tromost oka.

Naposljetku, testirajmo slučaj kada je frekvencija jednaka 50 Hz. U programskom kodu 10.26 poziv funkcije `timer1PhaseCorrectPWMICR1(1250);` zamijenite s pozivom funkcije `timer1PhaseCorrectPWMICR1(625);` prema relaciji (10.33). Prevedite datoteku `vjezba104.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Frekvencija 50 Hz dovoljno je visoka da oko ne vidi promjene u uključenju i isključenju LED diode, odnosno da dobijemo efekt glatke promjene intenziteta crvene LED diode. Dodatno, kada pritisnete tipkalo T1, crvena LED dioda neće svijetliti jer je onemogućeno generiranje PWM signala na kanal A (OC1A).

Perifernim vidom je moguće vidjeti da LED dioda treperi i na frekvenciji PWM signala iznosa 50 Hz, stoga preporučujemo da frekvenciju dignete iznad 1000 Hz.

Zatvorite datoteku `vjezba104.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 11

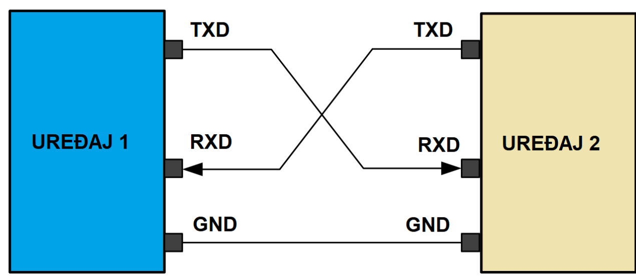
Univerzalna asinkrona serijska komunikacija

Mikroupravljači često moraju razmijeniti podatke s drugim mikroupravljačima ili drugim uređajima poput računala, GSM-a (engl. *Global System for Mobile communication*) i slično. Prijenos podataka se najčešće odvija serijskom tehnikom gdje se podaci šalju bit po bit. Mikroupravljač ATmega328P opremljen je sklopovljem USART (engl. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*) koje omogućuje sinkroni i asinkroni serijski prijenos podataka. Sklopovlje USART se najčešće koristi za asinkroni prijenos podataka. Asinkroni prijenos podataka znači da se podaci šalju unutar podatkovnog okvira koji ima svoj početak (start bitove) i svoj kraj (stop bitove). Nadalje, kod asinkkrone komunikacije nema izvora takta koji sinkronizira prijenos podataka između dva uređaja.

Univerzalnu asinkronu serijsku komunikaciju omogućuje sklopovlje UART (engl. *Universal Asynchronous Serial Receiver and Transmitter*). UART sklopovlje koristi dva pina:

- RXD - pin pomoću kojeg se primaju podaci (engl. *receiver*),
- TXD - pin pomoću kojeg se šalju podaci (engl. *transmitter*).

Pomoću pinova RXD i TXD omogućena je dvosmjerna (engl. *full-duplex*) komunikacija. To znači da dva uređaja u isto vrijeme mogu primati i slati podatke pomoću dviju žice spojena na dva pina: RXD i TXD. Sklopovlje UART ima odvojeno sklopovlje za primanje i slanje podataka pa je takva dvosmjerna komunikacija moguća. Shema povezivanja dvaju uređaja sa sklopovljem UART prikazana je na slici 11.1. Povezivanje dvaju uređaja provodi se uvijek na isti način: RXD pin prvoga uređaja spaja se na TXD pin drugoga, a RXD drugoga uređaja spaja se na TXD prvoga uređaja (slika 11.1). Referentni potencijali `Gnd` dvaju uređaja koji komuniciraju sklopovljem UART moraju biti spojeni zajedno.



Slika 11.1: Shema povezivanja dvaju uređaja sa sklopovljem UART

Zbog asinkrone serijske komunikacije dva uređaja sa sklopovljem UART koja razmjenjuju podatke moraju imati jednaku konfiguraciju parametara. Parametri sklopovlja UART jesu [1]:

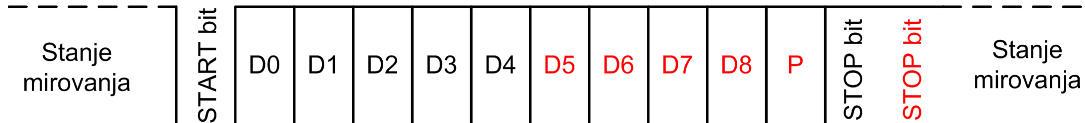
- Brzina prijenosa podataka (engl. *baude rate*) - sklopovlje UART ima registar kojim se konfigurira brzina prijenosa podataka u bitovima po sekundi (b/s). Često su korištene brzine prijenosa od 9600 do 115200 b/s. Brzina prijenosa ovisi o radnom taktu uređaja (mikroupravljača, računala, modema) pa su između brzina prijenosa od 9600 do 115200 b/s dostupne samo neke brzine (najčešće one koje su višekratnik frekvencije radnog takta).

- Broj podatkovnih bitova - broj podatkovnih bitova može biti 5, 6, 7, 8 i 9. Najčešće se kao broj podatkovnih bitova koristi 8 jer je to 1 bajt (B) podataka.

- Paritetni bit (engl. *parity bit*) - omogućuje detekciju jednostrukih grešaka u prijenosu podataka. Paritetni bit može biti omogućen ili onemogućen. Ako je omogućen, tada se pomoću njega može osigurati parni ili neparni paritet.

- *Stop* bit - bit koji označava kraj jednog podatka. Broj stop bitova može biti 1 ili 2.

Podatkovni okvir kod univerzalne asinkrone serijske komunikacije prikazan je na slici 11.2.



Slika 11.2: Podatkovni okvir kod univerzalne asinkrone serijske komunikacije

Kada nema prijenosa podataka pomoću UART sučelja, pinovi RXD i TXD su u visokom stanju. Ovo stanje se još zove stanje mirovanja (engl. *idle*). Svako slanje podataka između dvaju uređaja počinje tako da uređaj koji šalje podatak svoj TXD pin postavlja u logičku nulu. Ovaj prijelaz iz logičke jedinice u logičku nulu naziva se *Start* bit. Nakon *Start* bita, šalju se bitovi podatka D0, D1, ..., D9. Broj podatkovnih bitova može se konfigurirati, a najčešće se šalje podatak širine 8 bitova (1 B). Iza podatkovnih bitova, ako je omogućen, šalje se paritetni bit. Komunikacija završava *Stop* bitom/bitovima. *Stop* bitovi uvijek su logičke jedinice. Trajanje bitova podatkovnog okvira sa slike 11.2 ovisi o brzini prijenosa podataka. Najčešće postavke UART sučelja jesu:

- brzina prijenosa podataka: ovisi o uređaju s kojim se razmjenjuju podaci (npr. 9600, 19200, ...),

- broj podatkovnih bitova: 8 (najčešće su to ASCII znakovi),

- paritetni bit: onemogućen,

- *Stop* bit: 1.

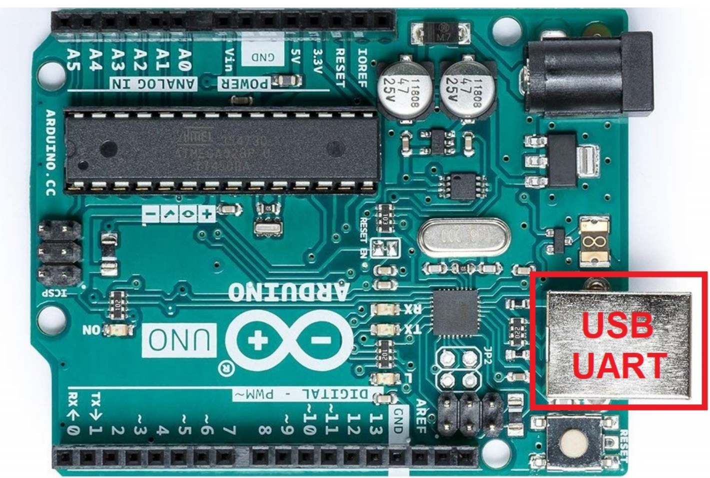
Za slanje 1 B (8 bitova) podataka, potrebno je poslati ukupno 10 bitova (1 *Start* bit + 8 bitova podataka + 1 *Stop* bit = 10 bitova). Zbog asinkronog načina prijenosa, dva uređaja koja razmjenjuju podatke moraju biti jednako konfigurirana. U suprotnom tumačenje podatka neće biti ispravno.

UART komunikacija najčešće se koristi za komunikaciju mikroupravljača s računalom. Nekada su računala imala serijske portove na koje su se spajali uređaji koji kominiciraju pomoću UART komunikacije (RS232 standard). Današnja računala nemaju takve portove pa komunikacija mikroupravljača s računalom zahtjeva korištenje prilagodnih međusklopova koji

TTL razine mikroupravljača pretvaraju na USB razine. Neki od poznatijih integriranih krugova koji omogućuju ovu pretvorbu jesu CP2102 i FT232. Računalo ove uređaje promatra kao virtualni COM port koji radi na RS232 standardu. Također, AVR mikroupravljači koji na sebi imaju USB sklopovlje (ATmega32U4, ATmega16u2, ...) mogu emulirati virtualni COM port.

11.1 Vježbe - UART komunikacija

Mikroupravljač ATmega328P posjeduje sklopovlje USART (engl. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*). Uobičajeno je ovo sklopovlje koristiti samo za asinkronu komunikaciju pa će se ove vježbe fokusirati isključivo na asinkronu komunikaciju (UART sklopovlje). Prijenos podataka serijskom komunikacijom pomoću mikroupravljača ATmega328P obavlja se pomoću pina za primanje podataka RXD (PD0) te pina za slanje podataka TXD (PD1). Kada na mikroupravljaču koristite UART sklopovlje za serijsku komunikaciju, pinovi PD0 i PD1 se ne mogu koristiti u druge svrhe. Razvojnom okruženju sa slike 2.1 sastoji se od Arduino UNO razvojnog okruženja i elektroničkog modula. Arduino UNO razvojno okruženje na sebi ima USB konektor koji se može koristiti za UART komunikaciju (slika 11.3).



Slika 11.3: USB priključak na Arduino UNO razvojnom okruženju za UART komunikaciju

Prema specifikaciji Arduino UNO razvojnog okruženja koje se nalazi na stranici `https://store-usa.arduino.cc/products/arduino-uno-rev3`, Arduino UNO je opremljen s mikroupravljačem ATmega16u2 koji skluži kao međusklop za USB $\leftrightarrow$ UART (TTL) pretvorbu. Mikroupravljač ATmega16u2 na Arduino UNO razvojnom okruženju povezan je s pinovima RXD (PD0) i TXD (PD1) mikroupravljača ATmega328P. USB kabelom potrebno je Arduino UNO razvojno okruženje povezati s računalom. Prema shemi na slici 4.1 na digitalnom izlazu PD1 (TXD) spojeni su zujalica i relej preko kratkospojnika JP5. Kako ovi uređaji ne bi imali negativan utjecaj na prijenos podataka, kratkospojnik JP5 potrebno je skinuti s razvojnog okruženja.

Mikroupravljač prima podatke u skladu s podatkovnim okvirom sa slike 11.2 pomoću digitalnog pina RXD, a šalje podatke pomoću digitalnog pina TXD. Podaci koji se prihvaćaju i šalju putem UART komunikacije su najčešće širine 8 bitova. Radi se u pravilu o znakovnim nizovima kodiranim ASCII kodom. Slanje i primanje podataka je automatizirano. Nakon pojave *Start* bita na digitalnom pinu RXD, mikroupravljač generira prekid pomoću prekidnog vektora `USART_RX_vect`. U tom trenutku se podatkovni okvir sa slike 11.2 dekodira te se podatkovni bitovi spremaju u registar `UDR0`. Spremljeni 8-bitni podatak (znak u ASCII kodu) u registru `UDR0` potrebno je odmah pročitati i spremiti u neko interno polje znakova jer ukoliko se to

ne napravi odmah, potencijalno će aktualni znak u registru `UDR0` prepisati novi znak jer se serijskom komunikacijom šalju znakovni nizovi (znak po znak). Isti registar `UDR0` se koristi za slanje podataka. Kada u registar `UDR0` upišemo 8-bitni podatak, UART sučelje započinje automatiziranu proceduru slanja podatka pomoću digitalnog pina TXD. UART sklopovlje omogućuje istovremenu dvosmjernu komunikaciju (engl. *full duplex*) što zači da se u isto vrijeme podaci mogu i primati i slati. Razlog tomu jest taj što je registar `UDR0` dvostruki registar koji ima odvojen memorijski prostor za primanje i slanje podataka, ali se dohvaća pomoću iste adrese.

Rad UART sklopovlja konfigurira se pomoću registara `UCSR0A`, `UCSR0B` i `UCSR0C`. Konfiguracija se odnosi na broj podatkovnih bitova, paritetni bit, broj *Stop* bitova, omogućavanje prekida kada stigne podatak, omogućenje udvostručenja brzine prijenosa podataka i drugo. Više detalja o konfiguraciji registara `UCSR0A`, `UCSR0B` i `UCSR0C` pogledajte u literaturi [2] u poglavlju 19.

Brzinu prijenosa podataka možemo konfigurirati pomoću registra `UBRR0`, a računa se prema izrazu [2]:

$$BAUD_RATE = \frac{F_CPU}{16 \cdot (\texttt{UBRR0} + 1)}. \tag{11.1}$$

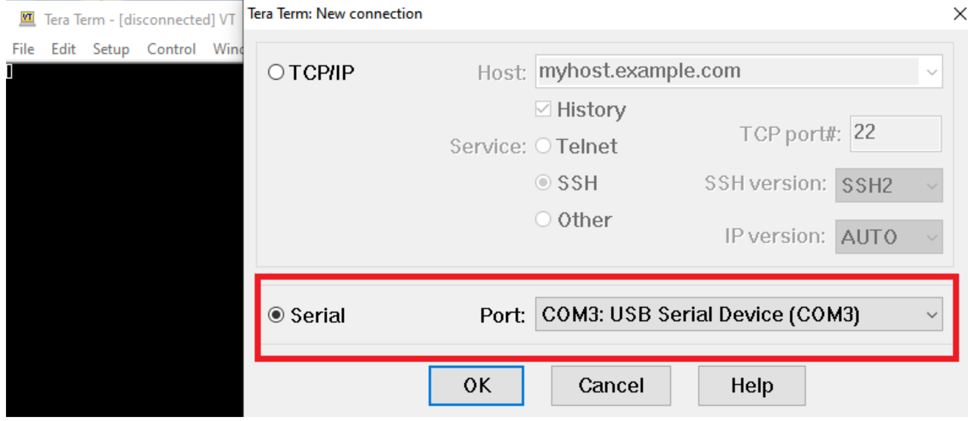
Da bismo postigli željenu brzinu prijenosa podataka, u registar `UBRR0` je potrebno upisati konstantu koja se može dobiti pomoću relacije (11.2) na sljedeći način:

$$\texttt{UBRR0} = \frac{F_CPU}{16 \cdot BAUD_RATE} - 1. \tag{11.2}$$

Rezultat dobiven pomoću relacije (11.2) mora biti cjelobrojan kako se ne bi pojavila greška u prijenosu podatka. Vrijednost registra `UBRR0` se u pravilu izračunava za standardne brzine prijenosa: 2400 b/s, 4800 b/s, 9600 b/s, 19200 b/s, 28800 b/s, 38400 b/s, 57600 b/s, 76800 b/s, 115200 b/s itd.

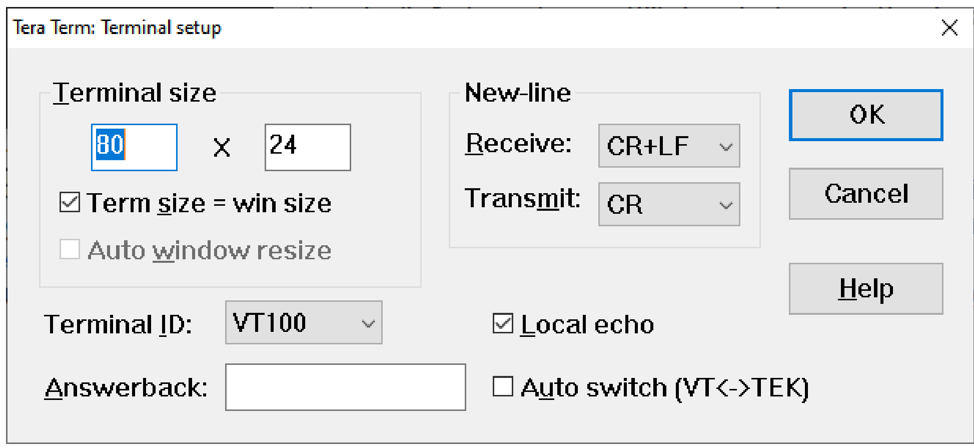
U svrhu testiranja UART komunikacije u praksi se koriste terminali. Jedan od poznatijih terminala za testiranje serijske komunikacije jest *Tera Term* koji možete preuzeti sa stranice `https://ttssh2.osdn.jp/`. Preuzmite *Tera Term* i instalirajte ga na svom računalu.

Kada pokrenete terminal (aplikaciju) *Tera Term* otvorit će se prozor sa slike 11.4. Vrsta komunikaciju koju je potrebno odabrati jest serijska komunikacija (odabir `Serial`). Preduvjet da postoji neki od COM portova jest taj da razvojno okruženje Arduino UNO povežete USB kablom na računalu pomoću priključka koji je crveno označen na slici 11.3. U padajućem izborniku odabira `Serial` prikazuju se svi dostupni COM portovi na računalu, stoga ako ih imate više, potrebno je odabrati onaj koji je dodijeljen Arduino UNO razvojnom okruženju. U našem slučaju dodijeljen nam je naziv porta COM3. Ovo ime neće biti isto na svakom računalu, no važno ga je upamtiti.



Slika 11.4: Pokretanje terminala *Tera Term*

Ako ste slijedili gore navedene upute, računalo će otvoriti COM port kojim će se moći komunicirati pomoću mikroupravljača. Sljedeći korak jest konfiguracija postavki terminala *Tera Term*. Postavke namjestite da budu istovjetne postavkama na slici 11.5. Konfiguracija postavki terminala *Tera Term* sa slike 11.5 otvara se tako da u izborniku `Setup` odaberete podizbornik `Terminal...`.



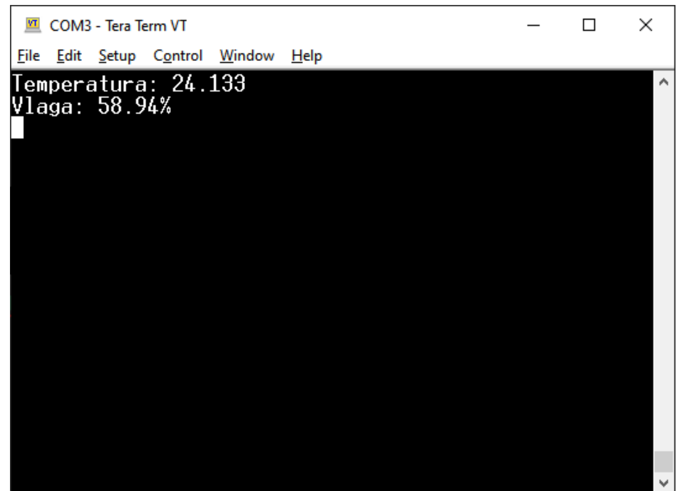
Slika 11.5: Konfiguracija postavki terminala *Tera Term*

Da bi dva uređaja ispravno komunicirala pomoću UART komunikacije, postavke COM porta moraju biti iste na oba uređaja. Podešavanje postavki serijskog porta u terminalu *Tera Term* prikazano je na slici 11.6.

Slika 11.6: Podešavanje postavki serijskog porta u terminalu *Tera Term*

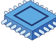
Postavke serijskog porta sa slike 11.6 možete otvoriti tako da u izborniku `Setup` odaberete podizbornik `Serial port...`. U prozoru sa slike 11.6 moguće je odabrati ime COM porta (COM3), brzinu prijenosa podataka (9600 b/s), broj podatkovnih bitova (8), paritetni bit (isključen), broj stop bitova (1 bit) i kontrolu toka podataka (isključena). Najčešće ćemo mijenjati samo brzinu prijenosa podataka koja će kroz vježbe biti uobičajeno 19200 b/s, a testirat ćemo i veće brzine prijenosa podatka.

Primljene tekstualne poruke u terminalu *Tera Term* prikazane su na slici 11.7. Terminali za serijsku komunikaciju korisni su i za ispravljanje pogrešaka u programskom kodu, jer u kodu možete definirati točke provjere u kojima ćete na terminal ispisati određenu poruku. Ako neku od poruka niste primili, znate gdje je program "zaglavio".



Slika 11.7: Prikaz tekstualnih poruka u terminalu *Tera Term*

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `UART.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `UART.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\UART\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Univerzalna asinkrona serijska komunikacija`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

 Vježba 11.1

Napišite program koji će slati proizvoljne tekstualne poruke UART komunikacijom mikroupravljača ATmega328P. Broj poruka je proizvoljan, a potrebno ih je slati s razmakom od jedne sekundi u `while` beskonačnoj petlji. Poruke je potrebno prikazivati na računalu u terminalu *Tera Term*. Znakovni niz koji pomoću mikroupravljača šaljemo na računalo UART komunikacijom mora biti zaključen s *Carriage Return* znakom (`'\r'`). Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*.

U projektnom stablu otvorite datoteku `vjezba111.cpp`. Omogućite prevođenje datoteke `vjezba111.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba111.cpp` prikazan je programskim kodom 11.1.

Programski kod 11.1: Početni sadržaj datoteke `vjezba111.cpp`

```
#include "AVR/avrlib.h"
#include <util/delay.h>

void init() {

}
```

```
int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) {

    }
}
```

U ovoj vježbi cilj nam je naučiti slati poruke putem UART sučelja mikroupravljača ATmega328P. Kako bi se olakšalo slanje i primanje podataka pomoću asinkrone serijske komunikacije mikroupravljača ATmega328P, autori su definirali brojne korisne funkcije za UART sučelje. Funkcije su definirane u zaglavlju `uart.h`, a u programski kod 11.1 zaglavlje `uart.h` uključite naredbom `#include "UART/uart.h"`.

U zaglavlju `uart.h` nalaze se sljedeće funkcije koje služe za inicijalizaciju UART komunikacije i slanje različitih vrsta poruka/znakova:

- `void uartInit(uint32_t baudRate)` - funkcija koja služi za inicijalizaciju UART sklopovlja. Kao argument, funkcija prima cjelobrojnu varijablu `baudRate` što predstavlja brzinu prijenosa podataka. Ovom se funkcijom postavljaju sljedeći parametri:
  - brzina prijenosa podataka koja se šalje kao argument funkcije: (npr. 9600, 19200 itd.),
  - broj podatkovnih bitova: 8,
  - paritetni bit: onemogućen,
  - *Stop* bit: 1,
  - prekid za primljene poruke: omogućen,
  - digitalni pinovi RXD i TXD: omogućeni.

  Brzina prijenosa podataka je jedini parametar koji se može mijenjati, dok su ostali parametri fiksni.

- `void uartputchar(char uartChar)` - funkcija kojom se šalje jedan znak putem UART komunikacije. Kao argument, funkcija prima znak, a sintaksa funkcije `uartputchar()` istovjetna je sintaksi funkcije `putchar()`.

- `void uartputs(const char *uartMessage)` - funkcija kojom se šalje string (tekstualna poruka) putem UART komunikacije. Kao argument, funkcija prima pokazivač na string, a sintaksa funkcije `uartputs()` istovjetna je sintaksi funkcije `puts()`.

- `uartprintf(const char *format, arg1, arg2, ...)` - funkcija kojom se šalje string (tekstualna poruka) putem UART komunikacije. Sintaksa funkcije `uartprintf()` istovjetna je sintaksi funkcije `printf()`. Za argumente funkcije vrijedi:
  - `const char *format` - string u funkciji `uartprintf()` koji sadrži tekst koji će se poslati putem UART komunikacije. String može sadržavati ugrađene oznake formata (formate ispisa (engl. *format tags*)) koji se zamjenjuju tekstualnim zapisom liste argumenata `arg1, arg2, ...` u zadanom formatu.

Pažljivi čitatelj će primijetiti da su sve funkcije pisane stilom *camelCase* osim funkcija `uartputchar()`, `uartputs()` i `uartprintf()`. Razlog tomu je što smo htjeli da te funkcije poprime naziv kao srodne funkcije `putchar()`, `puts()` i `printf()`.

Pokažimo sada nekoliko primjera korištenja navedenih funkcija:

- `uartInit(115200)` - UART sklopovlje inicijalizirano je sa svim navedenim fiksnim parametrima (8-bitni podaci, 1 stop bit, ...) i brzinom prijenosa podataka iznosa 115200 b/s,
- `uartputchar('A')` - UART komunikacijom šalje se znak `A`,
- `uartputs("ATmega328P")` - UART komunikacijom šalje se tekstualna poruka `ATmega328P`,
- `uartprintf("Napon: %.2f", 5.12)` - UART komunikacijom šalje se tekstualna poruka `Napon: 5.12`.

Funkcija `uartprintf()` može se koristiti za slanje jednog znaka te za slanje tekstualne poruke s fiksnim tekstom i tekstom koji ima uključene ugrađene oznake formata. Postavlja se pitanje, zašto smo uopće napravili funkcije `uartputchar()` i `uartputs()`? Razlog tomu je što ove dvije funkcije zauzimaju značajno manje programske memorije od funkcije `uartprintf()`. Stoga, ako UART komunikacijom šaljete samo fiksni tekst, tada se preporuča koristiti funkcije `uartputchar()` i `uartputs()`. U svim ostalim slučajevima koristite funkciju `uartprintf()`.

U ovoj vježbi postavit ćemo brzinu prijenosa podataka na 19200 b/s. U programski kod 11.1 u funkciju `init()` upišite poziv funkcije `uartInit(19200)`. Zahtjevi vježbe su slanje proizvoljnog broja proizvoljnih tekstualnih poruka UART komunikacijom s razmakom od jedne sekunde. U programski kod 11.1 u `while` beskonačnu petlju upišite sljedeće naredbe:

- `uartprintf("%d. poruka\r", brojPoruka++);` - funkcija koja će u terminalu *Tera Term* ispisivati redni broj ispisanih poruka u jednom prolazu kroz beskonačnu `while` petlju prema sintaksi funkcije `printf()` (u prvoj iteraciji ispisat će se tekst `1. poruka`"). Poruka je zaključana sa specijalnim znakom `'\r'` koja će omogućiti pozicioniranje ispisa u terminalu *Tera Term* na početak novoga retka. Broj poruka broji se varijablom `brojPoruka` koja se uvećava za 1 kroz argument funkcije `uartprintf()`. Varijablu `brojPoruka` deklarirajte i inicijalizirajte izvan beskonačne `while` petlje na sljedeći način: `int brojPoruka = 1;`.
- `_delay_ms(1000);` - kašnjenje programa za 1 sekundu,
- `uartputchar('A');` - funkcija koja će u terminalu *Tera Term* ispisati znak `'A'` prema sintaksi funkcije `putchar()`. Ovom funkcijom nije moguće poslati dva znaka pa je specijalni znak `'\r'` potrebno poslati posebno (vidjeti sljedeću natuknicu).
- `uartputchar('\r');` - funkcija koja će u terminalu *Tera Term* ispisati znak `'\r'` prema sintaksi funkcije `putchar()`, odnosno pozicionirati će ispis u terminalu *Tera Term* na početak novoga retka.
- `_delay_ms(1000);` - kašnjenje programa za 1 sekundu,
- `uartputs("ATmega328P ima 32 kB programske memorije!\r");` - funkcija koja će u terminalu *Tera Term* ispisati tekst `ATmega328P ima 32 kB programske memorije!` prema sintaksi funkcije `puts()`. Poruka je zaključana sa specijalnim znakom `'\r'` koja će omogućiti pozicioniranje ispisa u terminalu *Tera Term* na početak novoga retka.
- `_delay_ms(1000);` - kašnjenje programa za 1 sekundu,
- `uartprintf("Temperatura: %.2f\r", 22.34);` - funkcija koja će u terminalu *Tera Term* ispisati tekst `Temperatura: 22.34` prema sintaksi funkcije `printf()`. Poruka je zaključana sa specijalnim znakom `'\r'` koja će omogućiti pozicioniranje ispisa u terminalu

Tera Term na početak novoga retka.

- `_delay_ms(1000);` - kašnjenje programa za 1 sekundu,
- `uartprintf("Vlaga: %.2f%%\r", 57.82);` - funkcija koja će u terminalu *Tera Term* ispisati tekst `Vlaga: 57.82%` prema sintaksi funkcije `printf()`. Poruka je zaključana sa specijalnim znakom `'\r'` koja će omogućiti pozicioniranje ispisa u terminalu *Tera Term* na početak novoga retka.
- `_delay_ms(1000);` - kašnjenje programa za 1 sekundu.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba111.cpp` treba biti ista kao programski kod 11.2.

Programski kod 11.2: Ispis proizvoljnih poruka UART komunikacijom

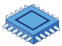
```
#include "AVR/avrlib.h"
#include <util/delay.h>
#include "UART/uart.h"

void init() {
    uartInit(19200); // inicijalizacija UART komunikacije
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    int brojPoruka = 1;
    while (1) {
        // ispis poruke sintaksom funkcije printf()
        uartprintf("%d. poruka\r", brojPoruka++);
        _delay_ms(1000);
        // ispis poruke sintaksom funkcije putchar()
        uartputchar('A');
        uartputchar('\r');
        _delay_ms(1000);
        // ispis poruke sintaksom funkcije puts()
        uartputs("ATmega328P ima 32 kB programske memorije!\r");
        _delay_ms(1000);
        // ispis poruke sintaksom funkcije printf()
        uartprintf("Temperatura: %.2f\r", 22.34);
        _delay_ms(1000);
        // ispis poruke sintaksom funkcije printf()
        uartprintf("Vlaga: %.2f%%\n\r", 57.82);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba111.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7. Podesite brzinu prijenosa podataka na 19200 b/s. Ako ste slijedili navedene korake, u terminalu će se ispisivati poruke prikazane na slici 11.8.

```
COM3 - Tera Term VT
File Edit Setup Control Window Help
1. poruka
A
ATmega328P ima 32 kB programske memorije!
Temperatura: 22.34
Vlaga: 57.82%

2. poruka
A
ATmega328P ima 32 kB programske memorije!
Temperatura: 22.34
Vlaga: 57.82%

3. poruka
A
ATmega328P ima 32 kB programske memorije!
Temperatura: 22.34
Vlaga: 57.82%

4. poruka
A
ATmega328P ima 32 kB programske memorije!
Temperatura: 22.34
Vlaga: 57.82%
```

Slika 11.8: Ispis proizvoljnih poruka UART komunikacijom - `vjezba111.cpp`

Zatvorite datoteku `vjezba111.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 11.2

Napišite program koji će primati i slati proizvoljne tekstualne poruke UART komunikacijom mikroupravljača ATmega328P na sljedeći način:

- poruke se šalju putem terminala *Tera Term*, a kraj poruke se definira tipkom *Enter*,
- prihvaćeni znakovi kao sadržaj cjelovite tekstualne poruke smještaju se u polje znakova (međuspremnik) do dolaska znaka `'\r'` koji predstavlja kraj tekstualne poruke,
- primljenu poruku ispisati na LCD displeju i poslati nazad u terminal *Tera Term* (engl. *echo*).

Znakovni nizovi koje pomoću mikroupravljača primamo i šaljemo UART komunikacijom zaključani su *Carriage Return* znakom (`'\r'`). Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba112.cpp`. Omogućite prevođenje datoteke `vjezba112.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba112.cpp` prikazan je programskim kodom 11.3.

Programski kod 11.3: Početni sadržaj datoteke `vjezba112.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"

void init() {
```

```
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) {

    }
}
```

U ovoj vježbi cilj nam je naučiti primati poruke putem UART sučelja mikroupravljača ATmega328P. U prethodnoj vježbi naučili smo kako poslati poruku pomoću mikroupravljača ATmega328P. UART komunikacijom tekstualne poruke se primaju i šalju znak po znak. Kako smo u uvodnom djelu opisali, primanje poruka odvija se pomoću prekida jer primljeni znak kao dio niza znakova potrebno odmah pokupiti kako ne bi bio prepisan sa sljedećim primljenim znakom. Kako bi se olakšalo slanje i primanje podataka pomoću asinkrone serijske komunikacije mikroupravljača ATmega328P, autori su definirali brojne korisne funkcije za UART sučelje. Funkcije su definirane u zaglavlju `uart.h`, a u programski kod 11.3 zaglavlje `uart.h` uključite naredbom `#include "UART/uart.h"`.

U zaglavlju `uart.h` deklariran je niz znakova `uartBuffer` (programski kod 11.4) koji predstavlja međuspremnik znakova (engl. *buffer*). Maksimalna duljina niza znakova međuspremnika `uartBuffer` definirana je konstantom `MESSAGE_LENGTH`. Ovoj konstanti dodijeljen je broj 100, a može se mijenjati ovisno o potrebnoj duljini poruke[1]. Zaključni znak primljene poruke definiran je konstantom `END_CHAR`. Ovoj konstanti dodijeljen je znak `'\r'`, a prema potrebi to može biti bilo koji drugi znak (npr. `'*'` `'\n'`, `':'`, ...).

Programski kod 11.4: Konstante i pokazivač koji se koriste za međuspremnik znakova `uartBuffer`

```
// zaključni znak '\r' - Carriage Return (CR) ASCII kod 0x0D
#define END_CHAR '\r'
// maksimalna duljina poruke (duljina međuspremnika)
#define MESSAGE_LENGTH 100
// deklaracija i inicijalizacija međuspremnika za dolazne poruke
char uartBuffer[MESSAGE_LENGTH + 1] = {};
volatile char *uartBufferPtr = uartBuffer; //pokazivač na uartBuffer
```

U zaglavlju `uart.h` nalaze se sljedeće funkcije koje služe za primanje različitih vrsta poruka/znakova putem UART komunikacije:

- `char uartgetchar()` - funkcija koja služi za dohvaćanje znaka iz registra `UDR0`. Kao povratnu vrijednost funkcija vraća znak (sadržaj registra `UDR0`), a sintaksa funkcije `uartgetchar()` istovjetna je sintaksi funkcije `getchar()`.

- `char* uartgets()` - funkcija koja vraća pokazivač na niz znakova, odnosno na međuspremnik `uartBuffer`. Kao povratnu vrijednost funkcija vraća adresu međuspremnika `uartBuffer`, a sintaksa funkcije `uartgets()` istovjetna je sintaksi funkcije `gets()`.

- `void uartCopyBuffer(char *usartMessage)` - funkcija koja kopira sadržaj međuspremnika `uartBuffer` u niz znakova na koji pokazuje pokazivač `*usartMessage`. Ova funkcija kao argument prima pokazivač na niz znakova `*usartMessage`.

- `bool uartIsMessageRecived()` - funkcija koja provjerava da li je primljena cjelovita

[1] Oprezno s duljinom znakova jer podatkovna memorija mikroupravljača ATmega328P može teoretski pohraniti najviše 2047 znakova. Veličina podatkovne memorije je 2 kB = 2048 B

tekstualna poruka u međuspremnik `uartBuffer` na način da provjerava da li je zadnji primljen znak jednak zaključnom znaku koji je definiran konstantom `END_CHAR`. Funkcija vraća vrijednost `false` ako nije stigla cjelovita poruka UART sučeljem, a vrijednost `true` ako je stigla cjelovita poruka (zadnji znak koji je primljen jest `END_CHAR`).

Pažljivi čitatelj će primijetiti da su sve funkcije pisane stilom *camelCase* osim funkcija `uartgetchar()` i `uartgets()`. Razlog tomu je što smo htjeli da te funkcije poprimi naziv kao funkcije koje su srodne funkcijama `getchar()` i `gets()`. Međuspremnik znakova `uartBuffer` puni se pomoću prekidne rutine prikazane programskim kodom 11.5. Punjenje međuspremnika `uartBuffer` provodi se pomoć pokazivača `*uartBufferPtr` koji je deklariran i inicijaliziran u programskom kodu 11.4. U međuspremnik se kopiraju svi znakovi osim zaključnog znaka `END_CHAR`.

Programski kod 11.5: Prekidna rutina UART komunikacije za primanje (RX) poruka

```
ISR(USART_RX_vect) {
    *uartBufferPtr = uartgetchar();
    if (*uartBufferPtr != END_CHAR) {
        uartBufferPtr++;
    }
}
```

Prekidna rutina `ISR(USART_RX_vect)` nalazi se u zaglavlju `uart.h` i nije ju potrebno mijenjati. Potrebno je imati na umu da će u programskom kodu 11.3 biti potrebno globalno omogućiti prekide. U nastavku ćemo prikazati tri načina primljene tekstualne poruke UART komunikacijom. Cjelovita poruka dostupna je ako funkcija `uartIsMessageRecived()` vraća vrijednost `true`.

Prvi način dohvaćanja primljene tekstualne poruke UART komunikacijom prikazan je programskim kodom 11.6. Unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` poruka se ispisuje na LCD displej direktnim pristupom međuspremniku `uartBuffer` jer je deklariran u globalnom području. Ovo je najjednostavniji način. Poruka je dostupna odmah.

Programski kod 11.6: Dohvaćanje primljene tekstualne poruke UART komunikacijom - 1. način

```
if (uartIsMessageRecived()) {
    // prikaz primljene poruke
    lcdprintf("%s", uartBuffer);
}
```

Drugi način dohvaćanja primljene tekstualne poruke UART komunikacijom prikazan je programskim kodom 11.7. S obzirom da ćemo koristiti pokazivač, potrebno ga je deklarirati (`char *pokPoruka;`). Unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` funkcijom `uartgets()` vraća se adresa međuspremnika `uartBuffer`. Pokazivač `*pokPoruka` se referencira na adresu međuspremnika `uartBuffer`. Tekstualna poruka koja je stigla UART komunikacijom dostupna je preko pokazivača za ispis na LCD displej.

Programski kod 11.7: Dohvaćanje primljene tekstualne poruke UART komunikacijom - 2. način

```
char *pokPoruka;
if (uartIsMessageRecived()) {
    pokPoruka = uartgets();
    // prikaz primljene poruke
    lcdprintf("%s", pokPoruka);
}
```

Treći način dohvaćanja primljene tekstualne poruke UART komunikacijom prikazan je programskim kodom 11.8. Zadnji pristup omogućuje kopiranje sadržaja međuspremnika `uartBuffer` u novi znakovni niz koji je potrebno deklarirati (`char poruka[100];`). U ovom

pristupu se udvostručuje zauzeće podatkovne memorije u odnosu na prethodna dva primjera jer se tekstualna poruka nalazi zapisana u dva znakovna niza. Unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` funkcijom `uartCopyBuffer(poruka);` se prosljeđuje adresa znakovnog niza `poruka` u koji se kopira sadržaj međuspremnika `uartBuffer`. U konačnici, znakovni niz `poruka` se ispisuje na LCD displej.

Programski kod 11.8: Dohvaćanje primljene tekstualne poruke UART komunikacijom - 3. način

```
char poruka[100];
if (uartIsMessageRecived()) {
    uartCopyBuffer(poruka);
    // prikaz primljene poruke
    lcdprintf("%s", poruka);
}
```

U ovoj vježbi postavit ćemo brzinu prijenosa podataka na 19200 b/s. U programski kod 11.3 u funkciju `init()` upišite poziv funkcije `uartInit(19200)`. S obzirom da ćemo primati poruke pomoću prekidne rutine, u programski kod 11.3 uključite zaglavlje `interrupt.h` naredbom `#include "Interrupt/interrupt.h"`, a u funkciju `init()` upišite poziv funkcije `interruptEnable();` kojom se globalno omogućuju prekidi. Prekide koje izaziva primljen znak na pin RXD omogućen je kroz poziv funkcije `uartInit(19200)`.

U beskonačnu `while` petlju kopirajte programski kod 11.6. Dodatno, unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` napišite poziv funkcije `uartprintf("%s\r", uartBuffer);` nakon ispisa poruke na LCD displej. Ovom funkcijom u terminal *Tera Term* vraćamo primljenu poruku.

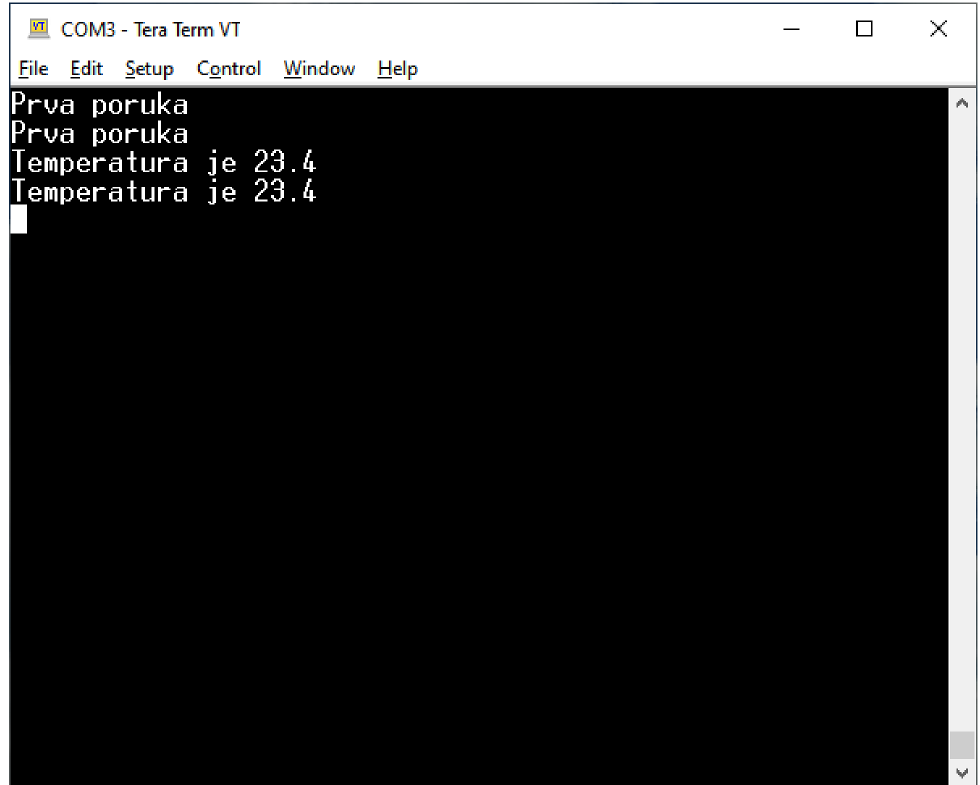
Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba112.cpp` treba biti ista kao programski kod 11.9.

Programski kod 11.9: Primanje poruka proizvoljnih poruka UART komunikacijom te ispis poruke na LCD displej i slanje poruke putem UART komunikacije

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    uartInit(19200); // inicijalizacija UART komunikacije
    interruptEnable(); // globalno omogućeni prekidi
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
            uartprintf("%s\r", uartBuffer);
        }
    }
}
```

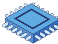
Prevedite datoteku `vjezba112.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7.

Podesite brzinu prijenosa podataka na 19200 b/s. U terminal sada upišite poruku `Prva poruka` i pritisnite tipku *Enter*. Nakon toga, u terminal upišite poruku `Temperatura je 23.4` i pritisnite tipku *Enter*. Ako ste slijedili navedene korake, u terminalu će se ispisivati poruke prikazane na slici 11.9. Pokušajte slati proizvoljne poruke. Iste će se ispisivati na LCD displej (prvih 16 znakova).



Slika 11.9: Ispis proizvoljnih poruka UART komunikacijom - `vjezba112.cpp`

Zatvorite datoteku `vjezba112.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 11.3

Napišite program kojim ćete mijenjati stanja crvene, žute i zelene LED diode koje su spojene na mikroupravljač ATmega328P pomoću kodiranih tekstualnih poruka koje se primaju putem UART komunikacije. Poruke se šalju putem terminala *Tera Term*, a kraj poruke se definira tipkom *Enter*. Poruke se kodiraju s tri znaka: `port`, `pin`, `stanje`. Značenje pojedinog znaka jest:

- `port` - prvi znak koji određuje port mikroupravljača ATmega328P na kojem će se mijenjati stanje (`port = 'B', 'C', 'D'`),
- `pin` - drugi znak koji određuje poziciju pina na portu mikroupravljača ATmega328P na kojem će se mijenjati stanje (`pin = '0', '1', '2', '3','4', '5', '6', '7'`),
- `stanje` - treći znak koji određuje stanje pina koje može biti nisko (0) i visoko (1) (`stanje = '0', '1' `),

Primljenu poruku ispišite na LCD displeju. Na primjer, ako je primljena poruka `"B11"`, crvenu LED diodu treba uključiti, a ako je primljena poruka `"B10"`, crvenu LED diodu treba isključiti. Brzinu prijenosa podataka postavite na 38400 b/s. Znakovni nizovi koje pomoću

mikroupravljača primamo i šaljemo UART komunikacijom zaključani su *Carriage Return* znakom (`'\r'`). Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, žuta LED dioda spojena je na digitalni pin PB2, a zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba113.cpp`. Omogućite prevođenje datoteke `vjezba113.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba113.cpp` prikazan je programskim kodom 11.10.

Programski kod 11.10: Početni sadržaj datoteke `vjezba113.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
}
int main(void) {

    init(); // inicijalizacija mikroupravljača
    char port, pin, stanje;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
        }
    }
}
```

U ovoj vježbi cilj nam je pomoću primljenih poruka UART komunikacijom napraviti akcije poput promjena stanja LED diode. Pošto u programskom kodu 11.10 nisu provedene inicijalizacije ni konfiguracije mikroupravljača ATmega328P, potrebno ih je provesti. U programski kod 11.10 u funkciju `init()` upišite sljedeće naredbe:

- `uartInit(38400);` - UART sklopovlje inicijalizirano je sa svim navedenim fiksnim parametrima (8-bitni podaci, 1 stop bit, ...) i brzinom prijenosa podataka iznosa 38400 b/s,
- `interruptEnable();` - globalno omogućenje prekida,
- `pinMode(B1, OUTPUT);` - pin PB1 na koji je spojena crvena LED dioda konfiguriran kao izlaz,
- `pinMode(B2, OUTPUT);` - pin PB2 na koji je spojena žuta LED dioda konfiguriran kao izlaz,
- `pinMode(B3, OUTPUT);` - pin PB3 na koji je spojena zelena LED dioda konfiguriran kao izlaz.

Kako je navedeno, poruke se kodiraju s tri znaka: `port`, `pin`, `stanje`. Deklarirajte ove tri varijable tako da ispod poziva `init()` funkcije u programskom kodu 11.10 napišete `char port, pin, stanje;`. Tri znaka čine poruku koja omogućuje odabir porta i pina na kojem

treba promijeniti stanje. Promjena stanja crvene, žute i zelene LED diode kodira se sljedećim porukama:

- `"B11"` - uključiti crvenu diodu na pinu PB1,
- `"B10"` - isključiti crvenu diodu na pinu PB1,
- `"B21"` - uključiti žutu diodu na pinu PB2,
- `"B20"` - isključiti žutu diodu na pinu PB2,
- `"B31"` - uključiti zelenu diodu na pinu PB3,
- `"B30"` - isključiti zelenu diodu na pinu PB3.

Ove poruke primaju se u međuspremniku `uartBuffer`. Za primjer primljene poruke `"B31"` vrijedi:

- `uartBuffer[0] = 'B'` - prvi znak u međuspremniku je na poziciji indeksa 0,
- `uartBuffer[1] = '3'` - drugi znak u međuspremniku je na poziciji indeksa 1,
- `uartBuffer[2] = '1'` - treći znak u međuspremniku je na poziciji indeksa 2.

Da bismo dekodirali primljenu poruku, moramo dohvatiti primljenu poruku međuspremnika `uartBuffer` i pojedine znakove poruke spremiti u varijable `port`, `pin` i `stanje`. Unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` upišite sljedeće naredbe za pridruživanje:

- `port = uartBuffer[0];` - prvi znak u međuspremniku `uartBuffer` sprema se u varijablu `port`,
- `pin = uartBuffer[1];` - drugi znak u međuspremniku `uartBuffer` sprema se u varijablu `pin`,
- `stanje = uartBuffer[2];` - treći znak u međuspremniku `uartBuffer` sprema se u varijablu `stanje`.

Ako ciljano želimo mijenjati stanje digitalnih pinova na kojima su spojene crvena, žuta i zelena LED dioda, tada je potrebno provjeriti da li je prvi znak u međuspremniku `uartBuffer` jednak znaku `'B'`. U nastavku programskog koda 11.10 dodajte naredbu grananja `if (port == 'B'){ }`. Ako je poruka započela sa znakom `'B'`, promjena stanja biti će na portu B. Drugim znakom u međuspremniku `uartBuffer` definirana je pozicija pina na kojoj se radi promjena stanja. S obzirom da je potrebno provjeriti da li je drugi znak jednak znakovima `'1'`, `'2'` ili `'3'`, koristi ćemo `switch case` uvjetovano grananje. Automat stanja za promjenu stanja LED dioda izveden je pomoću `switch case` uvjetovanog grananja prikazan je programskim kodom 11.11. Ovo `switch case` uvjetovano grananje upišite unutar naredbe grananja `if (port == 'B'){ }`.

Programski kod 11.11: Automat stanja za promjenu stanja LED dioda

```
switch(pin) {
    case '1':
    if (stanje == '1') digitalWrite(B1, 1);
    if (stanje == '0') digitalWrite(B1, 0);
    break;
    case '2':
    if (stanje == '1') digitalWrite(B2, 1);
```

```
    if (stanje == '0') digitalWrite(B2, 0);
    break;
    case '3':
    if (stanje == '1') digitalWrite(B3, 1);
    if (stanje == '0') digitalWrite(B3, 0);
    break;
    default:
    break;
}
```

Uvjetovano grananje `switch case` na temelju varijable `pin`, određuje da li će uključiti ili isključiti crvenu, žutu ili zelenu LED diodu. Na primjer, ako je varijabla `pin` jednaka `'1'` (`case '1':`), tada se uključuje ili isključuje crvena LED dioda spojena na pin PB1. Treći znak pohranjen u varijablu `stanje` određuje buduće stanje LED diode. Na primjeru crvene LED diode vrijedi:

- ako je varijabla `stanje` jednaka `'1'` (`if (stanje == '1')`), tada uključi crvenu LED diodu (`digitalWrite(B1, 1);`),
- ako je varijabla `stanje` jednaka `'0'` (`if (stanje == '0')`), tada isključi crvenu LED diodu (`digitalWrite(B1, 0);`).

Isti obrazac se može primijeniti na ostale LED diode. Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba113.cpp` treba biti ista kao programski kod 11.12.

Programski kod 11.12: Promjena stanja LED dioda pomoću kodiranih poruka koje se šalju UART komunikacijom

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    uartInit(38400); // inicijalizacija UART komunikacije
    interruptEnable(); // globalno omogućeni prekidi
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    char port, pin, stanje;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
            port = uartBuffer[0];
            pin = uartBuffer[1];
            stanje = uartBuffer[2];
            // provjera prvog znaka
            if (port == 'B') {
                // automat stanja za promjenu stanja dioda
                switch(pin) {
                    case '1':
                    if (stanje == '1') digitalWrite(B1, 1);
                    if (stanje == '0') digitalWrite(B1, 0);
                    break;
```

```
                    case '2':
                    if (stanje == '1') digitalWrite(B2, 1);
                    if (stanje == '0') digitalWrite(B2, 0);
                    break;
                    case '3':
                    if (stanje == '1') digitalWrite(B3, 1);
                    if (stanje == '0') digitalWrite(B3, 0);
                    break;
                    default:
                    break;
                }
            }
        }
    }
}
```

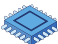
Prevedite datoteku `vjezba113.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7. Podesite brzinu prijenosa podataka na 38400 b/s.

U terminal sada upišite sljedeće poruke:

- `B11` i pritisnite tipku *Enter*,
- `B21` i pritisnite tipku *Enter*,
- `B20` i pritisnite tipku *Enter*,
- `B11` i pritisnite tipku *Enter*,
- `Ovo je poruka` i pritisnite tipku *Enter*,
- `B41` i pritisnite tipku *Enter*,
- `B31` i pritisnite tipku *Enter*,
- ...

Sve poslane poruke bit će ispisane na LCD displeju. Ova vježba može se minimalnim izmjenama primijeniti za potrebe promjene stanja bilo kojeg pina mikroupravljača ATmega328P. Pokušajte razmisliti što bi bilo sve potrebno napraviti kada bi za varijablu `stanje` definirali dodatnu vrijednost `'2'` koja bi mijenjala trenutno stanje pina (`digitalToggle()`).

Zatvorite datoteku `vjezba113.cpp` i onemogućite prevođenje ove datoteke.

 **Vježba 11.4**

Napišite program kojim ćete mijenjati intenzitet crvene i žute LED diode koje su spojene na mikroupravljač ATmega328P pomoću kodiranih tekstualnih poruka koje se primaju putem UART komunikacije. Poruke se šalju putem terminala *Tera Term*, a kraj poruke se definira tipkom *Enter*. Iz terminala *Tera Term* se šalju dvije vrste poruka. Prva vrsta poruka omogućuje i onemogućuje generiranje PWM signala na kanalima A i B sklopa *Timer/Counter1* na sljedeći način:

- poruka `"B10"` onemogućuje generiranje PWM signala na kanalu A sklopa *Timer/Counter1*
- poruka `"B11"` omogućuje generiranje PWM signala na kanalu A sklopa *Timer/Counter1*
- poruka `"B20"` onemogućuje generiranje PWM signala na kanalu B sklopa *Timer/Counter1*
- poruka `"B21"` omogućuje generiranje PWM signala na kanalu B sklopa *Timer/Counter1*

Druga vrsta poruka određuje faktor vođenja za crvenu i žutu LED diodu. Oblik poruke jest `"Pxxx;yyy"` gdje:

- znak `'P'` određuje da se poruka odnosi na PWM signal (intezitet LED dioda),
- niz znakova `"xxx"` određuje cjelobrojni faktor vođenja u rasponu 0 do 100 za crvenu LED diodu, odnosno A kanal (OC1A),
- niz znakova `"yyy"` određuje cjelobrojni faktor vođenja u rasponu 0 do 100 za žutu LED diodu, odnosno B kanal (OC1B).

Primljenu poruku ispišite na LCD displeju. Nakon svake ispravno primljene i obrađene poruke, UART komunikacijom pošaljite status kanala A i B (omogućen/onemogućen, faktor vođenja). Brzinu prijenosa podataka postavite na 19200 b/s. Znakovni nizovi koje pomoću mikroupravljača primamo i šaljemo UART komunikacijom zaključani su *Carriage Return* znakom (`'\r'`). Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*. Prema shemi na slici 4.1, crvena LED dioda spojena je na digitalni pin PB1, a žuta LED dioda spojena je na digitalni pin PB2 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba114.cpp`. Omogućite prevođenje datoteke `vjezba114.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba114.cpp` prikazan je programskim kodom 11.13.

Programski kod 11.13: Početni sadržaj datoteke `vjezba114.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    uartInit(19200); // inicijalizacija UART komunikacije
    interruptEnable(); // globalno omoguceni prekidi
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (zuta LED dioda)
```

```
    // postavljanje Phase Correct PWM nacina rada za timer1 - 8 bit
    timer1PhaseCorrectPWM8bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    int dutyCrvena = 0;
    int dutyZuta = 0;
    bool DUpdate = false;
    char pin, stanje;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
        }
    }
}
```

U ovoj vježbi cilj nam je pomoću primljenih poruka UART komunikacijom mijenjati intenzitet crvene i žute LED diode. Za potrebe ove vježbe, u programskom kodu 11.13 provedene su sve inicijalizacije u funkciji `init()`:

- inicijaliziran je LCD displej,
- UART sklopovlje inicijalizirano je sa svim navedenim fiksnim parametrima (8-bitni podaci, 1 stop bit, ...) i brzinom prijenosa podataka iznosa 19200 b/s,
- globalno je omogućen prekid,
- pin PB1 (crvena LED dioda) i pin PB2 (žuta LED dioda) konfigurirani su kao izlazni pinovi,
- sklop *Timer/Counter1* konfiguriran je za *Phase Correct* način rada s rezolucijom od 8 bitova.

Omogućenje generiranja PWM signala na kanalima A i B nismo proveli u inicijalizacijskoj funkciji. Ovo ćemo provesti na temelju poruka koje ćemo primati UART komunikacijom. U glavnoj funkciji `main()` deklarirane su i inicijalizirane sljedeće varijable:

- `dutyCrvena` - cjelobrojna varijabla koja predstavlja faktor vođenja za kanal A (OC1A), odnosno za crvenu LED diodu,
- `dutyZuta` - cjelobrojna varijabla koja predstavlja faktor vođenja za kanal B (OC1B), odnosno za žutu LED diodu,
- `DUpdate` - *Boolean* varijabla koja omogućuje promjenu faktora vođenja na kanalu A i B uslijed poruke koja je stigla putem UART komunikacije,
- `pin` - znakovna varijabla kojom se definira pin na portu B na kojem će se omogućiti ili onemogućiti generiranje PWM signala (u našem slučaju kanal A ili B),
- `stanje` - znakovna varijabla kojom se definira omogućenje (`'1'`) ili onemogućenje (`'0'`) PWM signala.

Prva vrsta poruka omogućuje i onemogućuje generiranje PWM signala na kanalima A i B

sklopa *Timer/Counter1*. Prvi znak u ovoj poruci jest znak `'B'`. Dakle, u međuspremniku `uartBuffer` potrebno je provjeriti da li je prvi znak jednak znaku `'B'`. Automat stanja za omogućenje generiranja PWM signala na kanalu A i B prikazan je programskim kodom 11.14. Ovaj programski kod (automat stanja) potrebno je upisati u programski kod 11.13 unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` (nakon naredbe za ispis sadržaja međuspremnika `uartBuffer` na LCD displej). Uvjetovano grananje `switch case` nalazi se unutar grananja `if (uartBuffer[0] == 'B'){ }` kojim se provjerava prvi znak u međuspremniku `uartBuffer`. Ako je prvi znak jednak znaku `'B'`, tada se provjeravaju druga dva znaka koji se spremaju u varijable `pin` i `stanje` na sljedeći način:

- `pin = uartBuffer[1];` - drugi znak u međuspremniku `uartBuffer` koji određuje kanal (`'1'` - kanal A (PB1), `'2'` - kanal B (PB2)),

- `stanje = uartBuffer[2];` - treći znak u međuspremniku `uartBuffer` koji određuje stanje kanala A ili B (`'1'` - omogući generiranje PWM signala, `'0'` - onemogući generiranje PWM signala).

Ovisno o pinu i stanju, ispisuje se statusna poruka i omogućuje se/onemogućuje se generiranje PWM signala na pojedinom kanalu. Na primjer, ako je varijabla `pin == '1'` i ako je varijabla `stanje == '1'`, tada se omogućuje neinvertirajući PWM signal na kanalu A i ispisuje se poruka putem UART komunikacije `OC1A omogucen (crvena LED dioda)!`.

Programski kod 11.14: Automat stanja za omogućenje generiranja PWM signala na kanalu A i B

```
if (uartBuffer[0] == 'B') {
    pin = uartBuffer[1];
    stanje = uartBuffer[2];
    // automat stanja za enable i disable PWM kanala
    switch(pin) {
        case '1':
        if (stanje == '1') {
            timer1OC1AEnableNonInvertedPWM();
            uartprintf("OC1A omogucen (crvena LED dioda)!\r");
        }
        if (stanje == '0') {
            timer1OC1ADisable();
            uartprintf("OC1A onemogucen (crvena LED dioda)!\r");
        }
        break;
        case '2':
        if (stanje == '1') {
            timer1OC1BEnableNonInvertedPWM();
            uartprintf("OC1B omogucen (zuta LED dioda)!\r");
        }
        if (stanje == '0') {
            timer1OC1BDisable();
            uartprintf("OC1B onemogucen (zuta LED dioda)!\r");
        }
        break;
        default:
        break;
    }
}
```

Druga vrsta poruka određuje faktor vođenja za crvenu i žutu LED diodu. Oblik poruke jest `"Pxxx;yyy"`. S obzirom da poruka ima zadanu strukturu, moguće ju je razdvojiti na četiri ključna elementa: znak `'P'`, prvi cijeli broj, znak `';'` i drugi cijeli broj. Razdvajanje strukturiranih poruka jednostavno je korištenjem funkcije `sscanf()`. Funkcija `sscanf()` istovjetna je funkciji

`scanf()`. Jedina razlika je što funkcija `sscanf()` kao prvi argument prima tekstualnu poruku koju treba razdvojiti na poznate elemente, a funkcija `scanf()` tekstualnu poruku prima sa standardnog ulaza.

Određivanje faktora vođenja za kanal A i B prikazano je programskim kodom 11.15. Ovaj programski kod potrebno je upisati u programski kod 11.13 unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` (nakon prethodno napisanog programskog koda 11.14). Poziv funkcije `sscanf()` nalazi se unutar uvjetovanog grananja `if (uartBuffer[0] == 'P'){ }` kojim se provjerava da li je početak tekstualne poruke povezan s PWM signalom. Poziv funkcije `sscanf(uartBuffer, "P%d;%d", &dutyCrvena, &dutyZuta);` ima sljedeće argumente:

- `uartBuffer` - međuspremnik u koji je stigla poruka UART komunikacijom (npr. `"P20;70"`),
- `P%d;%d` - formatirani string koji prikazuje strukturu poruke (npr. znak `'P'`, 20, znak `';'` i 70),
- `&dutyCrvena` - adresa varijable u koju se sprema prvi cijeli broj (npr. 20) koji predstavlja faktor vođenja za kanal A (crvena LED dioda),
- `&dutyZuta` - adresa varijable u koju se sprema drugi cijeli broj (npr. 70) koji predstavlja faktor vođenja za kanal B (žuta LED dioda).

Nakon poziva funkcije `sscanf()`, varijabla `DUpdate` postavlja se u vrijednost `true` kako bi se omogućilo osvježenje faktora vođenja na kanalima A i B te ispisale statusne poruke o faktorima vođenja na kanalima A i B.

Programski kod 11.15: Određivanje faktora vođenja za kanal A i B

```
if (uartBuffer[0] == 'P') {
    sscanf(uartBuffer, "P%d;%d", &dutyCrvena, &dutyZuta);
    DUpdate = true;
}
```

Postavljanje faktora vođenja za kanal A i B te ispis statusnih poruka o faktorima vođenja prikazano je programskim kodom 11.16. Ako je varijabla `DUpdate` postavljena u vrijednost `true` provode se sljedeće naredbe:

- `timer1OC1ADutyCycle(dutyCrvena);` - osvježava se faktor vođenja na kanalu A (crvena LED dioda),
- `timer1OC1BDutyCycle(dutyZuta);` - osvježava se faktor vođenja na kanalu B (žuta LED dioda),
- `uartprintf("Faktor vodenja za crvenu LED: %d %%\r", dutyCrvena);` - UART komunikacijom ispisuje se faktor vođenja na kanalu A,
- `uartprintf("Faktor vodenja za zutu LED: %d %%\r", dutyZuta);` - UART komunikacijom ispisuje se faktor vođenja na kanalu B,
- `DUpdate = false;` - onemogućenje osvježavanja faktora vođenja na kanalima A i B i ispisa statusnih poruka do nove poruke.

Ovaj programski kod dodajte na kraj beskonačne `while` petlje.

Programski kod 11.16: Postavljanje faktora vođenja za kanal A i B te ispis statusnih poruka o faktorima vođenja

```
if (DUpdate) {
    timer1OC1ADutyCycle(dutyCrvena);
    timer1OC1BDutyCycle(dutyZuta);
    uartprintf("Faktor vodenja za crvenu LED: %d %%\r", dutyCrvena);
    uartprintf("Faktor vodenja za zutu LEDF: %d %%\r", dutyZuta);
    DUpdate = false;
}
```

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba114.cpp` treba biti ista kao programski kod 11.17.

Programski kod 11.17: Promjena intenziteta crvene i žute LED diode pomoću kodiranih poruka koje se šalju UART komunikacijom

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    uartInit(19200); // inicijalizacija UART komunikacij
    interruptEnable(); // globalno omoguceni prekidi
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (zuta LED dioda)
    // postavljanje Phase Correct PWM nacina rada za timer1 - 8 bit
    timer1PhaseCorrectPWM8bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    int dutyCrvena = 0;
    int dutyZuta = 0;
    bool DUpdate = false;
    char pin, stanje;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
            // omogućenje PWM signala
            if (uartBuffer[0] == 'B') {
                pin = uartBuffer[1];
                stanje = uartBuffer[2];
                // automat stanja za enable i disable PWM kanala
                switch(pin) {
                    case '1':
                    if (stanje == '1') {
                        timer1OC1AEnableNonInvertedPWM();
                        uartprintf("OC1A omogucen (crvena LED dioda)!\r");
                    }
                    if (stanje == '0') {
                        timer1OC1ADisable();
                        uartprintf("OC1A onemogucen (crvena LED dioda)!\r");
                    }
                    break;
                    case '2':
```

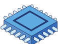
```
                    if (stanje == '1') {
                        timer1OC1BEnableNonInvertedPWM();
                        uartprintf("OC1B omogucen (zuta LED dioda)!\r");
                    }
                    if (stanje == '0') {
                        timer1OC1BDisable();
                        uartprintf("OC1B onemogucen (zuta LED dioda)!\r");
                    }
                    break;
                    default:
                    break;
                }
            }
            // odabir faktora vođenja
            if (uartBuffer[0] == 'P') {
                sscanf(uartBuffer, "P%d;%d", &dutyCrvena, &dutyZuta);
                DUpdate = true;
            }
        }
        // osvjezavanje faktora vođenja kroz registar OCR1A i OCR1B
        if (DUpdate) {
            timer1OC1ADutyCycle(dutyCrvena);
            timer1OC1BDutyCycle(dutyZuta);
            uartprintf("Faktor vodenja za crvenu LED: %d %%\r", dutyCrvena);
            uartprintf("Faktor vodenja za zutu LEDF: %d %%\r", dutyZuta);
            DUpdate = false;
        }
    }
}
```

Prevedite datoteku `vjezba114.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7. Podesite brzinu prijenosa podataka na 19200 b/s.

U terminal sada upišite sljedeće poruke:

- `B11` i pritisnite tipku *Enter*,
- `B21` i pritisnite tipku *Enter*,
- `P10;90` i pritisnite tipku *Enter*,
- `P80;1` i pritisnite tipku *Enter*,
- `P100;100` i pritisnite tipku *Enter*,
- `B20` i pritisnite tipku *Enter*,
- ...

Sve poslane poruke bit će ispisane na LCD displeju, a intenzitet crvene i žute LED diode će se mijenjati sukladno faktoru vođenja. Zatvorite datoteku `vjezba114.cpp` i onemogućite prevođenje ove datoteke.

 Vježba 11.5

Napišite program kojim ćete pomoću UART komunikacije i mikroupravljača ATmega328P osigurati sljedeće funkcionalnosti:

- kada u terminalu *Tera Term* unesemo znak `'I'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje osnovne informacije o mikroupravljaču ATmega328P (naziv, količina programske i podatkovne memorije, broj tajmera i PWM kanala),

- kada u terminalu *Tera Term* unesemo znak `'Z'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje informaciju o stanju zelene LED diode,

- kada u terminalu *Tera Term* unesemo znak `'T'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje informaciju o stanju tipkala T1,

- kada u terminalu *Tera Term* unesemo znak `'A'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje rezultat AD pretvorbe na analognom pinu ADC0,

- kada u terminalu *Tera Term* unesemo znak `'U'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje napon na analognom pinu ADC0,

- kada u terminalu *Tera Term* unesemo znak `'P'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje postotak zakreta potenciometra koji je spojen na analogni ulaz ADC0,

- kada u terminalu *Tera Term* unesemo znak `'V'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje vrijeme rada mikroupravljača u sekundama,

- kada u terminalu *Tera Term* unesemo znak `'?'` i pritisnemo tipku *Enter*, mikroupravljač ATmega328P UART komunikacijom šalje opis svakog gore navedenog znaka (*Help* za rad s programom),

- za sve druge znakove i/ili tekstualne poruke poslane kroz terminal *Tera Term*, mikroupravljač mora poslati poruku `"Neispravan unos!  Unesite ?  za pomoc"`.

Primljenu poruku ispišite na LCD displeju. Znakovni nizovi koje pomoću mikroupravljača primamo i šaljemo UART komunikacijom zaključani su *Carriage Return* znakom (`'\r'`). Stanje zelene LED diode mijenjate jednom u sekundi, a vrijeme mjerite pomoću sklopa *Timer/Counter1*. *Help* za rad s programom pošaljite UART komunikacijom na terminal *Tera Term* prilikom uključenja mikroupravljača. Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*. Prema shemi na slici 4.1, zelena LED dioda spojena je na digitalni pin PB3 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3.

U projektnom stablu otvorite datoteku `vjezba115.cpp`. Omogućite prevođenje datoteke `vjezba115.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba115.cpp` prikazan je programskim kodom 11.18.

Programski kod 11.18: Početni sadržaj datoteke `vjezba115.cpp`

```
#include "AVR/avrlib.h"
```

```
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

volatile uint32_t vrijemeRada = 0;

ISR(TIMER1_OVF_vect) {
    timer1SetValue(3036); // početna vrijednost TCNT1 za 1 s

}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    uartInit(19200); // inicijalizacija UART komunikacije
    interruptEnable(); // globalno omogućeni prekidi
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulazni pin (tipkalo T1)
    pullUpOn(D4); // uključen pull up na pinu PD4
    // postavljanje normalnog nacina rada
    timer1NormalMode();
    // djelitelj frekvencije F_CPU / 64
    timer1SetPrescaler(TIMER1_PRESCALER_256);
    timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
    timer1SetValue(3036); // početna vrijednost TCNT1 za 250 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    float U, P;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
        }
    }
}
```

U ovoj vježbi cilj nam je osigurati velik broj povratnih informacija s mikroupravljača. Informacije se dobivaju putem UART komunikacije na upit. Kako je u zadatku propisano, kada mikroupravljač primi neki od definiranih znakova, prvo mora provesti neku od akcija (provjeriti stanje pina, napraviti AD pretvorbu, ...), a zatim informaciju o provedenoj akciji poslati UART komunikacijom na terminal *Tera Term*.

Za potrebe ove vježbe, u programskom kodu 11.18 provedene su sve inicijalizacije u funkciji `init()`:

- inicijaliziran je LCD displej,
- inicijalizirana je AD pretvorba,
- UART sklopovlje inicijalizirano je sa svim navedenim fiksnim parametrima (8-bitni podaci, 1 stop bit, ...) i brzinom prijenosa podataka iznosa 19200 b/s,
- globalno je omogućen prekid,
- pin PB3 (zelena LED dioda) konfiguriran je kao izlaz,

- pin PD4 (tipkalo T1) konfiguriran je kao ulaz i uključen je pritezni otpornik,
- sklop *Timer/Counter1* konfiguriran je u normalnom načinu rada, a vrijeme između pozivanja prekidne rutine podešeno je na jednu sekundu (prema relaciji 9.22).

U programskom kodu 11.18 nalazi se prekidna rutina za sklop *Timer/Counter1*. U ovoj prekidnoj rutini pomoću varijable `vrijemeRada` potrebno je mjeriti vrijeme u sekundama te je potrebno mijenjati stanje zelene LED diode za potrebe očitanja stanja pina PB3. U programskom kodu 11.18 u prekidnu rutinu sklopa *Timer/Counter1* upišite sljedeće naredbe:

- `vrijemeRada++;` - kod svakog poziva prekidne rutine sklopa *Timer/Counter1* ova se varijabla uveća za 1. Vrijeme između dva poziva prekidne rutine sklopa *Timer/Counter1* iznosi jednu sekundu pa ova varijabla mjeri vrijeme rada mikroupravljača u sekundama.
- `digitalToggle(B3);` - funkcija kojom se svake sekunde promijeni stanje zelene LED diode koja je spojena na pin PB3.

Kada mikroupravljač primi znak `'?'`, UART komunikacijom mora poslati pomoć za rad s programom (popis znakova koji se mikroupravljaču šalju kao upit i njihovo značenje). S obzirom da je ovu pomoć potrebno ispisati i prilikom uključenja mikroupravljača napravit ćemo funkciju `ispisiHelp()`. Definicija funkcije `ispisiHelp()` prikazana je programskim kodom 11.19. Ovu definiciju funkcije upišite u programski kod 11.18 ispod prekidne rutine za sklop *Timer/Counter1*. Nadalje, pozovite ovu funkciju unutar funkcije `init()` (na kraju funkcije).

Programski kod 11.19: Pomoć u korištenju programa

```
void ispisiHelp() {
    uartprintf("\r");
    uartprintf("*****************************************************\r");
    uartprintf("* Povratne informacije za ispis podataka u terminal *\r");
    uartprintf("Unesite dolje navedeni znak i pritisnete Enter:\r");
    uartprintf(" I - informacije o mikroupravljacu\r");
    uartprintf(" Z - stanje zelene LED diode (PB3)\r");
    uartprintf(" T - stanje tipkala T1 (PD4)\r");
    uartprintf(" A - rezultat AD pretvorbe na pinu ADC0\r");
    uartprintf(" U - napon na pinu ADC0\r");
    uartprintf(" P - pozicija potenciometra u %%\r");
    uartprintf(" V - vrijeme rada mikroupravljaca u [s]\r");
    uartprintf(" ? - pomoc\r");
    uartprintf("*****************************************************\r");
    uartprintf("\r");
}
```

Vježbom je definirano što je potrebno napraviti ako se pojavi neki od definiranih znakova. Sve poruke koje se koriste u svrhu ove vježbe su duljine jednog znaka i svaku poruku koja je duža od jednog znaka ćemo smatrati neispravnim unosom. Jedan od načina kojima možemo provjeriti da li je primljena poruka duljine jednog znaka jest taj da provjerimo da li je drugi element međuspremnika `uartBuffer` jednak *null* znaku (`if (uartBuffer[1] == '\0')`). Ako je ovaj uvjet zadovoljen, tada je primljena poruka kandidat za ispravno uneseni znak. Ako uvjet nije zadovoljen tada se ispisuje poruka `Neispravan unos! Unesite ? za pomoc!`. Automat stanja za obradu pristiglih znakova na mikroupravljač putem UART komunikacije prikazan je programskim kodom 11.20. Ovaj programski kod napišite unutar uvjetovanog grananja `if (uartIsMessageRecived()){ }` (nakon naredbe za ispis sadržaja međuspremnika `uartBuffer` na LCD displej). Uvjetovanje grananje `switch case` prima prvi znak u međuspremniku `uartBuffer`.

Programski kod 11.20: Automat stanja za obradu pristiglih znakova na mikroupravljač putem UART komunikacije

```
if (uartBuffer[1] == '\0') {
    // automat stanja za obradu poruka
    switch(uartBuffer[0]) {
        case 'I':
            //informacije o ATmega328P
            uartprintf("Mikroupravljac: ATmega328P\r");
            uartprintf("Programska memorija: 32 kB\r");
            uartprintf("Podatkovna memorija: 2 kB\r");
            uartprintf("Broj tajmera: 3, Broj PWM kanala: 6\r\r");
        break;
        case 'Z':
            //stanje zelene LED diode
            if (digitalRead(B3)) {
                uartprintf("LED zeleno: ON\r\r");
            } else {
                uartprintf("LED zeleno: OFF\r\r");
            }
        break;
        case 'T':
            //stanje tipkala T1
            if (digitalRead(D4)) {
                uartprintf("Tipkalo T1: nije pritisnuto\r\r");
            } else {
                uartprintf("Tipkalo T1: pritisnuto je\r\r");
            }
        break;
        case 'A':
            //rezultat AD pretvorbe na pinu ADC0
            uartprintf("ADC0 = %d\r\r", adcRead(ADC0));
        break;
        case 'U':
            //napon na pinu ADC0
            U = adcReadVoltage(ADC0);
            uartprintf("Napon na pinu ADC: %.3f V\r\r", U);
        break;
        case 'P':
            //zakret potenciometra u %
            P = adcReadScale0To100(ADC0);
            uartprintf("Zakret potenciometra: %.2f%%\r\r", P);
        break;
        case 'V':
            //vrijeme rada mikroupravljaca
            uartprintf("Vrijeme rada: %d s\r\r", vrijemeRada);
        break;
        case '?':
        // ispisivanje pomoci oko unosa znakova
        ispisiHelp();
        break;
        default:
            uartprintf("Neispravan unos! Unesite ? za pomoc!\r");
        break;
    }
} else {
    uartprintf("Neispravan unos! Unesite ? za pomoc!\r");
}
```

Automat stanja za obradu pristiglih znakova na mikroupravljač putem UART komunikacije prikazan programskim kodom 11.20 sastoji se od sljedećih slučajeva u `switch case` uvjetovanom grananju:

- `case 'I':` - znakom `'I'` zahtijeva se slanje informacija o mikroupravljaču putem UART komunikacije. Unutar slučaja `case 'I':` ispisuju se četiri poruke jedna ispod druge: `Mikroupravljac: ATmega328P`, `Programska memorija: 32 kB`, `Podatkovna memorija: 2 kB` i `Broj tajmera: 3`, `Broj PWM kanala: 6` pomoću funkcije `uartprintf()`.

- `case 'Z':` - znakom `'Z'` zahtijeva se slanje stanja zelene LED diode (pin PB3) putem UART komunikacije. Unutar slučaja `case 'Z':` ispituje se stanje zelene LED diode (pin PB3) pomoću funkcije `digitalRead(B3)`. Ako je stanje visoko ispisuje se poruka `LED zeleno: ON`, a ako je stanje nisko ispisuje se poruka `LED zeleno: OFF` pomoću funkcije `uartprintf()`.

- `case 'T':` - znakom `'T'` zahtijeva se slanje stanja tipkala T1 (pin PD4) putem UART komunikacije. Unutar slučaja `case 'T':` ispituje se stanje tipkala T1 (pin PD4) pomoću funkcije `digitalRead(D4)`. Ako je stanje visoko (tipkalo nije pritisnuto) ispisuje se poruka `Tipkalo T1: nije pritisnuto`, a ako je stanje nisko (tipkalo je pritisnuto) ispisuje se poruka `Tipkalo T1: pritisnuto je` pomoću funkcije `uartprintf()`.

- `case 'A':` - znakom `'A'` zahtijeva se slanje rezultata AD pretvorbe na pinu ADC0 putem UART komunikacije. Unutar slučaja `case 'A':` pomoću funkcije `uartprintf()` šalje se poruka `ADC0 = ###`, gdje je `###` rezultat AD pretvorbe koji se kao drugi argument funkcije `uartprintf()` dobije pozivom funkcije `adcRead(ADC0)`.

- `case 'U':` - znakom `'U'` zahtijeva se slanje napona na pinu ADC0 putem UART komunikacije. Unutar slučaja `case 'U':` pomoću funkcije `uartprintf()` šalje se poruka `Napon na pinu ADC: #.##V`, gdje je `#.##` napon na pinu ADC0 koji se dobije naredbom `U = adcReadVoltage(ADC0);`. Varijabla `U` se ispisuje putem UART komunikacije kroz drugi argument funkcije `uartprintf()`.

- `case 'P':` - znakom `'P'` zahtijeva se slanje pozicije potenciometra u postocima putem UART komunikacije. Unutar slučaja `case 'U':` pomoću funkcije `uartprintf()` šalje se poruka `Zakret potenciometra: #.##%`, gdje je `#.##` zakret potenciometra koji se dobije naredbom `P = adcReadScale0To100(ADC0);`. Varijabla `P` se ispisuje putem UART komunikacije kroz drugi argument funkcije `uartprintf()`.

- `case 'V':` - znakom `'V'` zahtijeva se slanje vremena rada mikroupravljača putem UART komunikacije. Unutar slučaja `case 'V':` pomoću funkcije `uartprintf()` šalje se poruka `Vrijeme rada: ### s`, gdje je `###` vrijeme rada mikroupravljača koje je pohranjeno u varijablu `vrijemeRada`. Ovu varijablu osvježavamo unutar prekidne rutine sklopa *Timer/Counter1*. Varijabla `vrijemeRada` se ispisuje putem UART komunikacije kroz drugi argument funkcije `uartprintf()`.

- `case '?':` - znakom `'?'` zahtijeva se slanje pomoći za rad programa putem UART komunikacije. Unutar slučaja `case '?':` poziva se funkcija `ispisiHelp()`.

- `default:` - ukoliko primljeni znak nije jedan od gore navedenih, tada se ispisuje poruka `Neispravan unos! Unesite ? za pomoc!`.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba115.cpp` treba biti ista kao programski kod 11.21.

Programski kod 11.21: Program za obradu pristiglih znakova na mikroupravljač putem UART komunikacije

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

volatile uint32_t vrijemeRada = 0;

ISR(TIMER1_OVF_vect) {
    timer1SetValue(3036); // pocetna vrijednost TCNT1 za 1 s
    vrijemeRada++;
    digitalToggle(B3); // promjena stanja na zelene LED diode
}

void ispisiHelp() {
    uartprintf("\r");
    uartprintf("*****************************************************\r");
    uartprintf("* Povratne informacije za ispis podataka u terminal *\r");
    uartprintf("Unesite dolje navedeni znak i pritisnete Enter:\r");
    uartprintf(" I - informacije o mikroupravljacu\r");
    uartprintf(" Z - stanje zelene LED diode (PB3)\r");
    uartprintf(" T - stanje tipkala T1 (PD4)\r");
    uartprintf(" A - rezultat AD pretvorbe na pinu ADC0\r");
    uartprintf(" U - napon na pinu ADC0\r");
    uartprintf(" P - pozicija potenciometra u %%\r");
    uartprintf(" V - vrijeme rada mikroupravljaca u [s]\r");
    uartprintf(" ? - pomoc\r");
    uartprintf("*****************************************************\r");
    uartprintf("\r");
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    uartInit(19200); // inicijalizacija UART komunikacij
    interruptEnable(); // globalno omogućeni prekidi
    pinMode(B3, OUTPUT); // PB3 konfiguriran kao izlazni pin (zelena LED dioda)
    pinMode(D4, INPUT); // PD4 konfiguriran kao ulazni pin (tipkalo T1)
    pullUpOn(D4); // uključen pull up na pinu PD4
    // postavljanje normalnog nacina rada
    timer1NormalMode();
    // djelitelj frekvencije F_CPU / 64
    timer1SetPrescaler(TIMER1_PRESCALER_256);
    timer1InterruptOVFEnable(); // omogućenje prekida preljevom za timer1
    timer1SetValue(3036); // početna vrijednost TCNT1 za 250 ms
    ispisiHelp();
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float U, P;
    while (1) {
        if (uartIsMessageRecived()) {
            lcdClrScr();
            // prikaz primljene poruke
            lcdprintf("%s", uartBuffer);
            if (uartBuffer[1] == '\0') {
                // automat stanja za obradu poruka
                switch(uartBuffer[0]) {
```

```
                    case 'I':
                        //informacije o ATmega328P
                        uartprintf("Mikroupravljac: ATmega328P\r");
                        uartprintf("Programska memorija: 32 kB\r");
                        uartprintf("Podatkovna memorija: 2 kB\r");
                        uartprintf("Broj tajmera: 3, Broj PWM kanala: 6\r\r");
                    break;
                    case 'Z':
                        //stanje zelene LED diode
                        if (digitalRead(B3)) {
                            uartprintf("LED zeleno: ON\r\r");
                        } else {
                            uartprintf("LED zeleno: OFF\r\r");
                        }
                    break;
                    case 'T':
                        //stanje tipkala T1
                        if (digitalRead(D4)) {
                            uartprintf("Tipkalo T1: nije pritisnuto\r\r");
                        } else {
                            uartprintf("Tipkalo T1: pritisnuto je\r\r");
                        }
                    break;
                    case 'A':
                        //rezultat AD pretvorbe na pinu ADC0
                        uartprintf("ADC0 = %d\r\r", adcRead(ADC0));
                    break;
                    case 'U':
                        //napon na pinu ADC0
                        U = adcReadVoltage(ADC0);
                        uartprintf("Napon na pinu ADC: %.3f V\r\r", U);
                    break;
                    case 'P':
                        //zakret potenciometra u %
                        P = adcReadScale0To100(ADC0);
                        uartprintf("Zakret potenciometra: %.2f%%\r\r", P);
                    break;
                    case 'V':
                        //vrijeme rada mikroupravljaca
                        uartprintf("Vrijeme rada: %d s\r\r", vrijemeRada);
                    break;
                    case '?':
                    // ispisivanje pomoci oko unosa znakova
                    ispisiHelp();
                    break;
                    default:
                        uartprintf("Neispravan unos! Unesite ? za pomoc!\r");
                    break;
                }
            } else {
                uartprintf("Neispravan unos! Unesite ? za pomoc!\r");
            }
        }
    }
}
```

Prevedite datoteku `vjezba115.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7. Podesite brzinu prijenosa podataka na 19200 b/s.

Sve poslane poruke bit će ispisane na LCD displeju. Ako ste slijedili navedene korake, u terminalu će se ispisivati poruke prikazane na slici 11.10 ukoliko ste kroz terminal poslali znakove

koji su prikazani na istoj slici. Rekonstruirajte UART komunikaciju prikazanu slikom 11.10.

Slika 11.10: Slanje znakova i primanje poruka UART komunikacijom - `vjezba115.cpp`

Zatvorite datoteku `vjezba115.cpp` i onemogućite prevođenje ove datoteke.

Vježba 11.6

Napišite program kojim ćete pomoću UART komunikacije i mikroupravljača ATmega328P u terminal *Tera Term* logirati podatke o naponu na pinu ADC0 jednom u sekundi u strogo zajamčenom vremenu. Pojedini *log* mora imati vremensku oznaku i napon na pinu ADC0. Znakovni niz koji pomoću mikroupravljača šaljemo UART komunikacijom zaključan je *Carriage Return* znakom (`'\r'`). Povežite razvojno okruženje USB kabelom na računalo te konfigurirajte COM port u terminalu *Tera Term*. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3.

U projektnom stablu otvorite datoteku `vjezba116.cpp`. Omogućite prevođenje datoteke `vjezba116.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba116.cpp` prikazan je programskim kodom 11.22.

Programski kod 11.22: Početni sadržaj datoteke `vjezba116.cpp`

```
#include "AVR/avrlib.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

volatile bool sendADC = true;
volatile uint32_t vrijemeRada = 0;

void init(){

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    float U;
    while (1) {

    }
}
```

Cilj ove vježbe je logirati podatke pomoću UART komunikacije. Ovo je čest primjer u praksi kada je potrebno vremenski uzorkovati fizikalne veličine (temperatura, vlaga, tlak, ...) i slati ih na neko korisničko sučelje računala koje s mikroupravljačem komunicira pomoću UART komunikacije. Uzorak mjerenja napona na pinu ADC0 mora biti uzet jednom u sekundi u strogo zajamčenom vremenu, stoga je za realizaciju ovog zadatka potrebno koristiti tajmer. Također, uz svaki uzorak napona potrebno je poslati i vrijeme kada je taj uzorak uzet. Primijetit ćete sličnost potreba ove vježbe s prethodnom kada smo mjerili vrijeme rada mikroupravljača.

U programski kod 11.22 kopirajte prekidnu rutinu sklopa *Timer/Counter1* i inicijalizacijsku funkciju `init()` iz prethodne vježbe (Vježba 11.5). S obzirom da u ovoj vježbi nije potrebno koristiti LCD displej, inicijalizacijsku funkciju `lcdInit()` možete obrisati.

U prekidnoj rutinu sklopa *Timer/Counter1* napravite sljedeće radnje:

- obrišite funkciju `digitalToggle(B3);`,
- dodajte naredbu `sendADC = true;`.

Varijablu `sendADC` koristit ćemo u glavnom dijelu programa kao zastavicu (engl. *flag*) temeljem koje ćemo obaviti AD konverziju i izračun napona na ADC0 pinu te poslati informaciju putem UART komunikacije. Teoretski, AD konverziju mogli smo pokrenuti u prekidnoj rutini sklopa *Timer/Counter1*, no kako smo već napomenuli, u prekidnim rutinama važno je obaviti sve naredbe što prije. Kada se pozove prekidna rutina sklopa *Timer/Counter1*, varijabla `sendADC` se postavlja u vrijednost `true`.

U beskonačnoj `while` petlji otvorite uvjetovano grananje `if (sendADC){ }`. Unutar uvjetovanog grananja upišite sljedeće naredbe:

- `U = adcReadVoltage(ADC0);` - mjerenje napona na pinu ADC0 i spremanje u varijablu `U`,
- `uartprintf("t=%lus U=%.2fV\r", vrijemeRada, U);` - slanje vremena uzorkovanja i uzorka napona putem UART komunikacije. Struktura poruke je `t=2s U=2.63V`. Vrijeme

uzorka i napon odvojeni su razmakom. Primijetite da smo za format ispisa vremena uzorkovanja koristili `"%lu"` jer je varijabla `vrijemeRada` tipa `uint32_t` (vidi tablicu 6.1).

- `sendADC = false;` - onemogućenje AD pretvorbe i ispisa UART komunikacijom do sljedećeg poziva prekidne rutine.

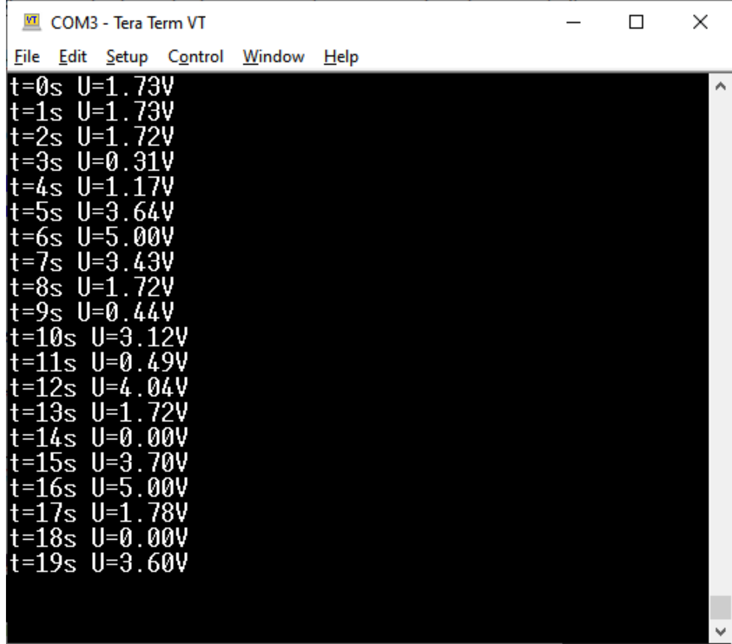
Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba116.cpp` treba biti ista kao programski kod 11.23.

Programski kod 11.23: Program za logiranje vremena i napona na pinu ADC0

```
#include "AVR/avrlib.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

volatile bool sendADC = true;
volatile uint32_t vrijemeRada = 0;

ISR(TIMER1_OVF_vect) {
    timer1SetValue(3036); // pocetna vrijednost TCNT1 za 1 s
    vrijemeRada++;
    sendADC = true;
}


void init() {
    adcInit(); // inicijalizacija AD pretvorbe
    uartInit(19200); // inicijalizacija UART komunikacij
    interruptEnable(); // globalno omoguceni prekidi
    // postavljanje normalnog nacina rada
    timer1NormalMode();
    // djelitelj frekvencije F_CPU / 64
    timer1SetPrescaler(TIMER1_PRESCALER_256);
    timer1InterruptOVFEnable(); // omogucenje prekida preljevom za timer1
    timer1SetValue(3036); // pocetna vrijednost TCNT1 za 250 ms
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    float U;
    while (1) {
        if (sendADC) {
            U = adcReadVoltage(ADC0);
            uartprintf("t=%lus U=%.2fV\r", vrijemeRada, U);
            sendADC = false;
        }
    }
}
```

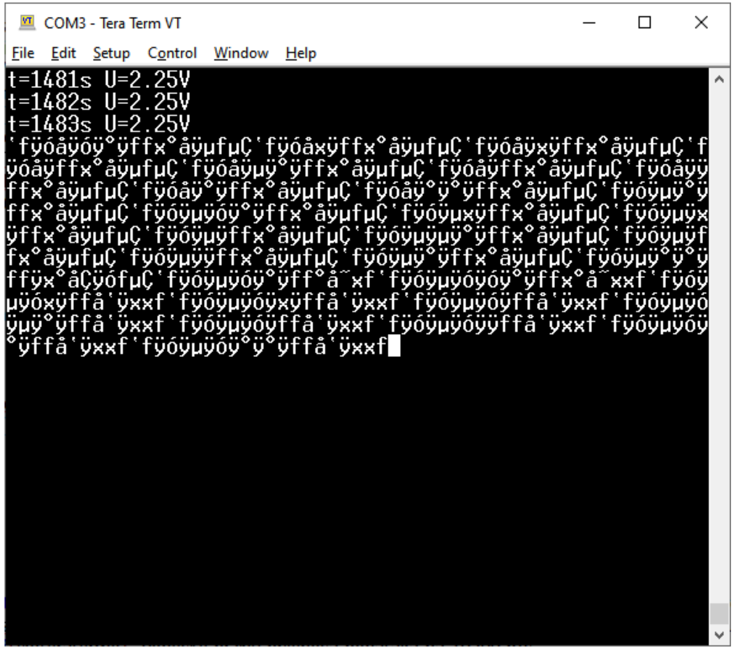
Prevedite datoteku `vjezba116.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P pomoću terminala *Tera Term*. Podesite terminal sukladno uputama na slikama 11.4 - 11.7. Podesite brzinu prijenosa podataka na 19200 b/s.

Ako ste slijedili navedene korake, u terminalu će se ispisivati poruke prikazane na slici 11.11. Vaš zadatak je okretati potenciometar prilikom uzorkovanja kako biste imali različite uzorke napona.



Slika 11.11: Logiranje vremena i napona UART komunikacijom - `vjezba116.cpp`

Pokušajte promijeniti brzinu prijenosa podataka na 38400 b/s samo u terminalu *Tera Term*. Na mikroupravljači neka brzina prijenosa ostane 19200 b/s. Logiranje vremena i napona UART komunikacijom s neispravno podešenom brzinom prijenosa podataka prikazano je na slici 11.12. Možete primijetiti da su poruke nečitljive jer terminal pretpostavlja da je brzina prijenosa dvostruko veća pa će zbog asinkronog prijenosa podataka, prihvaćene poruke biti neispravne.



Slika 11.12: Logiranje vremena i napona UART komunikacijom s neispravno podešenom brzinom prijenosa podataka - `vjezba116.cpp`

Zatvorite datoteku `vjezba116.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio 7*.

Poglavlje 12

Vanjski prekidi

Važan razlog zašto se mikroupravljači koriste u ugradbenim sustavima jest sposobnost mikroupravljača za detekciju niza vanjskih impulsa koji mogu biti niske ili visoke frekvencije. Takav niz impulsa može generirati senzor prisutnosti na proizvodnoj traci koja prati broj kutija ili enkoder kojim se može mjeriti pozicija ili brzina vrtnja elektromotora. Senzori prisutnosti uobičajeno generiraju impulse niskih frekvencija, dok enkoderi mogu generirati i nekoliko tisuća impulsa u jednoj sekundi. Da bismo uspješno prebrojili niz impulsa možemo koristiti brojače, no ako nam je važno prebrojiti bridove signala (rastući i padajući) u tu svrhu možemo koristiti vanjske prekide. Vanjski prekidi (engl. *External Interrupts*) jesu prekidi koji se generiraju zbog promjene stanja vanjskih signala. Promjenu vanjskih signala mogu uzrokovati razni senzori poput tipkala, enkodera, senzora prisutnosti i drugi. Vanjski signali koji su u pravilu nizovi impulsa imaju rastuće i padajuće bridove koje je potrebno prebrojiti. Pomoću mehanizma vanjskog prekida, padajući i rastući bridovi signala mogu izazvati prekid u mikroupravljaču. Kada se dogodi vanjski prekid (padajući i/ili rastući brid signala), poziva se prekidna rutina kojom se obrađuje izazvani prekid. Unutar takve prekidne rutine moguće je imati softverski brojač (varijablu) koju na svaki prekid uvećavamo za 1 kako bismo prebrojili sve impulse.

12.1 Vježbe - vanjski prekidi

Vrste vanjskih prekida koje može generirati mikroupravljač ATmega328P jesu vanjski prekidi (engl. *External Interrupts* - INT) i prekidi izazvani promjenom stanja na pinu (engl. *Pin Change Interrupts* - PCINT). Vanjski prekidi mikroupravljača ATmega328P mogu se generirati na pinovima INT0 (PD2) i INT1 (PD3). Ponovno vidimo da je ovo alternativna primjena digitalnih pinova, stoga ako pinove PD2 i PD3 koristimo u svrhu vanjskih prekida, ne možemo ih koristiti za neku drugu namjenu. Ovi pinovi imaju i jednu iznimku koja odstupa od uobičajenih pravila. Prekidi se na pinovima PD2 i PD3 mogu generirati čak i kada su ovi pinovi konfigurirani kao izlazni pinovi, što nam omogućuje softverski prekid.

Na pinovima INT0 i INT1 prekid se može generirati na padajući i/ili rastući brid signala te niskom razinom signala. Najčešća primjena vanjskih prekida jest za padajuće i/ili rastuće bridove signala. Kod prekida niskom razinom signala, prekidi se uzastopno pozivaju dok god je razina signala niska. Broj prekida u ovom slučaju će biti reda veličine milijun u jednoj sekundi, stoga valja biti oprezan. Način na koji se prekidi izazivaju na pinovima INT0 i INT1 konfigurira se u registru `EICRA` pomoću bitova `ISC00` i `ISC01` za vanjski prekid INT0 te pomoću bitova `ISC10` i `ISC11` za vanjski prekid INT1. Konfiguracija za vanjski prekid na pinu INT0 (PD2) prikazana je tablicom 12.1, dok je konfiguracija za vanjski prekid na pinu INT1 (PD3) prikazana tablicom 12.2.

Vanjski prekidi omogućuju se u registru EIMSK na sljedeći način:

- vanjski prekid na pinu INT0 (PD2) bit će omogućen ako u registar EIMSK na mjesto bita INT0 upišete vrijednost 1,

- vanjski prekid na pinu INT1 (PD3) bit će omogućen ako u registar EIMSK na mjesto bita INT1 upišete vrijednost 1.

Tablica 12.1: Konfiguracija za vanjski prekid na pinu INT0 (PD2)

ISC01	ISC00	Način rada za prekid INT0
0	0	Niska razina signala na pinu INT0 (PD2) generira zahtjev za prekid
0	1	Oba brida signala (padajući i rastući) na pinu INT0 (PD2) generira asinkroni zahtjev za prekid
1	0	Padajući brid signala na pinu INT0 (PD2) generira asinkroni zahtjev za prekid
1	1	Rastući brid signala na pinu INT0 (PD2) generira asinkroni zahtjev za prekid

Tablica 12.2: Konfiguracija za vanjski prekid na pinu INT1 (PD3)

ISC11	ISC10	Način rada za prekid INT1
0	0	Niska razina signala na pinu INT1 (PD3) generira zahtjev za prekid
0	1	Oba brida signala (padajući i rastući) na pinu INT1 (PD3) generira asinkroni zahtjev za prekid
1	0	Padajući brid signala na pinu INT1 (PD3) generira asinkroni zahtjev za prekid
1	1	Rastući brid signala na pinu INT1 (PD3) generira asinkroni zahtjev za prekid

Primjeri omogućenja i konfiguracije vanjskih prekida INT0 i INT1 prikazani su programskim kodom 12.1. Vanjski prekid INT0 konfiguriran je tako da rastući brid signala na pinu INT0 generira prekid. Ova konfiguracija provodi se tako da se u registar EICRA na poziciji bitova ISC00 i ISC01 postave vrijednosti 1 (prema tablici 12.1). Vanjski prekid INT1 konfiguriran je tako da padajući brid signala na pinu INT1 generira prekid. Ova konfiguracija provodi se tako da se u registar EICRA na poziciji bita ISC10 postavi vrijednost 0, a na poziciji bita ISC11 postavi vrijednost 1 (prema tablici 12.2).

Programski kod 12.1: Omogućenje i konfiguracija vanjskih prekida INT0 i INT1

```
// vanjski prekid INT0
EIMSK |= (1 << INT0);  // omogući prekid INT0
// rastući brid generira prekid  na INT0
EICRA |= (1 << ISC01) | (1 << ISC00);
// vanjski prekid INT1
EIMSK |= (1 << INT1);  // omogući prekid INT1
// padajući brid generira prekid  na INT1
EICRA |= (1 << ISC11) | (0 << ISC10);
```

Kada se omoguće vanjski prekidi, tada će oni pozivati sljedeće prekidne rutine:

- ISR(INT0_vect) - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT0 (PD2),

- `ISR(INT1_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT1 (PD3).

Preporuka je da se pinovi za vanjske prekide konfiguriraju kao ulazni pinovi. Kada se za generiranje niza impulsa koriste tipkala ili senzori s otvorenim kolektorom, tada je potrebno uključiti pritezni otpornik na digitalnom ulazu.

Druga vrsta prekida koju podržava mikroupravljač ATmega328P jesu prekidi izazvani promjenom stanja na pinu (PCINT prekidi). Ovi prekidi mogu se generirati na svim digitalnim pinovima mikroupravljača ATmega328P. Kod prekida izazvanih promjenom stanja na pinu, prekidi se mogu generirati čak i kada su pinovi konfigurirani kao izlazni pinovi (kao i kod vanjskih prekida), što nam omogućuje softverski prekid. PCINT prekidi generiraju se kod promjene stanje na pinu (i rastući i padajući brid generira prekid), a dijele se u tri skupine:

- PCINT7..0 - prva skupina obuhvaća alternativne pinove PCINT0 (PB0), PCINT1 (PB1), PCINT2 (PB2), PCINT3 (PB3), PCINT4 (PB4), PCINT5 (PB5), PCINT6 (PB6), PCINT7 (PB7),

- PCINT14..8 - druga skupina obuhvaća alternativne pinove PCINT8 (PC0), PCINT9 (PC1), PCINT10 (PC2), PCINT11 (PC3), PCINT12 (PC4), PCINT13 (PC5), PCINT14 (PC6),

- PCINT23..16 - treća skupina obuhvaća alternativne pinove PCINT16 (PD0), PCINT17 (PD1), PCINT18 (PD2), PCINT19 (PD3), PCINT20 (PD4), PCINT21 (PD5), PCINT22 (PD6), PCINT23 (PD7).

PCINT prekidi omogućuju se u registru `PCICR` tako da na mjesto bita:

- `PCIE0` upišete vrijednost 1 za omogućenje prekida na pinovima iz skupine PCINT7..0,

- `PCIE1` upišete vrijednost 1 za omogućenje prekida na pinovima iz skupine PCINT14..8,

- `PCIE2` upišete vrijednost 1 za omogućenje prekida na pinovima iz skupine PCINT23..16.

PCINT prekid bit će generiran samo ako se omogući za određeni pin iz skupina PCINT7..0, PCINT14..8 i PCINT23..16. Pinovi koji će moći generirati prekid omogućuju se kroz sljedeće registre:

- `PCMSK0` - omogućenje prekida na pinovima iz skupine PCINT7..0,

- `PCMSK1` - omogućenje prekida na pinovima iz skupine PCINT14..8,

- `PCMSK2` - omogućenje prekida na pinovima iz skupine PCINT23..16.

Primjeri omogućenja i konfiguracije PCINT prekida PCINT1, PCINT8, PCINT9 i PCINT18 prikazani su programskim kodom 12.2. Na primjer, ako želimo generirati PCINT prekid pomoću pina PCINT1, tada ćemo u registar `PCMSK0` na mjesto bita `PCINT1` upisati vrijednost 1. S obzirom da je pin PCINT1 iz skupine PCINT7..0, potrebno je omogućiti PCINT prekid pinova iz navedene skupine tako da u registar `PCICR` na mjesto bita `PCIE0` upišemo vrijednost 1. Ako želimo generirati PCINT prekid pomoću pina PCINT18, tada ćemo u registar `PCMSK2` na mjesto bita `PCINT18` upisati vrijednost 1. S obzirom da je pin PCINT18 iz skupine PCINT23..16, potrebno je omogućiti PCINT prekid pinova iz navedene skupine tako da u registar `PCICR` na mjesto bita `PCIE2` upišemo vrijednost 1.

Programski kod 12.2: Omogućenje i konfiguracija PCINT prekida PCINT1, PCINT8, PCINT9, PCINT18

```
// vanjski prekid PCINT1, PCINT8, PCINT9, PCINT18,
// omogući PCINT prekid na pinovima iz skupine PCINT7..0
PCICR |= (1 << PCIE0);
// omogući PCINT prekid na pinovima iz skupine PCINT14..8
PCICR |= (1 << PCIE1);
// omogući PCINT prekid na pinovima iz skupine PCINT23..16
PCICR |= (1 << PCIE2);
// omogućenje pinova koji će generirati prekid
PCMSK0 |= (1 << PCINT1); // omogući PCINT prekid na pinu PCINT1
PCMSK1 |= (1 << PCINT8); // omogući PCINT prekid na pinu PCINT8
PCMSK1 |= (1 << PCINT9); // omogući PCINT prekid na pinu PCINT9
PCMSK2 |= (1 << PCINT18); // omogući PCINT prekid na pinu PCINT18
```

Kada se omoguće PCINT prekidi, tada će oni pozivati sljedeće prekidne rutine:

- `ISR(PCINT0_vect)` - prekidna rutina koja se poziva kada se dogodi promjena stanja na pinovima iz skupine PCINT7..0,
- `ISR(PCINT1_vect)` - prekidna rutina koja se poziva kada se dogodi promjena stanja na pinovima iz skupine PCINT14..8,
- `ISR(PCINT2_vect)` - prekidna rutina koja se poziva kada se dogodi promjena stanja na pinovima iz skupine PCINT23..16.

U prethodnim programskim kodovima prikazana je konfiguracija vanjskih prekida pomoću registara. Kroz vježbe ćemo dodatno prikazati konfiguriranje vanjskih prekida pomoću funkcija čije se definicije nalaze u biblioteci `interrupt.h`. U nastavku ćemo prikazati sve definirane funkcije u zaglavlju `interrupt.h` koje služe za konfiguriranje vanjskih prekida. Autor je pripremio brojne funkcije koje omogućuju jednostavnu konfiguraciju vanjskih i PCINT prekida pomoću zaglavlja `"interrupt.h"` koje se nalazi u mapi `"Interrupt"`.

Konfiguriranje vanjskih prekida INT0 i INT1

U nastavku je popis funkcija koje se koriste za konfiguraciju vanjskih prekida:

- `int0Enable()` - funkcija koja omogućuje vanjski prekid na pinu INT0 (PD2),
- `int1Enable()` - funkcija koja omogućuje vanjski prekid na pinu INT1 (PD3),
- `int0Disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT0 (PD2),
- `int1Disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT1 (PD3),
- `int0LowLevel()` - funkcija kojom se vanjski prekid konfigurira tako da niska razina na pinu INT0 (PD2) generira prekid,
- `int1LowLevel()` - funkcija kojom se vanjski prekid konfigurira tako da niska razina na pinu INT1 (PD3) generira prekid,
- `int0RisingFallingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da oba brida signala na pinu INT0 (PD2) (i rastući i padajući) generiraju prekid,
- `int1RisingFallingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da oba brida signala na pinu INT1 (PD3) (i rastući i padajući) generiraju prekid,

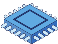
- `int0FallingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da padajući brid signala na pinu INT0 (PD2) generira prekid,
- `int1FallingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da padajući brid signala na pinu INT1 (PD3) generira prekid,
- `int0RisingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da rastući brid signala na pinu INT0 (PD2) generira prekid,
- `int1RisingEdge()` - funkcija kojom se vanjski prekid konfigurira tako da rastući brid signala na pinu INT1 (PD3) generira prekid.

Konfiguriranje PCINT prekida

U nastavku je popis funkcija koje se koriste za konfiguraciju PCINT prekida:

- `pcintEnable7to0()` - funkcija koja omogućuje PCINT prekid na pinovima iz skupine PCINT7..0,
- `pcintEnable14to8()` - funkcija koja omogućuje PCINT prekid na pinovima iz skupine PCINT14..8,
- `pcintEnable23to16()` - funkcija koja omogućuje PCINT prekid na pinovima iz skupine PCINT23..16,
- `pcintDisable()` - funkcija koja onemogućuje PCINT prekid na pinovima iz svih skupina,
- `pcintPinEnable7to0(uint8_t pin)` - funkcija koja omogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT0`, `PCINT1`, `PCINT2`, `PCINT3`, `PCINT4`, `PCINT5`, `PCINT6`, `PCINT7`.
- `pcintPinEnable14to8(uint8_t pin)` - funkcija koja omogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT8`, `PCINT9`, `PCINT10`, `PCINT11`, `PCINT12`, `PCINT13`, `PCINT14`.
- `pcintPinEnable23to16(uint8_t pin)` - funkcija koja omogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT16`, `PCINT17`, `PCINT18`, `PCINT19`, `PCINT20`, `PCINT21`, `PCINT22`, `PCINT23`.
- `pcintPinDisable7to0(uint8_t pin)` - funkcija koja onemogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT0`, `PCINT1`, `PCINT2`, `PCINT3`, `PCINT4`, `PCINT5`, `PCINT6`, `PCINT7`.
- `pcintPinDisable14to8(uint8_t pin)` - funkcija koja onemogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT8`, `PCINT9`, `PCINT10`, `PCINT11`, `PCINT12`, `PCINT13`, `PCINT14`.
- `pcintPinDisable23to16(uint8_t pin)` - funkcija koja onemogućuje PCINT prekid na pinu koji funkciji šaljemo kroz argument `pin`. Argument `pin` poprima konstante `PCINT16`, `PCINT17`, `PCINT18`, `PCINT19`, `PCINT20`, `PCINT21`, `PCINT22`, `PCINT23`.
- `pcintPinDisableAll()` - funkcija koja onemogućuje prekid svih pinova.
- `pcintInit()` - funkcija koja globalnim varijablama `pcintPINBOld`, `pcintPINCOld` i `pcintPINDOld` deklariranim u zaglavlju `interrupt.h` dodjeljuje inicijalne vrijednosti registara `PINB`, `PINC` i `PIND`.

S mrežne stranice https://vub.hr/atmega328p skinite datoteku Vanjski prekidi.zip. Na radnoj površini stvorite praznu datoteku koju ćete nazvati Vaše Ime i Prezime ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se Pero Peric. Datoteku Vanjski prekidi.zip raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite Mikroupravljaci.atsln u datoteci \\Vanjski prekidi\vjezbe. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju Vanjski prekidi. Vježbe ćemo pisati u datoteke s ekstenzijom *.cpp. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.



Vježba 12.1

Napišite program kojim ćete na pinu INT0 (PD2) pomoću tipkala T2 testirati vanjski prekid na način da se prekidi generiraju rastućim bridom, padajućim bridom i s oba brida. Vrijednost brojača bridova signala na pinu INT0 ispišite na LCD displeju. Na svaki padajući brid signala na pinu INT0 (PD2) promijeniti stanje žute LED diode spojene na pin PB2. Prema shemi na slici 5.2, žuta LED dioda spojena je na digitalni izlaz PB2, a tipkalo T2 spojeno je na digitalni ulaz PD2 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku vjezba121.cpp. Omogućite prevođenje datoteke vjezba121.cpp, a prevođenje ostalih datoteka s funkcijom main() onemogućite. Početni sadržaj datoteke vjezba121.cpp prikazan je programskim kodom 12.3.

Programski kod 12.3: Početni sadržaj datoteke vjezba121.cpp

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"

// prekidana rutina za INT0
ISR(INT0_vect) {

}


void init() {
    lcdInit(); // inicijalizacija LCD displeja

}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) {

    }
}
```

U ovoj vježbi testirat ćemo digitalni pin PD2 (INT0) za generiranje vanjskog prekida. Kroz vježbu ćemo provesti konfiguriranje vanjskog prekida INT0 za rastući, za padajući i za oba brida. Prije omogućenja vanjskog prekida INT0, potrebno je pin PD2 konfigurirati kao ulazni pin te globalno omogućiti prekide u mikroupravljaču ATmega328P. U funkciju init() upišite sljedeće naredbe:

- `pinMode(D2, INPUT_PULLUP);` - konfiguriranje digitalnog pina PD2 kao ulazni pin i istovremeno uključenje priteznog otpornika na pinu PD2,

- `interruptEnable();` - globalno omogućenje prekida.

Vanjski prekid INT0 omogućava se u registru `EIMSK` tako da na mjesto bita `INT0` upišemo vrijednost 1. U prvom koraku ćemo konfigurirati vanjski prekid INT0 na način da rastući brid signala na pinu PD2 generira prekid. Ova konfiguracija provodi se tako da se u registar `EICRA` na poziciji bitova `ISC00` i `ISC01` postave vrijednosti 1 (prema tablici 12.1). Za ovu konfiguraciju u funkciju `init()` upišite sljedeće naredbe:

- `EIMSK |= (1 << INT0);` - omogućenje vanjskog prekida INT0,

- `EICRA |= (1 << ISC01)| (1 << ISC00);` - rastući brid generira vanjski prekid na INT0.

Rastuće bridove signala na pinu PD2 potrebno je prebrojiti i ispisati na LCD displej. U programskom kodu 12.3 prikazana je prekidna rutina `ISR(INT0_vect)`. Ova se prekidna rutina poziva, ovisno o konfiguraciji, na svaki brid signala na digitalnom pinu PD2. S obzirom da smo vanjski prekid INT0 konfigurirali tako da rastući brid generira vanjski prekid, prekidna rutina `ISR(INT0_vect)` pozvat će se na svaki rastući brid signala. Na pin PD2 je spojeno tipkalo T2. Kada se pritisne tipkalo, generirat ćemo padajući brid signala (priteže se pin na 0 V), a kada se otpusti tipkalo, generirat ćemo rastući brid signala na digitalnom pinu PD2. Ovu činjenicu treba imati na umu kada se provodi testiranje vježbe.

Broj bridova signala brojit ćemo pomoću softverskog brojača, odnosno pomoću cjelobrojne varijable koju ćemo deklarirati u globalnom prostoru. Dodatno, deklarirat ćemo varijablu koja će nam omogućiti ispis novog stanja brojača bridova na LCD displej. U programski kod 12.3 upišite sljedeće deklaracije u globalnom području varijabli:

- `volatile int brojac = 0;` - deklariran i inicijaliziran brojač (cjelobrojna varijabla) bridova signala na digitalnom pinu PD2,

- `volatile bool lcdUpdate = true;` - deklarirana i inicijalizirana *Boolean* varijabla koja će se postaviti u vrijednost `true` samo kada je potrebno ispisati novu vrijednost brojača na LCD displej (samo po pojavi prekida).

Primijetite da smo ove dvije deklaracije varijabli proglasili `volatile` jer ih mijenja prekidna rutina. U prekidnu rutinu `ISR(INT0_vect)` upišite sljedeće naredbe:

- `brojac++;` - kod svakog prekida (poziva prekidne rutine `ISR(INT0_vect)`) uvećava se varijabla `brojac` za 1,

- `lcdUpdate = true;` - kod svakog prekida (poziva prekidne rutine `ISR(INT0_vect)`) varijabla `lcdUpdate` će se postaviti u vrijednost `true` kako bismo osvježili prikaz na LCD displeju.

Kako smo već spomenuli, novo stanje brojača bridova signala na digitalnom pinu PD2 ispisivat ćemo samo kada se to stanje promjeni, odnosno samo onda kada se pojavio prekid. U beskonačnoj `while` petlji napišite uvjetovano grananje `if (lcdUpdate) { }`, a unutar njega upišite sljedeće naredbe:

- `lcdClrScr();` - obriši sadržaj LCD displeja,

- `lcdprintf("INT = %d", brojac);` - ispiši fiksni dio teksta `INT = ` te stanje varijable `brojac` (broj bridova signala na digitalnom pinu PD2),

- `lcdUpdate = false;` - varijabla `lcdUpdate` se postavlja u vrijednost `false` kako bismo onemogućili ispis na LCD displeju do pojave novog prekida.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba121.cpp` treba biti ista kao programski kod 12.4.

Programski kod 12.4: Program za testiranje vanjskih prekida INT0 pomoću tipkala T2

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"

volatile int brojac = 0;
volatile bool lcdUpdate = true;

// prekidana rutina za INT0
ISR(INT0_vect) {
    brojac++;
    lcdUpdate = true;
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    // pin PD2 konfiguriran kao ulaz + ukljucen pritezni otpornik
    pinMode(D2, INPUT_PULLUP);
    interruptEnable(); // globalno omogu?eni prekidi
    // vanjski prekid INT0
    EIMSK |= (1 << INT0);  // omoguci prekid INT0
    // rastuci brid generira prekid  na INT0
    EICRA |= (1 << ISC01) | (1 << ISC00);

}

int main(void) {

    init(); // inicijalizacija mikroupravljaca

    while (1) {
        if (lcdUpdate) {
            lcdClrScr();
            lcdprintf("INT = %d", brojac);
            lcdUpdate = false;
        }
    }
}
```

Prevedite datoteku `vjezba121.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, svaki puta kada otpustite tipkalo T2 (rastući brid), brojač bridova će se uvećati. Ponekad će to uvećanje biti veće od 1 radi istitravanja tipkala (o ovom problemu smo pričali u poglavlju koje se bavi digitalnim ulazima).

U drugom koraku ćemo konfigurirati vanjski prekid INT0 na način da padajući brid signala na pinu PD2 generira prekid. Ova konfiguracija provodi se tako da se u registar `EICRA` na poziciji bita `ISC00` postavi vrijednost 0, a na poziciji bita `ISC01` postavi vrijednost 1 (prema tablici 12.1). Za ovu konfiguraciju u funkciji `init()` zamijenite naredbu `EICRA |= (1 << ISC01)| (1 << ISC00);` s naredbom `EICRA |= (1 << ISC01)| (0 << ISC00);`.

Prevedite datoteku `vjezba121.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, svaki puta kada pritisnete tipkalo T2 (padajući brid), brojač

bridova će se uvećati.

U trećem koraku ćemo konfigurirati vanjski prekid INT0 na način da padajući i rastući bridovi signala na pinu PD2 generiraju prekid. Ova konfiguracija provodi se tako da se u registar `EICRA` na poziciji bita `ISC00` postavi vrijednost 1, a na poziciji bita `ISC01` postavi vrijednost 0 (prema tablici 12.1). Za ovu konfiguraciju u funkciji `init()` zamijenite naredbu `EICRA |= (1 << ISC01)| (0 << ISC00);` s naredbom `EICRA |= (0 << ISC01)| (1 << ISC00);`.

Prevedite datoteku `vjezba121.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Ako ste slijedili navedene korake, svaki puta kada pritisnete i otpustite tipkalo T2 (padajući i rastući brid), brojač bridova će se uvećati.

U teorijskom dijelu ovog poglavlja prikazali smo funkcije kojima se mogu konfigurirati vanjski prekidi. Inicijalizacije vanjskog prekida INT0 za rastući, za padajući i za oba brida prikazane su programskim kodovima 12.5, 12.6 i 12.7.

Programski kod 12.5: Inicijalizacija vanjskog prekida INT0 za rastući brid - funkcijski pristup

```
// vanjski prekid INT0
int0Enable();  // omoguci prekid INT0
// rastuci brid generira prekid  na INT0
int0RisingEdge();
```

Programski kod 12.6: Inicijalizacija vanjskog prekida INT0 za padajući brid - funkcijski pristup

```
// vanjski prekid INT0
int0Enable();  // omoguci prekid INT0
// rastuci brid generira prekid  na INT0
int0FallingEdge();
```

Programski kod 12.7: Inicijalizacija vanjskog prekida INT0 za rastući i padajući brid - funkcijski pristup

```
// vanjski prekid INT0
int0Enable();  // omoguci prekid INT0
// rastuci brid generira prekid  na INT0
int0RisingFallingEdge();
```

Zamijenite postojeće inicijalizacije vanjskog prekida s inicijalizacijama koje su prikazane programskim kodovima 12.5, 12.6 i 12.7 (svaku pojedinačno) i testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P.

Zatvorite datoteku `vjezba121.cpp` i onemogućite prevođenje ove datoteke.

Vježba 12.2

Napišite program kojim ćete:

- detektirati rastuće i padajuće bridove na pinu PD2 (tipkalo T2) pomoću PCINT prekida. Kada se dogodi rastući brid signala na pinu PD2 na LCD displeju ispišite `PD2 R`, a kada se dogodi padajući brid na LCD displeju ispišite `PD2 F`.
- detektirati rastuće i padajuće bridove na pinu PD4 (tipkalo T1) pomoću PCINT prekida. Kada se dogodi rastući brid signala na pinu PD4 na LCD displeju ispišite `PD4 R`, a kada se dogodi padajući brid na LCD displeju ispišite `PD4 F`.
- detektirati rastuće i padajuće bridove na pinu PD5 (tipkalo T3) pomoću PCINT prekida.

Kada se dogodi rastući brid signala na pinu PD5 na LCD displeju ispišite `PD5 R`, a kada se dogodi padajući brid na LCD displeju ispišite `PD5 F`.

- mijenjati intenzitet crvene LED diode koja je spojena na pin PB1 pomoću sklopa *Timer/Counter1*. Preslikajte promjenu intenziteta crvene LED diode na zelenu LED diodu bez da koristite PWM način rada za zelenu LED diodu koja je spojena na pin PB3. U ovom slučaju koristite PCINT prekid.

Na LCD displeju ostaje ispisana zadnja promjena brida koja se dogodila na pinovima PD2, PD4 ili PD5. Prema shemi na slici 5.2, crvena LED dioda spojena je na digitalni izlaz PB1, zelena LED dioda spojena je na digitalni izlaz PB3, tipkalo T1 spojeno je na digitalni ulaz PD4, tipkalo T2 spojeno je na digitalni ulaz PD2, a tipkalo T3 spojeno je na digitalni ulaz PD5 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3.

U projektnom stablu otvorite datoteku `vjezba122.cpp`. Omogućite prevođenje datoteke `vjezba122.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba122.cpp` prikazan je programskim kodom 12.8.

Programski kod 12.8: Početni sadržaj datoteke `vjezba122.cpp`

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "ADC/adc.h"
#include <string.h>

volatile bool lcdUpdate = true;
char LCDispis[6] = "READY";

// prekidna rutina za PCINT0
ISR(PCINT0_vect) {

}

// prekidna rutina za PCINT2
ISR(PCINT2_vect) {

}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    //konfiguracija izlaza PB1 i PB3
    DDRB |= (1 << PB1) | (1 << PB3);
    //konfiguracija ulaza PD2, PD4 i PD5
    DDRD &= ~((1 << PD2) | (1 << PD4) | (1 << PD5));
    //pull up otpornici ukljuceni na PD2, PD4 i PD5
    PORTD |= (1 << PD2) | (1 << PD4) | (1 << PD5);
    // postavljanje Phase Correct PWM načina rada za timer1 - 10 bit
    timer1PhaseCorrectPWM10bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_64);
    // neinvertirajuci PWM signal na PB1 (OC1A)
    timer1OC1AEnableNonInvertedPWM();
    interruptEnable(); // globalno omogućenje prekida
}
```

```
int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D;
    while (1) {
        // ispis na LCD displej

        D = adcReadScale0To100(ADC0);
        // osvjezavanje faktora vodenja kroz registar OCR1A
        timer1OC1ADutyCycle(D);
    }
}
```

Kako smo objasnili u teorijskom djelu poglavlja, PCINT prekidi generiraju se na promjenu stanja signala (rastući i padajući signal na nekom digitalnom pinu). Ova vrsta prekida korisna je za obradu tipkala te raznih digitalnih senzora. PCINT prekid je na mikroupravljaču ATmega328P omogućen na svim digitalnim pinovima što predstavlja veliku prednost naspram mikroupravljača koji nemaju takve značajke.

U vježbi je potrebno detektirati rastuće i padajuće bridove signala tipkala T1, T2 i T3 koji su spojeni na digitalne pinove PD4, PD2 i PD5. Dodatno, potrebno je preslikati stanje pina PB1 na pin PB3. Na pinu PB1 generiran je PWM signal frekvencije 122,2 Hz pomoću sklopa *Timer/Counter1*.

S obzirom na oblik PWM signala, on mijenja svoje stanje dvaput unutar jednog perioda (na početku perioda pojavljuje se rastući brid signala, a nakon što se dostigne faktor vođenja pojavljuje se padajući brid signala). Oba brida moguće je detektirati PCINT prekidom. S obzirom na prirodu prekida da se trenutno obrađuju kada se dogode, unutar prekidne rutine potrebno je kopirati stanje pina PB1 na pin PB3 kod svake promjene stanja pina PB1. U ovom slučaju pomoću PWM signala na pinu PB1 ćemo generirati softverske prekide. Ovo je vrlo zanimljiva mogućnost jer nam omogućuje da prividno povećamo broj PWM kanala. Općenito, ako je potrebno napraviti kopiranje stanja nekog digitalnog pina, PCINT prekidi su u tu svrhu izvrsno rješenje. Zašto se stanje digitalnog pina ne može kopirati u glavnom programu? Razlog je taj što glavni program uključuje brojna vremenska kašnjenja pa bismo pri preslikavanju stanja s pina na pin imali kašnjenje.

U programskom kodu 12.8 provedene su sljedeće konfiguracije unutar inicijalizacijske funkcije `init()`:

- inicijaliziran je LCD displej,
- inicijalizirana je AD pretvorba,
- pinovi PB1 (crvena LED dioda) i PB3 (zelena LED dioda) konfigurirani su kao izlazni pinovi,
- pinovi PD2 (tipkalo T2), PD4 (tipkalo T1) i PD5 (tipkalo T3) konfigurirani su kao ulazni pinovi te su za ove pinove uključeni pritezni otpornici,
- sklop *Timer/Counter1* konfiguriran je za *Phase Correct* način rada rezolucije 10 bitova: djelitelj frekvencije je 64, a vršna vrijednost do koje tajmer broji jest 1023 (frekvencija 122,2 Hz). Generiranje PWM signala omogućeno je na kanalu A (PB1).
- globalno su omogućeni prekidi.

Preostaje nam u inicijalizacijskoj funkciji `init()` konfigurirati PCINT prekid. Pinovi koji prema zahtjevima vježbe moraju generirati prekide su PB1, PD2, PD4 i PD5. Njihovi

alternativni nazivi su PCINT1 (PB1), PCINT18 (PD2), PCINT20 (PD4) i PCINT21 (PD5).

Pin PCINT1 (PB1) pripada pinovima iz skupine PCINT7..0, dok pinovi PD2, PD4 i PD5 pripadaju pinovima iz skupine PCINT23..16. PCINT prekidi omogućuju se u registru `PCICR` pomoću bitova `PCIE0`, `PCIE1` i `PCIE2`. Da bismo omogućili generiranje prekida pinova iz navedenih skupina, upišite u inicijalizacijsku funkciju `init()` sljedeće naredbe:

- `PCICR |= (1 << PCIE0);` - omogućenje PCINT prekida na pinovima iz skupine PCINT7..0,
- `PCICR |= (1 << PCIE2);` - omogućenje PCINT prekida na pinovima iz skupine PCINT23..16.

S obzirom da ne koristimo niti jedan pin porta C, nije potrebno omogućiti PCINT prekide na pinovima iz skupine PCINT14..8. Dodatno, potrebno je svaki pin pojedinačno konfigurirati tako da može generirati prekid unutar svoje skupine. Prekidi na razini pina omogućuju se u registrima `PCMSK0`, `PCMSK1` i `PCMSK2`. Da bismo omogućili pojedinačno da pinovi PCINT1 (PB1), PCINT18 (PD2), PCINT20 (PD4) i PCINT21 (PD5) mogu generirati prekide unutar svoje skupine, u inicijalizacijsku funkciju `init()` upišite sljedeće naredbe:

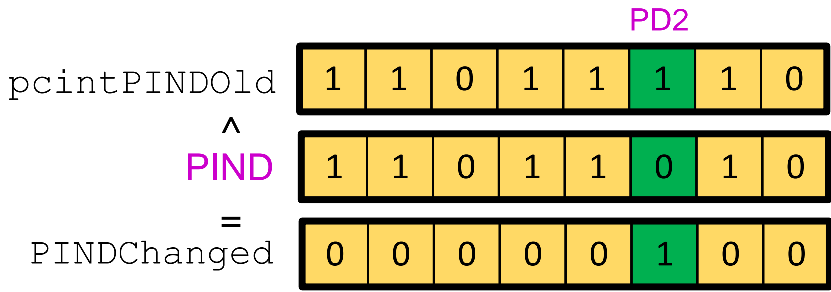
- `PCMSK0 |= (1 << PCINT1);` - omogućenje generiranje prekida za pin PCINT1 (PB1) u skupini PCINT7..0,
- `PCMSK2 |= (1 << PCINT18);` - omogućenje generiranje prekida za pin PCINT18 (PD2) u skupini PCINT23..16,
- `PCMSK2 |= (1 << PCINT20);` - omogućenje generiranje prekida za pin PCINT20 (PD4) u skupini PCINT23..16,
- `PCMSK2 |= (1 << PCINT21);` - omogućenje generiranje prekida za pin PCINT21 (PD5) u skupini PCINT23..16.

Prekid na pinu PB1 pozivat će prekidnu rutinu `ISR(PCINT0_vect)`, dok će prekidi na pinovima PD2, PD4 i PD5 pozivati prekidnu rutinu `ISR(PCINT2_vect)`. Unutar prekidnih rutina potrebno je provesti zadatke za svaki pojedinačni pin. S obzirom da se ista prekidna rutina poziva za sve omogućene pinove unutar skupine, potrebno je detektirati koji je pin izazvao prekid. Da bismo to mogli, potrebno je pohraniti prošlu vrijednost pinova u varijable `pcintPINBOld` i `pcintPINDOld` koje su deklarirane u zaglavlju `interrupt.h`. U inicijalizacijsku funkciju `init()` upišite sljedeće naredbe:

- `pcintPINBOld = PINB;` - spremanje početne vrijednosti registra PINB kako bi se pri sljedećoj promjeni signala na bilo kojem pinu porta B moglo detektirati koji je pin izazvao prekid,
- `pcintPINDOld = PIND;` - spremanje početne vrijednosti registra PIND kako bi se pri sljedećoj promjeni signala na bilo kojem pinu porta D moglo detektirati koji je pin izazvao prekid.

U nastavku ćemo najprije popuniti tijelo prekidne rutine `ISR(PCINT2_vect)`. Kako smo već spomenuli, prvi zadatak prekidne rutine `ISR(PCINT2_vect)` treba biti detekcija pina koji je izazvao prekid. U tu svrhu koristit ćemo bitovni *ex-ili* operator koji ima mogućnost detekcije promjene stanja između dvije varijable. Primjer detekcije promjene stanja pina PD2 koji je izazvao prekid prekidne rutine `ISR(PCINT2_vect)` prikazan je na slici 12.1. Ako staru vrijednost registra `PIND` koja se nalazi u varijabli `pcintPINDOld` podvrgnete bitovnoj *ex-ili* s aktualnom vrijednosti registra `PIND`, tada će rezultat bitovne operacije `PINDChanged` sadržavati vrijednosti

1 samo na onim pozicijama bitova gdje se desila promjena stanja. Na primjer, ako je stara vrijednost na poziciji pina PD2 bila 1, a nova vrijednost je 0, *ex-ili* operacija između 1 i 0 dat će rezultat 1. Za bilo koju drugu situaciju, gdje su stanja i prije i sada jednaka (0 i 0 ili 1 i 1), *ex-ili* operacija će dati rezultat 0 (vidi sliku 12.1).



Slika 12.1: Detekcija pina koji je izazvao PCINT prekid naredbom
`PINDChanged = pcintPINDOld ^ PIND;`

Kod PCINT prekida, prekid ne može generirati više od jednog pina unutar skupine u isto vrijeme. Zbog toga će u varijabli `PINDChanged` vrijednost 1 imat samo jedan bit i to onaj na čijoj se poziciji dogodila promjena stanja u registru `PIND` u odnosu na prethodno stanje registra `PIND`. U prekidnu rutinu `ISR(PCINT2_vect)` unesite naredbu `uint8_t PINDChanged = pcintPINDOld ^ PIND;`.

Preostaje nam provjeriti da li je vrijednost 1 u varijabli `PINDChanged` na poziciji pina PD2, PD4 ili PD5. To ćemo napraviti bitovnom I operacijom pomoću sljedećih uvjeta:

- `if (PINDChanged & (1 << PD2)){ }` - provjera da li je bit na poziciji 2 (PD2) u stanju 1 (provjera da li je pin PD2 izazvao prekid),
- `if (PINDChanged & (1 << PD4)){ }` - provjera da li je bit na poziciji 4 (PD4) u stanju 1 (provjera da li je pin PD4 izazvao prekid),
- `if (PINDChanged & (1 << PD5)){ }` - provjera da li je bit na poziciji 5 (PD5) u stanju 1 (provjera da li je pin PD5 izazvao prekid).

Upišite prethodna uvjetovana grananja u prekidnu rutinu `ISR(PCINT2_vect)`. Ovisno o tome koji pin je izazvao prekid, na LCD displej moramo ispisati poruke:

- `PD2 R` - ako se dogodio rastući brid signala na pinu PD2,
- `PD2 F` - ako se dogodio padajući brid signala na pinu PD2,
- `PD4 R` - ako se dogodio rastući brid signala na pinu PD4
- `PD4 F` - ako se dogodio padajući brid signala na pinu PD4,
- `PD5 R` - ako se dogodio rastući brid signala na pinu PD5,
- `PD5 F` - ako se dogodio padajući brid signala na pinu PD5.

U programskom kodu 12.8 deklarirano je i inicijalizirano polje znakova `char LCDispis[6] = "READY";`. Kroz ovo polje znakova prosljeđivati ćemo navedene tekstove za ispis na LCD displej. U programskom jeziku C za kopiranje znakova u polje znakova koristi se funkcija `char *strcpy(char *dest, const char *src)` koja kopira konstantan niz znakova `src` u polje znakova `dest`. Za korištenje funkcije `strcpy()` potrebno je uključiti zaglavlje

string.h.

Prekidna rutina ISR(PCINT2_vect) za isti se pin poziva i pri rastućem i pri padajućem pinu. Kako odrediti da li je brid signala koji je generirao prekid bio rastući ili padajući? Na primjeru pina PD2 vrijedi:

- ako se dogodio prekid i novo stanje na pinu PD2 je visoko, dogodio se rastući brid,
- ako se dogodio prekid i novo stanje na pinu PD2 je nisko, dogodio se padajući brid.

Isto pravilo vrijedi i za pinove PD4 i PD5. Slijedom svega navedenoga, napravite sljedeće korake:

- unutar uvjetovanog grananja `if (PINDChanged & (1 << PD2)){ }` napišite programski kod 12.9,
- unutar uvjetovanog grananja `if (PINDChanged & (1 << PD4)){ }` napišite programski kod 12.10,
- unutar uvjetovanog grananja `if (PINDChanged & (1 << PD5)){ }` napišite programski kod 12.11.

Programski kod 12.9: Sadržaj uvjetovanog grananja `if (PINDChanged & (1 << PD2)){ }`

```
if (get_pin(PIND, PD2)) {
    strcpy(LCDispis, "PD2 R");
} else {
    strcpy(LCDispis, "PD2 F");
}
```

Programski kod 12.10: Sadržaj uvjetovanog grananja `if (PINDChanged & (1 << PD4)){ }`

```
if (get_pin(PIND, PD4)) {
    strcpy(LCDispis, "PD4 R");
    } else {
    strcpy(LCDispis, "PD4 F");
}
```

Programski kod 12.11: Sadržaj uvjetovanog grananja `if (PINDChanged & (1 << PD5)){ }`

```
if (get_pin(PIND, PD5)) {
    strcpy(LCDispis, "PD5 R");
    } else {
    strcpy(LCDispis, "PD5 F");
}
```

Objasnimo sada programski kod 12.9. Uvjetovanim grananjem `if (get_pin(PIND, PD2)){ }` ispituje se da li je trenutno stanje pina PD2 visoko. Ako je trenutno stanje pina PD2 visoko, dogodio se rastući brid. Pomoću naredbe `strcpy(LCDispis, "PD2 R");` u znakovni niz `LCDispis` kopira se tekst `PD2 R`. Ako je trenutno stanje pina PD2 nisko, dogodio se padajući brid. U alternativnom grananju naredbom `strcpy(LCDispis, "PD2 F");` se u znakovni niz `LCDispis` kopira tekst `PD2 F`.

U nastavku ćemo objasniti programski kod 12.10. Uvjetovanim grananjem `if (get_pin(PIND, PD4)){ }` ispituje se da li je trenutno stanje pina PD4 visoko. Ako je trenutno stanje pina PD4 visoko, dogodio se rastući brid. Pomoću naredbe `strcpy(LCDispis, "PD4 R");` u znakovni niz `LCDispis` kopira se tekst `PD4 R`. Ako je trenutno stanje pina PD4 nisko, dogodio se padajući brid. U alternativnom grananju naredbom `strcpy(LCDispis, "PD4 F");` se u znakovni niz `LCDispis` kopira tekst `PD4 F`.

Naposljetku ćemo objasniti programski kod 12.11. Uvjetovanim grananjem `if (get_pin(PIND, PD5)){ }` ispituje se da li je trenutno stanje pina PD5 visoko. Ako je trenutno stanje pina PD5 visoko, dogodio se rastući brid. Pomoću naredbe `strcpy(LCDispis, "PD5 R");` u znakovni niz `LCDispis` kopira se tekst `PD5 R`. Ako je trenutno stanje pina PD5 nisko, dogodio se padajući brid. U alternativnom grananju naredbom `strcpy(LCDispis, "PD5 F");` se u znakovni niz `LCDispis` kopira tekst `PD5 F`.

Na kraju prekidne rutina `ISR(PCINT2_vect)` upišite sljedeće naredbe:

- `lcdUpdate = true;` - varijabla `lcdUpdate` se postavlja u vrijednost `true` kako bismo omogućili ispis varijable `LCDispis` na LCD displeju.
- `pcintPINDOld = PIND;` - spremanje aktualne vrijednosti registra PIND kako bi se pri sljedećoj promjeni signala na bilo kojem pinu porta D moglo detektirati koji je pin izazvao prekid.

U nastavku ćemo se baviti prekidnom rutinom `ISR(PCINT0_vect)`. U prekidnu rutinu `ISR(PCINT0_vect)` unesite naredbu `uint8_t PINBChanged = pcintPINBOld ^ PINB;` kako bismo detektirali koji je pin na portu B generirao prekid. Detekciju promjene na pinu PB1 ispitat ćemo uvjetovanim grananjem `if (PINBChanged & (1 << PB1)){ }`. Ovo uvjetovano grananje upišite u prekidnu rutinu `ISR(PCINT0_vect)`.

Unutar uvjetovanog grananja `if (PINBChanged & (1 << PB1)){ }` napišite programski kod 12.12.

Programski kod 12.12: Sadržaj uvjetovanog grananja `if (PINBChanged & (1 << PB1)){ }`

```
bool stanje = get_pin(PINB, PB1);
if (stanje) {
    set_pin_on(PORTB, PB3);
}
else {
    set_pin_off(PORTB, PB3);
}
```

Objasnimo sada programski kod 12.12. U *Boolean* varijablu `stanje` pohranjuje se stanje pina PB1. Ako je stanje pina PB1 visoko (`if (stanje) { }`), tada se pin PB3 postavlja u visoko stanje (`set_pin_on(PORTB, PB3)`). Inače se pin PB3 postavlja u nisko stanje (`set_pin_off(PORTB, PB3)`). Na ovaj način smo kopirali PWM signal pina PB1 na pin PB3. Na kraju prekidne rutina `ISR(PCINT0_vect)` upišite naredbu `pcintPINBOld = PINB;`. Ovom naredbom sprema se aktualna vrijednost registra PINB kako bi se pri sljedećoj promjeni signala na bilo kojem pinu porta B moglo detektirati koji je pin izazvao prekid.

Sada nam preostaje provesti ispis poruka na LCD displeju. U beskonačnoj `while` petlji napišite uvjetovano grananje `if (lcdUpdate) { }`, a unutar njega upišite sljedeće naredbe:

- `lcdHome();` - postavite ispis na početak LCD displeja (s obzirom da su sve poruke jednako duge, dovoljno je prepisati poruku jednu preko druge bez brisanja LCD displeja),
- `lcdprintf("%s", LCDispis);` - ispiši znakovni niz u varijabli `LCDispis`,
- `lcdUpdate = false;` - varijabla `lcdUpdate` se postavlja u vrijednost `false` kako bismo onemogućili ispis na LCD displeju do nove poruke u varijabli `LCDispis`.

Unutar beskonačne `while` petlje mijenja se faktor vođenja za kanal A, odnosno za pin PB1.

Ako ste slijedili gore navedene korake, Vaša datoteka `vjezba122.cpp` treba biti ista kao programski kod 12.13.

Programski kod 12.13: Program kojim se provodi PCINT prekidi na pinovima PB1, PD2, PD4 i PD5

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "ADC/adc.h"
#include <string.h>

volatile bool lcdUpdate = true;
char LCDispis[6] = "READY";

// prekidna rutina za PCINT0
ISR(PCINT0_vect) {
    // detekcija promjene brida na portu B
    uint8_t PINBChanged = pcintPINBOld ^ PINB;
    // ako se dogodio brid na pinu PB1
    if (PINBChanged & (1 << PB1)) {
        bool stanje = get_pin(PINB, PB1);
        if (stanje) {
            set_pin_on(PORTB, PB3);
        }
        else {
            set_pin_off(PORTB, PB3);
        }
    }
    // osvježenje stare vrijednosti registra PINB
    pcintPINBOld = PINB;
}

// prekidna rutina za PCINT2
ISR(PCINT2_vect) {
    // detekcija promjene brida na portu D
    uint8_t PINDChanged = pcintPINDOld ^ PIND;
    // ako se dogodio brid na pinu PD2
    if (PINDChanged & (1 << PD2)) {
        if (get_pin(PIND, PD2)) {
            strcpy(LCDispis, "PD2 R");
        } else {
            strcpy(LCDispis, "PD2 F");
        }
    }
    // ako se dogodio brid na pinu PD4
    if (PINDChanged & (1 << PD4)) {
        if (get_pin(PIND, PD4)) {
            strcpy(LCDispis, "PD4 R");
            } else {
            strcpy(LCDispis, "PD4 F");
        }
    }
    // ako se dogodio brid na pinu PD5
    if (PINDChanged & (1 << PD5)) {
        if (get_pin(PIND, PD5)) {
            strcpy(LCDispis, "PD5 R");
            } else {
            strcpy(LCDispis, "PD5 F");
        }
    }
    // omogućenje ispisa na LCD
    lcdUpdate = true;
    // osvježenje stare vrijednosti registra PIND
    pcintPINDOld = PIND;
}
```

```
void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    //konfiguracija izlaza PB1 i PB3
    DDRB |= (1 << PB1) | (1 << PB3);
    //konfiguracija ulaza PD2, PD4 i PD5
    DDRD &= ~((1 << PD2) | (1 << PD4) | (1 << PD5));
    //pull up otpornici ukljuceni na PD2, PD4 i PD5
    PORTD |= (1 << PD2) | (1 << PD4) | (1 << PD5);
    // postavljanje Phase Correct PWM načina rada za timer1 - 10 bit
    timer1PhaseCorrectPWM10bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_64);
    // neinvertirajuci PWM signal na PB1 (OC1A)
    timer1OC1AEnableNonInvertedPWM();
    interruptEnable(); // globalno omogućenje prekida
    // omogući PCINT prekid na pinovima iz skupine PCINT7..0
    PCICR |= (1 << PCIE0);
    // omogući PCINT prekid na pinovima iz skupine PCINT23..16
    PCICR |= (1 << PCIE2);
    // omogućenje pinova koji će generirati prekid
    PCMSK0 |= (1 << PCINT1); // omogući PCINT prekid na pinu PCINT1 (PB1)
    PCMSK2 |= (1 << PCINT18); // omogući PCINT prekid na pinu PCINT18 (PD2)
    PCMSK2 |= (1 << PCINT20); // omogući PCINT prekid na pinu PCINT20 (PD4)
    PCMSK2 |= (1 << PCINT21); // omogući PCINT prekid na pinu PCINT21 (PD5)
    // spremanje inicijalnih vrijednosti registara PINB i PIND
    pcintPINBOld = PINB;
    pcintPINDOld = PIND;
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    float D;
    while (1) {
        // ispis na LCD displej
        if (lcdUpdate) {
            lcdHome();
            lcdprintf("%s", LCDispis);
            lcdUpdate = false;
        }
        D = adcReadScale0To100(ADC0);
        // osvjezavanje faktora vodenja kroz registar OCR1A
        timer1OC1ADutyCycle(D);
    }
}
```

Prevedite datoteku `vjezba122.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Pritisnite tipkala T1, T2 i T3 i pratite ispis na LCD displeju. Na primjer, kada pritisnete tipkalo T1, na LCD displeju će se ispisati tekst `PD4 F`. Kada otpustite tipkalo T1, na LCD displeju će se ispisati tekst `PD4 R`. Promijenite zakret potenciometra i pratite intenzitet crvene i zelene LED diode. Primijetit ćete da se intenzitet mijenja na obje LED diode što znači da smo uspješno kopirali PWM signal s pina PB1 na pin PB3.

U programskom kodu 12.12 smo za čitanje i postavljanje stanja pina koristili makronaredbe `get_pin`, `set_pin_on` i `set_pin_off`. Teoretski smo mogli koristiti funkcije `digitalRead()` i `digitalWrite()`. Kod ovog pristupa problem se javlja kada je faktor vođenja blizu 0 i kada je faktor vošenja blizu 100 %. U tim situacijama promjena stanja PWM signala je vremenski jako brza. Ako se koristi funkcijski pristup moguć je neispravan rad u ovim krajnjim slučajevima

jer je vrijeme izvođenja funkcija digitalRead() i digitalWrite() puno veće od makronaredbi get_pin, set_pin_on i set_pin_off. Provjerite ovu tvrdnju!

Zakomentirajte naredbu PCMSK0 |= (1 << PCINT1); te ponovno prevedite datoteku vjezba122.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Primijetite da zelena LED dioda više ne kopira stanje crvene LED diode jer je softverski PCINT prekid na pinu PB1 onemogućen.

U teorijskom dijelu ovog poglavlja prikazali smo funkcije kojima se mogu konfigurirati PCINT prekidi. Konfiguracija PCINT prekida koji smo koristili u ovoj vježbi može se provesti funkcijskim pristupom.

Programski kod 12.14: Konfiguracija PCINT prekida za pinove PB1, PD2, PD4 i PD5 - funkcijski pristup

```
// omogući PCINT prekid na pinovima iz skupine PCINT7..0
pcintEnable7to0();
// omogući PCINT prekid na pinovima iz skupine PCINT23..16
pcintEnable23to16();
pcintPinEnable7to0(PCINT1); // omogući PCINT prekid na pinu PCINT1 (PB1)
pcintPinEnable23to16(PCINT18); // omogući PCINT prekid na pinu PCINT18 (PD2)
pcintPinEnable23to16(PCINT20); // omogući PCINT prekid na pinu PCINT20 (PD4)
pcintPinEnable23to16(PCINT21); // omogući PCINT prekid na pinu PCINT21 (PD5)
// spremanje inicijalnih vrijednosti registara PINB, PINC i PIND
pcintInit();
```

Zamijenite postojeću konfiguraciju PCINT prekida s konfiguracijom prikazanom programskim kodom 12.14. Prevedite datoteku vjezba122.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Funkcionalnost sustava trebala bi ostati nepromijenjena.

Zatvorite datoteku vjezba122.cpp i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 13

Povezivanje odabranih elektroničkih modula na mikroupravljač

Postoji velik izbor perifernih modula u obliku senzora i aktuatora, koji se mogu ugraditi u ugradbene sustave kojima upravlja mikroupravljač kao što je ATmega328P. U ovoj vježbi korišteni su sljedeći periferni moduli:

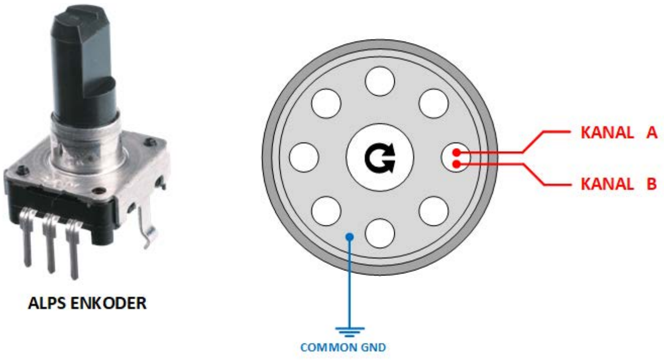
- Rotacijski enkoder
- Tranzistor kao sklopka i relej
- Ultrazvučni senzor
- Numerički displej i posmačni registri
- Servo motor
- RGB dioda

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `Moduli.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `Moduli.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\Moduli\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Povezivanje odabranih elektroničkih modula na mikroupravljač`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

13.1 Rotacijski enkoder

Rotacijski enkoder je elektro-mehanička komponenta koja pretvara kutnu poziciju ili zakret u neki kratkotrajni digitalni signal. Mogli bismo reći da je rotacijski enkoder - senzor zakreta. Zbog svoje robusnosti i mogućnosti precizne kontrole, rotacijski enkoder se koriste u mnogim industrijskim i hobi primjenama, kao što su robotika, CNC strojevi, 3D printeri i razne primjene izbornika na uređajima. Postoje apsolutni i inkrementalni enkoderi. Apsolutni enkoder ima kodiranu ploču na osovini preko koje je uvijek poznata pozicija zakreta osovine, dok inkrementalni enkoder generira impulse tijekom zakreta. Enkoder u ovoj vježbi je inkrementalnog tipa.

Rotacijski enkoder pomalo podsjeća na potenciometar. Glavna razlika je što rotacijski enkoder nema krajnje točke, dok potenciometar ima i to obično na manjem kutu od cijelog kruga mogućeg zakreta. Rotacijski enkoder u ovoj će se vježbi koristiti za zadavanje neke referentne vrijednosti. Na zakret enkodera vrijednost će se povećavati ili umanjivati. Rotacijski enkoder ima dva kanala i prilikom zakreta generira impulse na A i B kanalima. Kombinacijom stanja na kanalima A i B, može se odrediti smjer zakreta enkodera. Na enkoderu postoji i tipkalo koje radi kao i obično tipkalo, a aktivira se kada je osovina enkodera pritisnuta. Rotacijski enkoder korišten u ovoj vježbi i grafički prikaz principa rada istog prikazani su na slici 13.1.

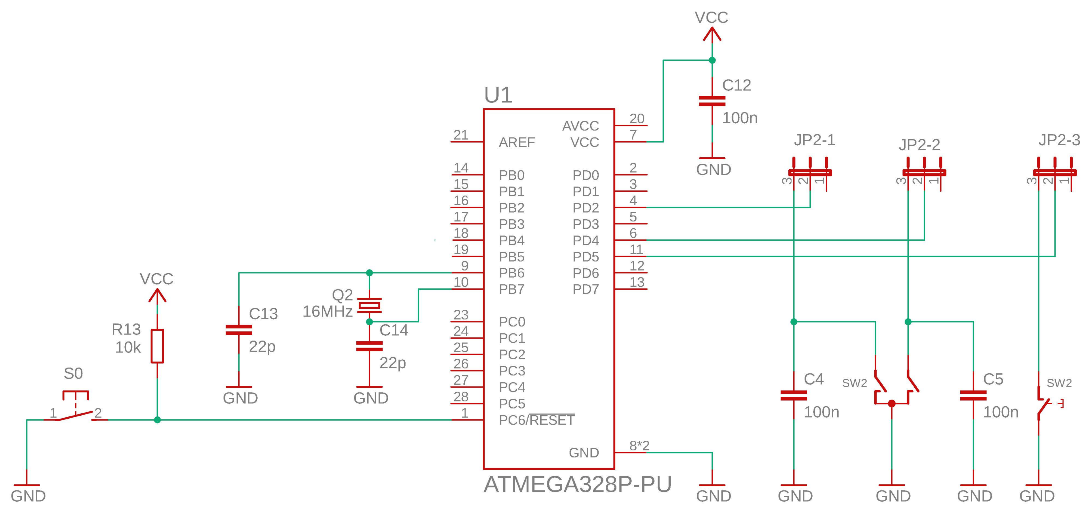


Slika 13.1: Rotacijski enkoder ALPS i princip rada rotacijskog enkodera

Rotacijski enkoder ima tri izlazna pina koji su priključeni na mikroupravljač ATmega328P na sljedeći način:

- A kanal rotacijskog enkodera je priključen na pin mikroupravljača `PD2` (`INT0`),
- B kanal rotacijskog enkodera je priključen na pin mikroupravljača `PD4` i
- tipkalo rotacijskog enkodera je priključeno na pin mikroupravljača `PD5`.

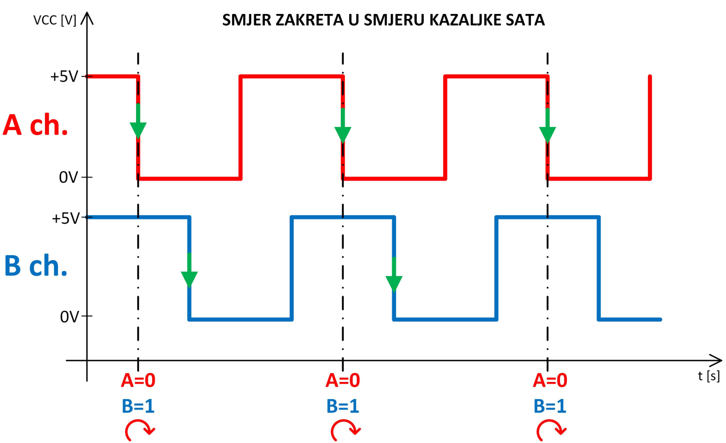
Prilikom korištenja rotacijskog enkodera na razvojnom sučelju potrebno je postaviti kratkospojnik `JP2` tako da spaja srednje trnove i trnove nazvane `ENK`. Shema spajanja rotacijskog enkodera s pripadajućim izlaznim signalima prikazana je na slici 13.2.



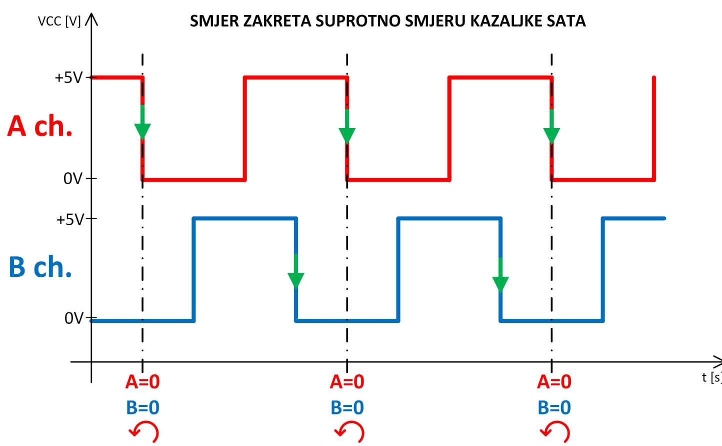
Slika 13.2: Shema spajanja rotacijskog enkodera na mikroupravljač ATmega328P

Da bi se promijenila vrijednost neke varijable pomoću zakreta rotacijskog enkodera, potrebno

je obraditi izlazne signale rotacijskog enkodera. Razlikujemo 2 kanala enkodera: A i B kanal. Kanal A je kanal koji prvi detektira zakret i zbog toga se koristi kao signal za poziv vanjske prekidne rutine. Vanjske prekidne rutine mogu se pozvati na rastući, padajući ili rastući i padajući brid signala. U ovoj vježbi koristi se poziv vanjske prekidne rutine na padajući brid signala jer se u stanju kada nema zakreta, na ulaznom pinu mikroupravljača nalazi visoko stanje signala zbog uključenog pripadajućeg priteznog otpornika. Stanje kanala B u kombinaciji sa stanjem kanala A može biti informacija za zaključivanje u kojem smjeru se dogodio zakret osovine rotacijskog enkodera. Ukoliko se na kanalu A obrađuje samo padajući ili samo rastući brid signala, tada se radi o načinu rada u 1. kvadrantu. Ukoliko se na kanalu A obrađuju rastući i padajući brid signala, tada se radi o načinu rada u 2. kvadrantu. Postoji i način rada u 4 kvadranta, kada se oba kanala rotacijskog enkodera obrađuju uz pomoć vanjskih prekidnih rutina. Način rada u 4 kvadranta predstavlja najprecizniji mogući način rada. U ovoj vježbi zbog jednostavnosti, koristit će se način rada u 1. kvadrantu, što znači da će se promatrati padajući brid signala na kanalu A i stanje signala na kanalu B za određivanje smjera zakreta rotacijskog enkodera. Prema sljedećem pravilu mogu se odrediti referentni smjerovi vrtnje rotacijskog enkodera: ako je na A kanalu nisko stanje signala (padajući brid), dok je na B kanalu visoko stanje signala, tada je smjer vrtnje u smjeru kazaljke na satu. U drugom slučaju, kad je na A kanalu nisko stanje signala (padajući brid), dok je na B kanalu nisko stanje signala, tada je smjer vrtnje suprotan smjeru kazaljke na satu. Signali na kanalima rotacijskog enkodera i referentni smjerovi vrtnje na vremenskim grafovima prikazani su na slici 13.3.

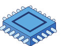


(a) Smjer vrtnje kazaljke na satu (A=LOW, B=HIGH)



(b) Smjer vrtnje suprotan kazaljci na satu (A=LOW, B=LOW)

Slika 13.3: Referentni smjerovi zakreta enkodera

 Vježba 13.1

Potrebno je napraviti program koji će uvećavati i umanjivati neku varijablu pomoću rotacijskog enkodera u intervalu od 0 do 100. Potrebno je prikazati vrijednost te varijable u prvom retku LCD displeja uz indikator smjera vrtnje rotacijskog enkodera. U drugom retku LCD displeja prikazana je traka završenosti (engl. *loading bar*), koja odgovara vrijednosti navedene varijable. Pritiskom na tipku rotacijskog enkodera, vrijednost varijable treba postaviti na nulu. Shema spajanja rotacijskog enkodera na razvojno okruženje upravljano mikroupravljačem ATmega328P prikazana je na slici 13.2.

U projektnom stablu otvorite datoteku `vjezba131.cpp`. Omogućite samo prevođenje datoteke `vjezba131.cpp`. Početni sadržaj datoteke `vjezba131.cpp` prikazan je programskim kodom 13.1.

Programski kod 13.1: Početni sadržaj datoteke `vjezba131.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"

// najmanja moguca vrijednost enkodera
// najveca moguca vrijednost enkodera

// zastavica koja oznacava trenutni smjer vrtnje enkodera
// zastavica koja oznacava promijenu vrijednosti enkodera
// varijabla za spremanje trenutne vrijednosti enkodera

// funkcija za azuriranje trenutne vrijednosti enkodera
void updateEncoder(int8_t dir) {
    if (dir == 1) { // smjer u smjeru kazaljke na satu

        // ako je vrijednost enkodera manja od najvece moguce

        // uvecaj vrijednost enkodera za 1
    } else {
        // ako je vrijednost enkodera veca od najmanje moguce

        // smanji vrijednost enkodera za 1
    }

    // postavi zastavicu u logicku istinu, kad zavrsi azuriranje
}

// prekidna rutina -> promjena signala na kanalu A
ISR(INT0_vect) {

    if (get_pin(PIND, PD4)) {       // ako je B - HIGH
    // smjer jkazaljke na satu
    } else {
    // smjer suprotnom kazaljci na satu
    }
    // poziv funkcije za azuriranje vrijednosti enkodera

}

void init() {       // inicijalizacija mikroracunala

    // inicijalizacija LCD displeja
```

```
    // INT0 na pinu PD2 je omogucen
    // padajuci brid PD2 aktivira
    // globalno omogucavanje prekida

    config_input(DDRD, PD2);    // pin PD2 ulazni
    config_input(DDRD, PD4);    // pin PD4 ulazni
    config_input(DDRD, PD5);    // pin PD5 ulazni

    pull_up_on(PORTD, PD2);     // pull-up ulazni pin PD2
    pull_up_on(PORTD, PD4);     // pull-up ulazni pin PD4
    pull_up_on(PORTD, PD5);     // pull-up ulazni pin PD5
}

int main(void) {

    init();     // inicijalizacija uC

    uint8_t loading = 0;    // varijabla za simulaciju loading
    char dirChar;       // pomocna varijabla za znak strelice

    while (1) {

        if(encUpdated) {

            // ciscenje cijelog LCD displeja
            // pokazivac LCD na koordinate 0,0
            // ispis trenutne vrijednosti enkodera na displej

            if (encDir == 1) {  // smjer kazaljke na satu
                dirChar = 62;   // ASCII kod strelica desno >
            } else {
                dirChar = 60;   // ASCII kod strelica lijevo <
            }
            lcdGotoXY(1,11);    // pokazivac LCD na koordinate 1,11
            // ispis trenutnog smjera enkodera na displej

            // izracun vrijednosti za prikaz loading bara

            // ispis loading bara na LCD displej
            for(uint8_t i = 0; i < loading; i++) {
                lcdChar(255);
            }

            encUpdated = false;     // zastavica za ispis
        }

        // pritisnuto tipkalo enkodera, padajuci brid detekcija
        if(isFallingEdge(D5))
        {
            encValue = 0;   // postavi vrijednost na nulu
            encUpdated = true;  // zastavica za azuriranje aktivna
        }
    }
}
```

Globalno su definirane tri varijable koje su vidljive svim funkcijama uključujući glavnu funkciju. Varijabla `encDir` ima ulogu pomoćne zastavice u koju će biti pohranjena trenutna referenca smjera zakreta enkodera. Sljedeća varijabla je `encUpdated` koja ima ulogu zastavice koja označava da li je izvršeno ažuriranje varijable u koju je pohranjena trenutna vrijednost. Varijabla za pohranu trenutne vrijednosti koju umanjuje i uvećava rotacijski enkoder nazvana je `encValue` i može imati vrijednost od 0 do 100, što je i definirano globalnim konstantama `ENCODER_LOW_THRESHOLD` i `ENCODER_HIGH_THRESHOLD`. Globalno definirane konstante i varijable

prikazane su programskim kodom 13.2. Programski kod 13.2 upišite u datoteku `vjezba131.cpp` nakon makronaredbe `#include "LCD/lcd.h"`.

Programski kod 13.2: Globalno definirane konstante i varijable

```
// najmanja moguca vrijednost enkodera
#define ENCODER_LOW_THRESHOLD 0
// najveca moguca vrijednost enkodera
#define ENCODER_HIGH_THRESHOLD 100

// zastavica koja oznacava trenutni smjer vrtnje enkodera
volatile int8_t encDir = 0;

// zastavica koja oznacava promijenu vrijednosti enkodera
volatile bool encUpdated = true;

// varijabla za spremanje trenutne vrijednosti enkodera
volatile int8_t encValue = 0;
```

Funkcija `updateEncoder()` za ažuriranje trenutne vrijednosti enkodera ima dva moguća razvoja svojeg algoritma. Ukoliko je smjer vrtnje u smjeru kazaljke na satu, tada će se varijabla `encValue` povećavati sve dok ne postigne vrijednost konstante `ENCODER_HIGH_THRESHOLD`. Drugi razvoj algoritma je ako je smjer vrtnje suprotan smjeru kazaljke na satu, tada će se varijabla `encValue` smanjivati sve dok ne postigne vrijednost konstante `ENCODER_LOW_THRESHOLD`. Kada je izvršeno uvećanje ili umanjenje varijable `encValue`, postavlja se zastavica `encUpdated` za označavanje završenog ažuriranja na logičku istinu. Potpuna definicija funkcije `updateEncoder()` prikazana je programskim kodom 13.3. Osvježite funkciju `updateEncoder()` u datoteci `vjezba131.cpp` kako bi bila istovjetna programskom kodu 13.3.

Programski kod 13.3: Definicija funkcije `updateEncoder()`

```
// funkcija za azuriranje trenutne vrijednosti enkodera
void updateEncoder(int8_t dir) {

    if (dir == 1) { // smjer u smjeru kazaljke na satu
        // ako je vrijednost enkodera manja od najvece moguce
        if (encValue < ENCODER_HIGH_THRESHOLD) {
            encValue++;     // uvecaj vrijednost enkodera za 1
        }
    } else {
        // ako je vrijednost enkodera veca od najmanje moguce
        if (encValue > ENCODER_LOW_THRESHOLD) {
            encValue--;     // smanji vrijednost enkodera za 1
        }
    }
    // postavi zastavicu u logicku istinu, kad zavrsi azuriranje
    encUpdated = true;
}
```

Na pojavu padajućeg brida signala na vanjskom digitalnom pinu `INT0` (`PD2`), aktivira se prekidna rutina `ISR(INT0_vect)`. Unutar prekidne rutine provjerava se da li je kanal B u visokom ili niskom stanju signala na pinu `PD4`, prema čemu je moguće odlučiti koji je smjer zakreta enkodera. Informacija o smjeru zakreta će se pohraniti u varijablu ili zastavicu `encDir`. Nakon što je određen smjer zakreta enkodera, informacija se prosljeđuje funkciji `updateEncoder()` za ažuriranje trenutne vrijednosti enkodera unutar njenog poziva. Definicija prekidne rutine `ISR(INT0_vect)` prikazana je programskim kodom 13.4. Osvježite prekidnu rutinu `ISR(INT0_vect)` u datoteci `vjezba131.cpp` kako bi bila istovjetna programskom kodu 13.4.

Programski kod 13.4: Definicija prekidne rutine `ISR(INT0_vect)`

```
// prekidna rutina -> promjena signala na kanalu A
ISR(INT0_vect) {

    if (get_pin(PIND, PD4)) {       // ako je B - HIGH
        encDir = 1;     // smjer jkazaljke na satu
    } else {
        encDir = 0;     // smjer suprotnom kazaljci na satu
    }
    // poziv funkcije za azuriranje vrijednosti enkodera
    updateEncoder(encDir);
}
```

U funkciji `init()` za inicijalizaciju mikroupravljača i periferije konfigurirana je vanjska prekidna rutina `INT0`. Da bi prekidna rutina bila dostupna za korištenje potrebno je istu omogućiti funkcijom `int0Enable()`. Potrebno je odrediti na koji brid signala vanjska prekidna rutina će biti pozvana. To može biti rastući, padajući ili oba brida signala spojena na pin `PD2` vanjske prekidne rutine `INT0`. U ovoj vježbi vanjski prekid `INT0` biti će pozvan na padajući brid signala generiranog rotacijskim enkoderom (kanal A). Da bi vanjski prekidi bili funkcionalni, potrebno je i globalno omogućiti prekide. Rotacijski enkoder ima 2 kanala i tipkalo koji su spojeni na mikroupravljač. Potrebno je inicijalizirati 3 ulazna pina i uključiti pritezne otpornike na odgovarajućim pinovima `PD2`, `PD4` i `PD5`. Potpuna definicija inicijalizacijske funkcije `init()` prikazana je programskim kodom 13.5. Osvježite funkciju `init()` u datoteci `vjezba131.cpp` kako bi bila istovjetna programskom kodu 13.5.

Programski kod 13.5: Definicija inicijalizacijske funkcije `init()`

```
void init() {       // inicijalizacija mikroracunala

    lcdInit();      // inicijalizacija LCD displeja

    int0Enable();   // INT0 na pinu PD2 je omogucen
    int0FallingEdge();  // padajuci brid PD2 aktivira
    interruptEnable();  // globalno omogucavanje prekida

    config_input(DDRD, PD2);    // pin PD2 ulazni
    config_input(DDRD, PD4);    // pin PD4 ulazni
    config_input(DDRD, PD5);    // pin PD5 ulazni

    pull_up_on(PORTD, PD2);     // pull-up ulazni pin PD2
    pull_up_on(PORTD, PD4);     // pull-up ulazni pin PD4
    pull_up_on(PORTD, PD5);     // pull-up ulazni pin PD5
}
```

U glavnoj programskoj funkciji `main()` pozvana je prethodno opisana funkcija za inicijalizaciju i definirane su dvije pomoćne varijable, `loading` za prikaz trake završenosti na LCD displeju i `dirChar` za pohranu znaka za smjer u svrhu prikaza na LCD displeju. U beskonačnoj programskoj petlji `while()` provjerava se stanje zastavice koja označava da li je završeno ažuriranje vrijednosti neke varijable, uzrokovano promjenom na rotacijskom enkoderu. Ukoliko jest, ispisati će se nova, trenutna vrijednost varijable nazvane `encValue`. U prvom retku LCD displeja ispisati će se poruka, na primjer `Enc:  49 Dir:  >`. Prvi dio poruke je trenutna vrijednost varijable `encValue`, a drugi dio poruke je oznaka smjera zakreta rotacijskog enkodera. U drugom retku LCD displeja prikazuje se traka završenosti ili *Loading bar*. Ispis trake završenosti je napravljen tako da *for* petlja ispisuje kvadratne znakove sa svim popunjenim pikselima na zaslon (ASCII kod 255), toliko puta koliko je vrijednost varijable `loading`. Vrijednost varijable `encValue` potrebno je pomnožiti s 16 (16 znakova LCD displeja) te podijeliti sa 100,0 (najveća moguća vrijednost varijable `encValue`) kako bi se izračunala vrijednost varijable `loading`, do koje će *for* petlja ispisivati znakove na LCD displej. Zadnja provjera u glavnoj programskoj petlji `while()` je stanje tipkala na rotacijskom enkoderu. Ukoliko je pritisnuto tipkalo, vrijednost

varijable `encValue` će se postaviti na 0 i zastavica za završeno ažuriranje će se postaviti na logičku istinu, kako bi se ažurirana vrijednost varijable `encValue` ispisala na LCD displej. Potpuna definicija glavne programske funkcije `main()` prikazana je programskim kodom 13.6. Osvježite funkciju `main()` u datoteci `vjezba131.cpp` kako bi bila istovjetna programskom kodu 13.6.

Programski kod 13.6: Definicija glavne programske funkcije `main()`

```
int main(void) {

    init();     // inicijalizacija uC

    uint8_t loading = 0;    // varijabla za simulaciju loading

    char dirChar;       // pomocna varijabla za znak strelice

    while (1) {

        if(encUpdated) {

            lcdClrScr();    // ciscenje cijelog LCD displeja
            lcdHome();      // pokazivac LCD na koordinate 0,0

            // ispis trenutne vrijednosti enkodera na displej
            lcdprintf("Enc: %d", encValue);

            if (encDir == 1)    // smjer kazaljke na satu
                dirChar = 62;   // ASCII kod strelica desno >
            else
                dirChar = 60;   // ASCII kod strelica lijevo <

            lcdGotoXY(1,11);    // pokazivac LCD na koordinate 1,11
            // ispis trenutnog smjera enkodera na displej
            lcdprintf("Dir: %c\n", dirChar);

            // izracun vrijednosti za prikaz loading bara
            loading = 16 * encValue / 100.0;

            // ispis loading bara na LCD displej
            for(uint8_t i = 0; i < loading; i++) {
                lcdChar(255);
            }

            encUpdated = false;     // zastavica za ispis
        }

        // pritisnuto tipkalo enkodera, padajuci brid detekcija
        if(isFallingEdge(D5))
        {
            encValue = 0;   // postavi vrijednost na nulu
            encUpdated = true;  // zastavica za azuriranje aktivna
        }
    }
}
```

Ako ste slijedili navedene korake vaša će datoteka `vjezba131.cpp` biti istovjetna programskom kodu 13.7.

Programski kod 13.7: Potpuni sadržaj datoteke `vjezba131.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"
```

```
// najmanja moguca vrijednost enkodera
#define ENCODER_LOW_THRESHOLD 0
// najveca moguca vrijednost enkodera
#define ENCODER_HIGH_THRESHOLD 100

// zastavica koja oznacava trenutni smjer vrtnje enkodera
volatile int8_t encDir = 0;
// zastavica koja oznacava promijenu vrijednosti enkodera
volatile bool encUpdated = true;
// varijabla za spremanje trenutne vrijednosti enkodera
volatile int8_t encValue = 0;

// funkcija za azuriranje trenutne vrijednosti enkodera
void updateEncoder(int8_t dir) {

    if (dir == 1) { // smjer u smjeru kazaljke na satu
        // ako je vrijednost enkodera manja od najvece moguce
        if (encValue < ENCODER_HIGH_THRESHOLD) {
            encValue++;     // uvecaj vrijednost enkodera za 1
        }
    } else {
        // ako je vrijednost enkodera veca od najmanje moguce
        if (encValue > ENCODER_LOW_THRESHOLD) {
            encValue--;     // smanji vrijednost enkodera za 1
        }
    }
    // postavi zastavicu u logicku istinu, kad zavrsi azuriranje
    encUpdated = true;
}

// prekidna rutina -> promjena signala na kanalu A
ISR(INT0_vect) {

    if (get_pin(PIND, PD4)) {       // ako je B - HIGH
        encDir = 1;     // smjer jkazaljke na satu
    } else {
        encDir = 0;     // smjer suprotnom kazaljci na satu
    }
    // poziv funkcije za azuriranje vrijednosti enkodera
    updateEncoder(encDir);
}


void init() {       // inicijalizacija mikroracunala

    lcdInit();      // inicijalizacija LCD displeja

    int0Enable();   // INT0 na pinu PD2 je omogucen
    int0FallingEdge();  // padajuci brid PD2 aktivira
    interruptEnable();  // globalno omogucavanje prekida

    config_input(DDRD, PD2);    // pin PD2 ulazni
    config_input(DDRD, PD4);    // pin PD4 ulazni
    config_input(DDRD, PD5);    // pin PD5 ulazni

    pull_up_on(PORTD, PD2);     // pull-up ulazni pin PD2
    pull_up_on(PORTD, PD4);     // pull-up ulazni pin PD4
    pull_up_on(PORTD, PD5);     // pull-up ulazni pin PD5
}

int main(void) {
```

```
    init();     // inicijalizacija uC

    uint8_t loading = 0;    // varijabla za simulaciju loading
    char dirChar;       // pomocna varijabla za znak strelice

    while (1) {

        if(encUpdated) {

            lcdClrScr();    // ciscenje cijelog LCD displeja
            lcdHome();      // pokazivac LCD na koordinate 0,0
            // ispis trenutne vrijednosti enkodera na displej
            lcdprintf("Enc: %d", encValue);

            if (encDir == 1) {  // smjer kazaljke na satu
                dirChar = 62;   // ASCII kod strelica desno >
            } else {
                dirChar = 60;   // ASCII kod strelica lijevo <
            }

            lcdGotoXY(1,11);    // pokazivac LCD na koordinate 1,11
            // ispis trenutnog smjera enkodera na displej
            lcdprintf("Dir: %c\n", dirChar);

            // izracun vrijednosti za prikaz loading bara
            loading = 16 * encValue / 100.0;

            // ispis loading bara na LCD displej
            for(uint8_t i = 0; i < loading; i++) {
                lcdChar(255);
            }

            encUpdated = false;     // zastavica za ispis
        }

        // pritisnuto tipkalo enkodera, padajuci brid detekcija
        if(isFallingEdge(D5))
        {
            encValue = 0;   // postavi vrijednost na nulu
            encUpdated = true;  // zastavica za azuriranje aktivna
        }
    }
}
```

Prevedite datoteku `vjezba131.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Provjerite da li vrijednost koju postavljate enkoderom broji do najmanje vrijednosti 0 i najveće vrijednosti 100. Provjerite da li radi ispravno prikazivanje smjera vrtnje rotacijskog enkodera. Provjerite da li se ispravno popunjava traka završenosti te da li odgovara vrijednosti varijable u prvom retku LCD displeja. Nakon što ste testirali vježbu, zatvorite datoteku `vjezba131.cpp` i onemogućite prevođenje ove datoteke.

13.2 Tranzistor kao sklopka i relej

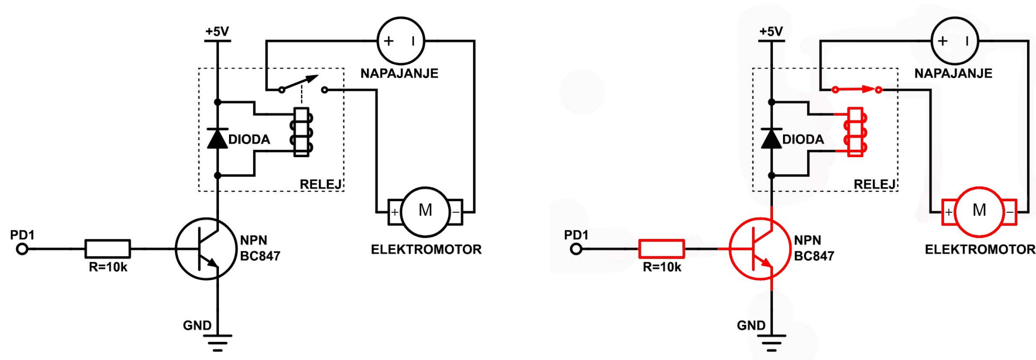
U ugradbenim sustavima kojima upravljaju mikroupravljači često je potrebno upravljati periferijom (trošilima) visoke snage. Za takav zadatak potreban je posrednik koji može podnijeti veću snagu od mikroupravljača. Postoji uobičajen način rješavanja ove potrebe, a to je upravljanje relejem posredno preko tranzistora u načinu rada sklopke, skraćeno tranzistor kao sklopka i relej. Ovaj način upravljanja snažnijim trošilima je primjenjiv za niske frekvencije rada.

Za više frekvencije rada upotrebljava se MOSFET tip tranzistora umjesto bipolarnih tranzistora [1].

Bipolarni tranzistor (engl. *Bipolar Junction Transistor; BJT*) može biti u dva stanja, stanje zapiranja i stanje vođenja. Kada se na bazu tranzistora dovede napon (obično 5 V u TTL naponskoj logici), tada tranzistor prelazi iz stanja zapiranja u stanje vođenje. Svaki elektronički element koji je spojen između kolektora i napona napajanja +5V sada će biti u zatvorenom strujnom krugu. Tranzistor kao komponenta ima i ograničenja u svojstvima struje koja može protjecati kroz isti i naponu na koji je spojen u načinu rada sklopke. Upravo zbog tih ograničenih vrijednosti, tranzistor je dobar posrednik u upravljanju sklopkama za veće snage, no ne i dobra samostalna sklopka za strujne krugove koji uključuju trošila većih snaga. Za rješavanje problema upravljanja snažnijim trošilima, uvedena je elektro-mehanička komponenta koja se naziva relej. Sastoji se od metalne kotve koja spaja određene izlazne i ulazne kontakte. Kotva je osjetljiva na elektromagnetske pojave jer je metalna i njome se upravlja na pojavu magnetske sile. Magnetska sila se generira pomoću zavojnice vrlo niske snage (struje) koja generira magnetsku silu kao pojavu uslijed protjecanja struje kroz vodič. U strujni krug bipolarnog tranzistora se može spojiti upravo ta zavojnica, koja će sada biti u zatvorenom strujnom krugu i generirati će magnetska svojstva te time privući kotvu releja i spojiti izlazne i ulazne kontakte te će se time relej uključiti.

Releji mogu uklapati istosmjerne i izmjenične električne uređaje te imaju svoj ograničeni životni vijek zbog mehaničkih svojstava konstrukcijske izvedbe. Životni vijek releja je definiran brojem uključenja i isključenja, što ujedno čini i glavni nedostatak releja [1].

Na slici 13.4 prikazan je primjer spajanja tranzistora u svojstvu sklopke koja preklapa strujni krug releja.



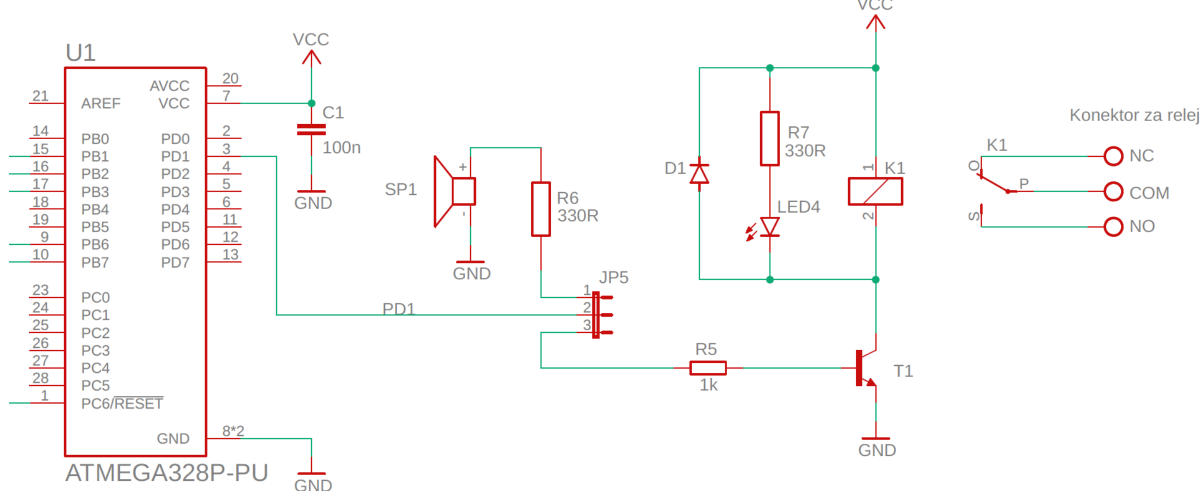
(a) Tranzistor u stanju zapiranja (b) Tranzistor u stanju vođenja

Slika 13.4: Električna shema spajanja releja na mikroupravljač

Strujni krug unutar mikroupravljača koji se može zatvoriti preko pina mikroupravljača može sigurno propustiti najveću struju iznosa 40 mA, što je u skoro svim primjerima upotrebe trošila visokih snaga nedovoljno. Međutim, ta struja (zapravo vrlo manja), dovoljna je za aktiviranje bipolarnog tranzistora. Stoga, spojen je pin `PD1` mikroupravljača preko otpornika iznosa 10 kΩ u seriji na bazu tranzistora BC847. Tranzistor je tipa NPN, stoga je trošilo (relej), spojeno u kolektorski krug tranzistora. Paralelno s trošilom potrebno je spojiti jednosmjerno propusni elektronički element diodu, kako bi spriječili pojavu induciranog napona na zavojnici releja.

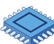
Napon koji upravlja strujnim krugom je TTL naponskih razina, zbog jednostavnosti korištenja s mikroupravljačem koji radi upravno na tim razinama napona. Kao trošilo visoke snage, u ovom primjeru spojen je istosmjerni motor s pripadajućim napajanjem na NO kontakte releja, čiji je rad objašnjen u nastavku.

U ovoj vježbi prikazan je uobičajeni spoj tranzistora kao sklopke u posrednom upravljanju relejem. Shema spajanja tranzistora i releja na mikroupravljač prikazana je na slici 13.5.



Slika 13.5: Shema spajanja bipolarnog tranzistora BC847 s relejem u kolektorskom krugu na mikroupravljač ATmega328P

Kontakte releja treba interpretirati na sljedeći način. Na zajednički COM kontakt releja se uvijek priključi jedna od žica trošila ili napajanja. Ovisno o zadanoj algoritamskoj logici, priključiti ćemo NO ili NC kontakt na drugu žicu trošila. NO kontakt je u stanju mirovanja otvoren, što znači da tada neće ni trošilo biti uključeno jer relej nije uključen. NC kontakt je u stanju mirovanja zatvoren, tada će trošilo biti uključeno, bez obzira što relej nije uključen. Iz perspektive programskog koda mikroupravljača, ako spojimo NO/COM kombinaciju kontakata, visoko stanje signala na pinu `PD1` (5 V) će uključiti trošilo. Ukoliko spojimo NC/COM kombinaciju, tada će visoko stanje signala na pinu `PD1` (5 V) isključiti trošilo. Naravno, vrijede i obrnute kombinacije kada se na izlazni pin mikroupravljača postavlja nisko stanje signala (0 V).

 Vježba 13.2

Potrebno je napisati program za mikroupravljač kojim će se relej spojen na pin `PD1` mikroupravljača, uključivati i isključivati prema sljedećem algoritmu:

- relej se uključuje ako je tipkalo spojeno na pin `PD4` pritisnuto, a relej je prethodno bio isključen,
- relej se uključuje ako je potenciometrom postavljena vrijednost AD pretvorbe - veća ili jednaka 800,
- ako je tipkalo spojeno na pin `PD4` otpušteno ili je vrijednost AD pretvorbe - manja od 800, u tom slučaju relej će ostati isključen.

Shema spajanja mikroupravljača ATmega328P u kolektorski krug s bipolarnim tranzistorom BC847 u svrhu posrednog upravljanja relejem prikazana je na slici 13.5.

U projektnom stablu otvorite datoteku `vjezba132.cpp`. Omogućite samo prevođenje datoteke `vjezba132.cpp`. Početni sadržaj datoteke `vjezba132.cpp` prikazan je programskim kodom 13.8.

Programski kod 13.8: Početni sadržaj datoteke `vjezba132.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {
    // inicijalizacija LCD displeja
    // inicijalizacija AD pretvorbe

    // PD4 konfiguriran kao ulazni pin
    // PD1 konfiguriran kao izlazni pin
    // ukljucen pritezni otpornik za pin PD4
}

int main(void) {

    init();     // inicijalizacija mikroupravljaca

    uint16_t varPot = 0;        // varijabla - AD pretvorbe
    uint8_t relayFlag = 0;      // varijabla - stanje releja

    while (1) {
        // AD pretvorba - kanal ADC0
        if (varPot >= 800)      // iznad 800, relej -> ON
        {
                // ukljucivanje releja pin PD1
            relayFlag = 1;
        }
        else if (!get_pin(PIND, PD4))   // provjera pina PD4
        {
                // ukljucivanje releja PD1
            relayFlag = 1;
        }
        else
        {
                // iskljucivanje releja PD1
            relayFlag = 0;
        }

                // ispis stanja releja
                // ispis AD pretvorbe
        _delay_ms(150);
    }
}
```

U funkciji `init()` potrebno je inicijalizirati periferiju i sklopove mikroupravljača koji se planiraju koristiti u ovoj vježbi. Inicijaliziran je LCD displej, AD pretvornik, tipkalo spojeno na pin PD4 i pripadajući pritezni otpornik te izlazni pin PD1 mikroupravljača koji je spojen na bipolarni tranzistor BC847. Potpuna definicija inicijalizacijske funkcija `init()` prikazana je programskim kodom 13.9. Osvježite funkciju `init()` u datoteci `vjezba132.cpp` kako bi bila istovjetna programskom kodu 13.9.

Programski kod 13.9: Definicija inicijalizacijske funkcije `init()`

```
void init() {

    lcdInit();  // inicijalizacija LCD displeja
    adcInit();  // inicijalizacija AD pretvorbe

    config_input(DDRD, PD4);    // PD4 konfiguriran kao ulazni pin
```

```
    config_output(DDRD, PD1);   // PD1 konfiguriran kao izlazni pin
    pull_up_on(PORTD, PD4); // ukljucen pritezni otpornik za pin PD4

}
```

Relej se uključuje posredno preko bipolarnog tranzistora. Da bi tranzistor promijenio stanje zapiranja u stanje vođenja, potrebno je dovesti na bazu tranzistora napon. Upravljačka struja za aktiviranje tranzistora prolazi putem `PD1` pina mikroupravljača. Da bi se upravljalo pinom mikroupravljača, korištene su dvije funkcije poznate od ranije iz poglavlja **`Digitalni izlazi mikroupravljača ATmega328P`**. To su sljedeće funkcije:

- za uključenje tranzistora funkcija `set_pin_on();` i
- za isključenje tranzistora funkcija `set_pin_off();`.

Upravljački signal može biti generiran na dva načina:

- pritiskom na tipkalo spojeno na pin `PD4` ili
- referentnom vrijednosti AD pretvorbe potenciometra (kanal `ADC0`).

U glavnoj programskoj funkciji `main()` prvo je pozvana funkcija `init()` koja ima ulogu inicijalizacije mikroupravljača. Nakon toga deklarirana je varijabla `varPot` koja se koristi za pohranu trenutne vrijednosti digitalne riječi dobivene AD pretvorbom na kanalu `ADC0`, na koji je spojen potenciometar. Druga definirana varijabla `relayFlag` služi za privremenu pohranu stanja releja, kako bi se mogla što lakše ostvariti algoritamska logika uključivanja i isključivanja releja.

Prilikom svake iteracije beskonačne programske petlje `while`, odrađuje se nekolicina zadataka:

- AD pretvorba na kanalu `ADC0` na koji je spojen potenciometar,
- ukoliko je vrijednost AD pretvorbe veća ili jednaka od 800, uključuje se relej preko pina `PD1`,
- ukoliko je pritisnuto tipkalo spojeno na pin `PD4`, uključuje se relej preko pina `PD1`,
- u svakom drugom slučaju, relej ostaje isključen preko pina `PD1`,
- u prvi redak LCD displeja ispisuje se trenutno stanje varijable `relayFlag` i
- u drugi redak LCD displeja ispisuje se trenutna vrijednost AD pretvorbe na kanalu `ADC0`.

Ako ste slijedili navedene korake vaša će datoteka `vjezba132.cpp` biti istovjetna programskom kodu 13.10.

Programski kod 13.10: Potpuni sadržaj datoteke `vjezba132.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {

    lcdInit();  // inicijalizacija LCD displeja
    adcInit();  // inicijalizacija AD pretvorbe

    config_input(DDRD, PD4);    // PD4 konfiguriran kao ulazni pin
    config_output(DDRD, PD1);   // PD1 konfiguriran kao izlazni pin
```

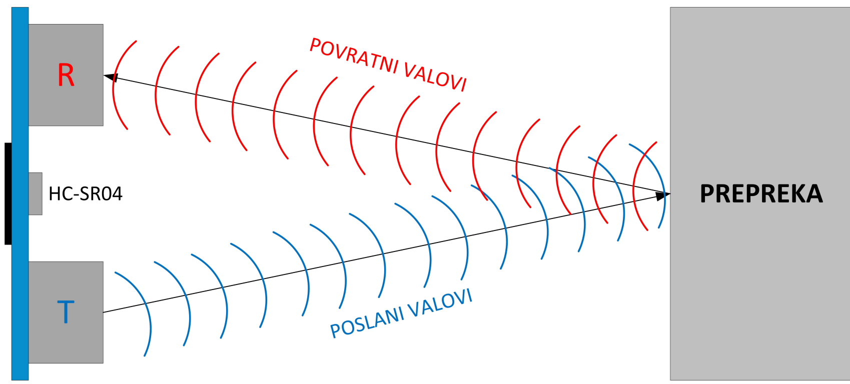
```
    pull_up_on(PORTD, PD4); // ukljucen pritezni otpornik za pin PD4

}

int main(void) {
    init();     // inicijalizacija mikroupravljaca

    uint16_t varPot = 0;        // pomocna varijabla - AD pretvorbe
    uint8_t relayFlag = 0;      // pomocna varijabla - stanje releja

    while (1) {

        varPot = adcRead(ADC0);

        if (varPot >= 800)      // iznad 800, ukljuciti relej, inace ne
        {
            set_pin_on(PORTD, PD1);     // ukljucivanje releja pin PD1
            relayFlag = 1;
        }

        else if (!get_pin(PIND, PD4))   // provjera  ulaznog pina PD4
        {
            set_pin_on(PORTD, PD1);     // ukljucivanje releja pin PD1
            relayFlag = 1;
        }

        else
        {
            set_pin_off(PORTD, PD1);    // iskljucivanje releja pin PD1
            relayFlag = 0;
        }

        lcdClrScr();
        lcdHome();
        lcdprintf("Relay: %u", relayFlag);      // ispis stanja releja
        lcdprintf("\nThreshold: %u", varPot);   // ispis AD pretvorbe
        _delay_ms(150);
    }
}
```

Prevedite datoteku `vjezba132.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P na način da provjerite uključuje li se relej pritiskom na tipkalo 1 (PD4 tipkalo). Drugi način provjere je da namjestite vrijednost digitalne riječi AD pretvorbe uz pomoć potenciometra na vrijednost 800 ili veću. Također, u ovom slučaju bi se trebao uključiti relej. U svim suprotnim slučajevima, relej bi trebao ostati isključen.

Zatvorite datoteku `vjezba132.cpp` i onemogućite prevođenje ove datoteke.

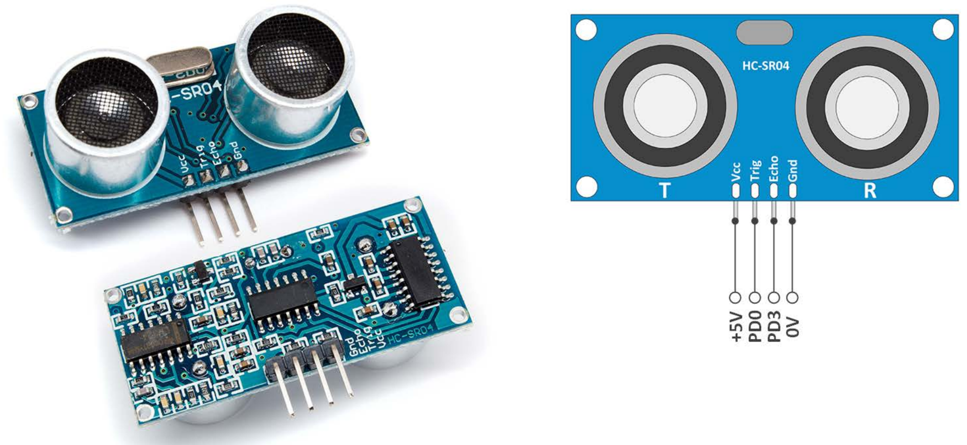
13.3 Ultrazvučni senzor HC-SR04

Ultrazvučni senzori služe za određivanje udaljenosti nekog objekta. Na istom principu rade radari i sonari. Ultrazvučni senzor generira ultrazvučni val visoke frekvencije kojeg odašilje u prostor. Kad ultrazvučni val dođe do prepreke, dio tog vala se odbije nazad prema senzoru u obliku povratnog signala, odnosno jeke (engl. *Echo*). Grafički prikaz rasprostiranja ultrazvučnog vala u prostoru prikazan je na slici 13.6.



Slika 13.6: Rasprostiranje ultrazvučnog vala u prostoru

Senzor ima funkciju odašiljača i prijemnika ultrazvučnih valova jer se sastoji od mikrofona i zvučnika. Prijemnik (mikrofon) detektira trenutak dolaska jeke u povratnoj reflektivnoj putanji. Mjerenjem vremena potrebnog da se jeka reflektira do senzora i poznavanjem iznosa brzine zvuka vrlo jednostavno je matematički doći do prijeđenog puta ultrazvučnog vala koji odgovara udaljenosti senzora do neke prepreke. Cijena takvih senzora je obično dosta visoka, no dostupni su senzori iz hobi kategorije kao što je senzor HC-SR04 koji može mjeriti udaljenosti od 2 cm do 400 cm i niske je nabavne cijene. Senzor HC-SR04 prikazan je na slici 13.7.



Slika 13.7: Senzor HC-SR04

Senzor se sastoji od 4 priključna pina koji su označeni i na slici 13.7:

- `VCC` - napajanje senzora +5 V,
- `TRIG` - pin na koji je potrebno poslati impuls za pokretanje mjerenja udaljenosti,
- `ECHO` - pin na kojem se generira vremenski impuls koji odgovara vremenu u kojem je

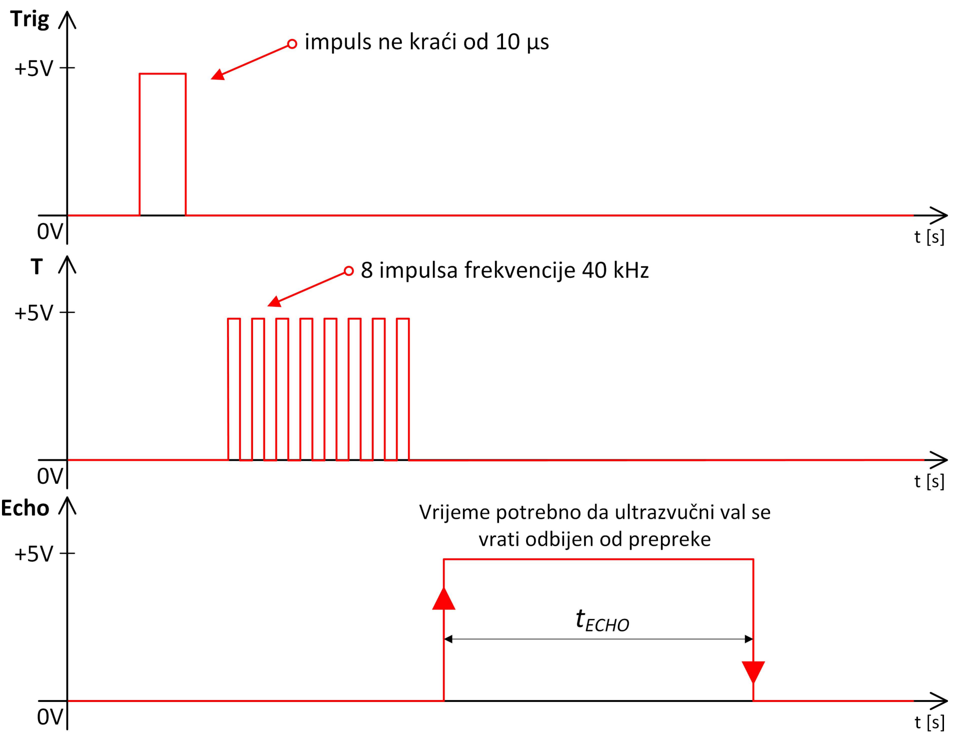
ultrazvučni val došao do prepreke i nazad,

- `GND` - napajanje senzora 0 V.

HC-SR04 ultrazvučni senzor radi na sljedećem principu:

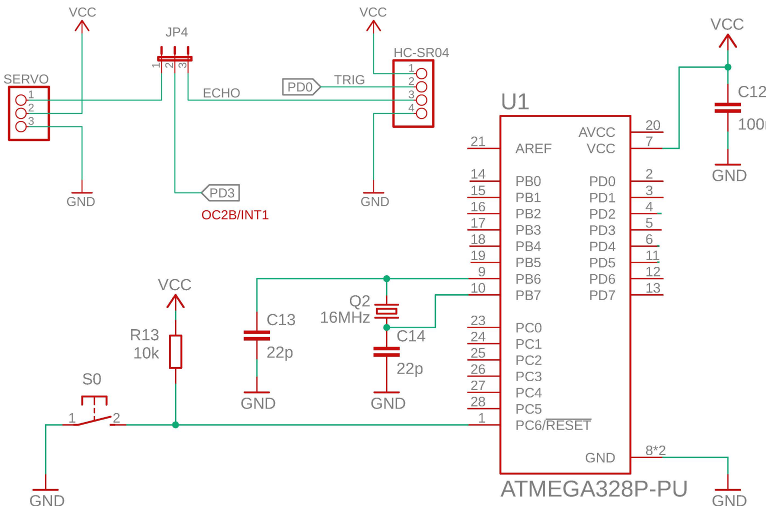
- mikroupravljač šalje impuls visokog stanja signala u trajanju od 10 μs na `TRIG` pin senzora,
- senzor generira 8 impulsa frekvencije 40 kHz na zvučniku u ulozi odašiljača (engl. *Transmitter*),
- nakon završenog generiranja niza od 8 impulsa, izlazni pin senzora *ECHO* postavlja se u visoko stanje u trajanju sve dok ultrazvučni val se ne reflektira do mikrofona u ulozi primatelja (engl. *Receiver*).

Vremenski dijagram signala generiranih na mikroupravljaču i ultrazvučnom senzoru HC-SR04 prikazan je na slici 13.8:



Slika 13.8: Vremenski dijagram signala na senzoru HC-SR04

Shema spajanja ultrazvučnog senzora HC-SR04 i mikroupravljača ATmega328P prikazana je na slici 13.9. Da bismo senzor HCSR04 mogli koristiti potrebno je spojiti kratkospojnik `JP4` između srednjeg trna i trna nazvanog `HCSR04`. Zbog korištenja temperaturnog senzora LM35, potrebno je spojiti kratkospojnik `JP6` između srednjeg trna i trna nazvanog `LM35`.



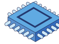
Slika 13.9: Shema spajanja ultrazvučnog senzora HC-SR04 i mikroupravljača ATmega328P

Udaljenost prepreke od ultrazvučnog senzora HC-SR04 može se prikazati relacijom 13.1:

$$d\,[m] = \frac{t_{echo} \cdot v_z}{2} \tag{13.1}$$

gdje je:

- t_{echo} - vrijeme potrebno za povratak poslanog ultrazvučnog vala,
- v_z - brzina zvuka i
- $d\,[m]$ - izračunata udaljenost u metrima.

 Vježba 13.3

Potrebno je napisati program koji će mjeriti udaljenost pomoću ultrazvučnog senzora HC-SR04. Izračun udaljenosti je izravno povezan s brzinom zvuka koja je promjenjiva s obzirom na temperaturu okoline. Stoga, potrebno je izračunati korigiranu udaljenost uz pomoć temperature okoline dobivene AD pretvorbom izlaza senzora temperature LM35. Potrebno je ispisati izračunatu udaljenost bez korekcije i s korekcijom na LCD displeju u mjernoj jedinici centimetara. Shema spajanja ultrazvučnog senzora HC-SR04 i razvojnog okruženja upravljanog mikroupravljačem ATmega328P prikazana je na slici 13.9.

U projektnom stablu otvorite datoteku `vjezba133.cpp`. Omogućite samo prevođenje datoteke `vjezba133.cpp`. Početni sadržaj datoteke `vjezba133.cpp` prikazan je programskim kodom 13.11.

Programski kod 13.11: Početni sadržaj datoteke `vjezba133.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Interrupt/interrupt.h"
#include <util/delay.h>
```

```
// zastavica za signalizaciju da je trigger puls poslan
// pomocna varijabla za spremanje brojenja pulseva
// pomocna zastavica za signalizaciju zavrsenog mjerenja

ISR(INT1_vect) {
    if() {  // ECHO pin -> idle state
        // timer counter registar -> nula
    }
    else {
        // broj pulseva -> pulseCount
        // zastavicom mjerenje zavrseno
    }
}

void HCSR04_trigger() {      // fnc za trigger puls
    // high state na pinu PD0
    _delay_us(20);           // koje traje 20 us
    // low state na pinu PD0
    HCSR04_triggerSent = true;  // zastavica trigger poslan
}

void init(){         // inicijalizacija mikroupravljaca
    // inicijalizacija LCD displeja
    // inicijalizacija AD pretvornika

    lcdClrScr();     // ocisti prethodni zapis na LCD
    lcdHome();       // pokazivac LCD na koordinate 1,1

    // prekidna rutina 1 rastuci i padajuci brid signala ECHO
    // omoguci prekidnu rutinu 1
    // timer 1 -> normalni nacin rada
    // djelitelj frekvencije za timer 1 ce biti 8

    config_output(DDRD,PD0);    // PD0 pin -> izlaz (TRIG)
    config_input(DDRD,PD3);     // PD3 pin -> ulaz (ECHO)

    interruptEnable();      // globalno omogucenje rutina
}

int main(void){

    init();     // poziv fnc inicijalizacija mikroupravljaca

    // varijabla za vremenski istek zahtjeva za trigger puls
    // varijabla za pohranu proteklog vremena koje se mjeri timerom
    // varijabla za izracunatu vrijednost udaljenosti bez korekcije
    // varijabla za izracunatu vrijednost udaljenosti s korekcijom
    // brzina zvuka
    // varijabla za korektiranu brzinu zvuka s obzirom na temperaturu
    // varijabla za vrijednost temperature dobivene ADC sa senz. LM35
    // varijabla za pohranu vrijednosti AD pretvorbe na kanalu ADC0

    while (1)
    {
        // ako nije poslan trigger i vremenski istek nije aktivan
        if(!HCSR04_triggerSent && timeout == 0) {
            // funkcija za slanje trigger pulsa
        }

        if ()   // zastavica -> zavrseno mjerenje
        {
            // izracun proteklog vremena da se puls vrati
```

```
            pulseTime =
            // izracun udaljenosti pomocu brzine zvuka i vremena
            dist =

            valADC = adcRead(ADC0);     // ADC izlaza senzora LM35
            temp = valADC * 5.0 / 1024 * 100;   // izracun temperature
            // izracun korigirane brzine zvuka uz vrijednost temperature
            vSoundCorrected =

            // izracun korigirane udaljenosti s korigiranom vr. brzine zv.
            distCorrected =

            lcdClrScr();    // ocisti prethodni ispis na LCD
            lcdHome();      // postavi pokazivac LCD na koordinate 1,1
            // u prvi redak LCD displeja ispisi udaljenost bez korekcije
            lcdprintf("Dist: %.2fcm\n", dist);
            // u drugi redak LCD displeja ispisi udaljenost s korekcijom
            lcdprintf("DistCorr: %.2fcm", distCorrected);

            // postavi zastavicu za signaliziranje mjerenja false
            HCSR04_measured = false;
            // postavi zastavicu za signaliziranje trigger signala false
            HCSR04_triggerSent = false;
        }
        _delay_ms(150);
    }

    return 0;
}
```

U programu će se koristiti LCD displej, AD pretvornik, vanjski prekidi i tajmeri. Da bi to bilo moguće potrebno je uključiti u prevođenje biblioteke `timer.h`, `lcd.h`, `adc.h` i `interrupt.h`. Globalno su deklarirane sljedeće varijable:

- `HCSR04_triggerSent` - zastavica koja označava da li je poslan impuls za pokretanje mjerenja,
- `pulseCount` - varijabla u koju će se spremati broj taktova sklopa *Timer/Counter1* i
- `HCSR04_measured` - zastavica koja će označavati da li je završeno mjerenje pomoću senzora.

Potrebno je generirati ultrazvučni signal u obliku niza od 8 impulsa frekvencije 40 kHz, jer takav signal je vrlo lako razlučiti od smetnja u prostoru, odnosno ostalih signala sličnih frekvencija. Mikroupravljač koji se nalazi na ultrazvučnom senzoru, generirat će opisani signal na vanjsku pobudu koja se dovodi na pin `TRIG` na senzoru. Signal za okidanje niza impulsa treba biti u visokom stanju u trajanju od najmanje 10 μs. Nakon što se signal pošalje senzoru, praktično je postaviti zastavicu `HCSR04_triggerSent` u logičku istinu, kako bi što bolje kontrolirali tok programa. Definicija funkcije `HCSR04_trigger()` napisane za namjenu odašiljanja impulsnog signala kao okidača prikazana je programskim kodom 13.12. Osvježite funkciju `HCSR04_trigger()` u datoteci `vjezba133.cpp` kako bi bila istovjetna programskom kodu 13.12.

Programski kod 13.12: Definicija funkcije `HCSR04_trigger()`

```
void HCSR04_trigger() {       // fnc za trigger puls
    PORTD |= (1 << PD0);      // high state na pinu PD0
    _delay_us(20);            // koje traje 20 us
    PORTD &= ~(1 << PD0);     // low state na pinu PD0
    HCSR04_triggerSent = true;  // zastavica trigger poslan
}
```

Potrebno je na neki način pratiti da li je poslan signal reflektiran do mikrofona senzora. Najbržu reakciju na promjenu signala uvijek ima vanjski prekid kao sklop mikroupravljača. Koristit će se vanjski prekid `INT1` koji je spojen na pin `PD3` mikroupravljača. Izlazni pin senzora `ECHO` spojen je na pin `PD3` mikroupravljača. Ukoliko se na pinu `PD3` pojavi padajući ili rastući brid signala, tada će se pozvati prekidna rutina `ISR(INT1_vect)`. Kada se dogodi poziv te prekidne rutine, provjerava se da li je stanje ulaznog pina `PD3` i dalje visoko te u tom slučaju treba zadržati vrijednost registra za brojenje `TCNT1` sklopa *Timer/Counter1* na vrijednosti 0. Onog trenutka kada se dogodi prvi padajući brid signala na `ECHO` pinu (pin `PD3` mikroupravljača), tada se pohrani broj izbrojenih taktova iz registra `TCNT1` u varijablu `pulseCount`. Na kraju prekidne rutine, zastavica `HCSR04_measured` se postavlja na logičku istinu kako bi signalizirala ostatku programa da je mjerenje završeno od strane senzora. Definicija prekidne rutine `ISR(INT1_vect)` prikazana je programskim kodom 13.13. Osvježite prekidnu rutinu `ISR(INT1_vect)` u datoteci `vjezba133.cpp` kako bi bila istovjetna programskom kodu 13.13.

Programski kod 13.13: Definicija prekidne rutine `ISR(INT1_vect)`

```
ISR(INT1_vect) {
    if(get_pin(PIND,PD3) == 1) {    // ECHO pin -> idle state
        TCNT1 = 0;      // timer counter registar -> nula
    }
    else {
        pulseCount = TCNT1;      // broj pulseva -> pulseCount
        HCSR04_measured = true; // zastavicom mjerenje zavrseno
    }
}
```

U inicijalizacijskoj funkciji `init()` potrebno je inicijalizirati periferiju i mikroupravljač. Pomoću funkcije `adcInit()` inicijalizirana je AD pretvorba koja će se koristiti za pretvorbu izlaznog signala temperaturnog senzora LM35 u koristan podatak. Senzor se nalazi na razvojnom okruženju i objašnjen je detaljnije u poglavlju `Analogno-digitalna pretvorba`. Nakon toga inicijaliziran je LCD displej. Da bi prekidna rutina za praćenje dolaznog signala na `ECHO` pinu senzora (pin `PD3` mikroupravljača) bila u funkciji, potrebno je istu omogućiti funkcijom `int1Enable()`. Potrebno je postaviti i vrstu signala na koji će se pozvati prekidna rutina. U ovom slučaju prekidna rutina će se pozvati na padajući ili rastući brid signala spojenog na pin `PD3` mikroupravljača i to je definirano funkcijom `int1RisingFallingEdge()`. Potreban je sklop *Timer/Counter1* kako bismo točno brojili proteklo vrijeme od trenutka poslanog signala za okidač do trenutka kada se vratio signal reflektiran od neke prepreke. Za tu namjenu dovoljno je sklop *Timer/Counter1* postaviti u normalan način rada s djeliteljem frenvencije iznosa 8. Potrebno je definirati ulazne i izlazne pinove (`ECHO` i `TRIG`, `PD3` i `PD0`). Nije potrebno uključiti pritezni otpornik za pin na mikroupravljaču gdje je spojen `ECHO` pin jer je ECHO izlaz HC-SR04 senzora tipa *Push-Pull*. Potpuna definicija inicijalizacijske funkcije `init()` prikazana je programskim kodom 13.14. Osvježite funkciju `init()` u datoteci `vjezba133.cpp` kako bi bila istovjetna programskom kodu 13.14.

Programski kod 13.14: Definicija inicijalizacijske funkcije `init()`

```
void init(){         // inicijalizacija mikroupravljaca
    lcdInit();       // inicijalizacija LCD displeja
    adcInit();       // inicijalizacija AD pretvornika

    lcdClrScr();     // ocisti prethodni zapis na LCD
    lcdHome();       // pokazivac LCD na koordinate 1,1

    // prekidna rutina 1 rastuci i padajuci brid signala ECHO
    int1RisingFallingEdge();
    int1Enable();       // omoguci prekidnu rutinu 1

    timer1NormalMode();     // timer 1 -> normalni nacin rada
```

```
    // djelitelj frekvencije za timer 1 ce biti 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);

    config_output(DDRD,PD0);    // PD0 pin -> izlaz (TRIG)
    config_input(DDRD,PD3);     // PD3 pin -> ulaz (ECHO)

    interruptEnable();      // globalno omogucenje rutina
}
```

U glavnoj programskoj funkciji `main()` pozvana je inicijalizacijska funkcija `init()`. Nakon toga, deklarirane su sljedeće pomoćne varijable.

- `timeout` - varijabla za definiranje vremenskog isteka signala na pinu `ECHO`,
- `pulseTime` - varijabla za pohranu proteklog vremena do vraćanja signala,
- `distance` - izračunata vrijednost udaljenosti bez korekcije,
- `distanceCorrected` - izračunata vrijednost udaljenosti s korekcijom,
- `vSound` - brzina zvuka bez korekcije,
- `vSoundCorrected` - brzina zvuka s korekcijom,
- `temp` - temperatura okoline dobivena izračunom rezultata AD pretvorbe i
- `valADC` - varijabla za pohranu rezultata AD pretvorbe (digitalna riječ).

Definirane varijable s pripadajućim tipovima podataka prikazane su na programskom kodu 13.15. Navedene varijable upišite u datoteku `vjezba133.cpp` na početku funkcije `main()`.

Programski kod 13.15: Pomoćne varijable s pripadajućim tipovima podataka

```
// varijabla za vremenski istek zahtjeva za trigger puls
uint16_t timeout = 0;
// varijabla za pohranu proteklog vremena koje se mjeri timerom
float pulseTime = 0.0;
// varijabla za izracunatu vrijednost udaljenosti bez korekcije
float dist = 0.0;
// varijabla za izracunatu vrijednost udaljenosti s korekcijom
float distCorrected = 0.0;
// brzina zvuka
float vSound = 343.0;
// varijabla za korektiranu brzinu zvuka s obzirom na temperaturu
float vSoundCorrected;
// varijabla za vrijednost temperature dobivene ADC sa senz. LM35
float temp = 0.0;
// varijabla za pohranu vrijednosti AD pretvorbe na kanalu ADC0
float valADC = 0;
```

U beskonačnoj petlji `while()` provjerava se da li je poslan signal za generiranje niza impulsa te ako nije, tada se poziva funkcija `HCSR04_trigger()` koja to učini. U svakoj iteraciji programske petlje `while()` provjerava se da li je zastavica `HCSR04_measured` u logičkoj istini. Kada je zastavica u logičkoj istini, mjerenje je završeno i podaci su spremni za daljnju obradu i prikaz. Trajanje jednog pulsa sklopa *Timer/Counter1* moguće je izračunati pomoću sljedeće relacije 13.2.

$$t_{puls} = \frac{prescaler}{F_CPU} \tag{13.2}$$

Potrebno je izvesti relaciju koja će točno odrediti vrijeme potrebno za povratak signala na `ECHO` pin senzora. Ukoliko je vrijeme trajanja jednog pulsa moguće izvesti relacijom 13.2, tada tu vrijednost pomnožimo s brojem pulsa spremljenim u varijablu `pulseCount` i moguće je izračunati vrijeme koje je potrebno da ultrazvučni val ode do prepreke i vrati se nazad do senzora pomoću relacije 13.3.

$$t_{ECHO} = pulseTime = pulseCount \cdot t_{puls} = pulseCount \cdot \frac{prescaler}{F_CPU} \tag{13.3}$$

Poznata je relacijska ovisnost puta o vremenu i brzini koja glasi da je brzina jednaka prijeđenom putu u nekom određenom vremenu. Ukoliko je poznata brzina (u ovom slučaju brzina ultrazvučnog signala) i poznato je prethodno izračunato vrijeme na pinu `ECHO`, jednostavnom matematičkom relacijom može se izračunati udaljenost od prepreke. Dogovorni iznos brzine zvuka bez korekcije s obzirom na sobnu temperaturu iznosi približno 343 m/s. Zapis te relacije glasi 13.4.

$$dist = \frac{pulseTime}{2} \cdot vSound \cdot 100.0 \tag{13.4}$$

Vrijeme koje je potrebno da ultrazvučni val ode do prepreke i vrati se nazad do senzora ($pulseTime$) dijeli se s 2 jer ultrazvučni val prelazi dvostruku udaljenost od senzora do prepreke. S obzirom da nije ista brzina širenja ultrazvučnog vala u prostorima različite okolišne temperature, relacijom 13.5 prikazana je ovisnost brzine zvuka o temperaturi:

$$vSoundCorrected = 331.0 + 0.6 \cdot temp \tag{13.5}$$

Potrebno je izračunati temperaturu okoline. Razvojno okruženje sadrži LM35 senzor temperature koji ima analogni izlaz. Potrebno je izračunati temperaturu pomoću rezultata AD pretvorbe kao što je to prikazano programskim kodom 13.16, gdje je prikazan i izračun brzine zvuka s korekcijom s obzirom na izmjerenu temperaturu.

Programski kod 13.16: Izračun temperature pomoću AD pretvorbe izlaza senzora LM35

```
valADC = adcRead(ADC0);      // ADC izlaza senzora LM35
temp = valADC * 5.0 / 1024 * 100;    // izracun temperature
// izracun korigirane brzine zvuka uz vrijednost temperature
vSoundCorrected = 331.0 + 0.6 * temp;
```

Nakon što se izračuna korigirana vrijednost brzine zvuka s obzirom na temperaturu, moguće je izračunati i korigiranu vrijednost udaljenosti prema relaciji 13.6:

$$distCorrected = \frac{pulseTime}{2} \cdot vSoundCorrected \cdot 100.0 \tag{13.6}$$

Potrebno je prikazati izračunate udaljenosti sa i bez korekcije na LCD displeju. U prvom retku prikažite poruku npr. `Dist: 15.23 cm` koja prikazuje udaljenost bez korekcije brzine zvuka s obzirom na temperaturu, dok u drugom retku LCD displeja možete prikazati npr. `DistCorr: 14.98 cm` poruku koja prikazuje udaljenost s korekcijom brzine zvuka s obzirom na temperaturu. Nakon izračuna i prikaza potrebno je postaviti zastavice `HCSR04_measured` i `HCSR04_triggerSent` u logičku laž, kako bi sljedeća iteracija mjerenja i izračuna imala potrebne logičke početne uvjete. Potpuna definicija glavne beskonačne programske petlje `while()` prikazana je programskim kodom 13.17. Osvježite programsku petlju `while()` u datoteci `vjezba133.cpp` kako bi bila istovjetna programskom kodu 13.17.

Programski kod 13.17: Potpuna definicija programske petlje `while()`

```
while (1)
{
    // ako nije poslan trigger i vremenski istek nije aktivan
    if(!HCSR04_triggerSent && timeout == 0) {
        HCSR04_trigger();   // funkcija za slanje trigger pulsa
    }

    if (HCSR04_measured)    // zastavica -> zavrseno mjerenje
    {
        // izracun proteklog vremena da se puls vrati
        pulseTime = pulseCount * 8.0 / F_CPU;
        // izracun udaljenosti pomocu brzine zvuka i vremena
        dist = pulseTime / 2.0 * vSound * 100.0;

        valADC = adcRead(ADC0);     // ADC izlaza senzora LM35
        temp = valADC * 5.0 / 1024 * 100;   // izracun temperature
        // izracun korigirane brzine zvuka uz vrijednost temperature
        vSoundCorrected = 331.0 + 0.6 * temp;

        // izracun korigirane udaljenosti s korigiranom vr. brzine zv.
        distCorrected = pulseTime / 2.0 * vSoundCorrected * 100.0;

        lcdClrScr();    // ocisti prethodni ispis na LCD
        lcdHome();      // postavi pokazivac LCD na koordinate 1,1
        // u prvi redak LCD displeja ispisi udaljenost bez korekcije
        lcdprintf("Dist: %.2fcm\n", dist);
        // u drugi redak LCD displeja ispisi udaljenost s korekcijom
        lcdprintf("DistCorr: %.2fcm", distCorrected);

        // postavi zastavicu za signaliziranje mjerenja na false
        HCSR04_measured = false;
        // postavi zastavicu za signaliziranje poslanog trigger signala na false
        HCSR04_triggerSent = false;
    }
    _delay_ms(150);
}
```

Ako ste slijedili navedene korake vaša će datoteka `vjezba133.cpp` biti istovjetna programskom kodu 13.18.

Programski kod 13.18: Potpuni sadržaj datoteke `vjezba133.cpp`

```
#include <avr/io.h>
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include "Interrupt/interrupt.h"
#include <util/delay.h>

// zastavica za signalizaciju da je trigger puls poslan
bool HCSR04_triggerSent = false;
// pomocna varijabla za spremanje brojenja pulseva
volatile uint32_t pulseCount = 0;
// pomocna zastavica za signalizaciju zavrsenog mjerenja
volatile bool HCSR04_measured = false;

ISR(INT1_vect) {
    if(get_pin(PIND,PD3) == 1) {    // ECHO pin -> idle state
        TCNT1 = 0;      // timer counter registar -> nula
    }
    else {
```

```
        pulseCount = TCNT1;     // broj pulseva -> pulseCount
        HCSR04_measured = true; // zastavicom mjerenje zavrseno
    }
}

void HCSR04_trigger() {     // fnc za trigger puls
    PORTD |= (1 << PD0);    // high state na pinu PD0
    _delay_us(20);          // koje traje 20 us
    PORTD &= ~(1 << PD0);   // low state na pinu PD0
    HCSR04_triggerSent = true;  // zastavica trigger poslan
}

void init(){         // inicijalizacija mikroupravljaca
    lcdInit();       // inicijalizacija LCD displeja
    adcInit();       // inicijalizacija AD pretvornika

    lcdClrScr();     // ocisti prethodni zapis na LCD
    lcdHome();       // pokazivac LCD na koordinate 1,1

    // prekidna rutina 1 rastuci i padajuci brid signala ECHO
    int1RisingFallingEdge();
    int1Enable();        // omoguci prekidnu rutinu 1

    timer1NormalMode();     // timer 1 -> normalni nacin rada
    // djelitelj frekvencije za timer 1 ce biti 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);

    config_output(DDRD,PD0);    // PD0 pin -> izlaz (TRIG)
    config_input(DDRD,PD3);     // PD3 pin -> ulaz (ECHO)

    interruptEnable();      // globalno omogucenje rutina
}

int main(void){

    init();     // poziv fnc inicijalizacija mikroupravljaca

    // varijabla za vremenski istek zahtjeva za trigger puls
    uint16_t timeout = 0;
    // varijabla za pohranu proteklog vremena koje se mjeri timerom
    float pulseTime = 0.0;
    // varijabla za izracunatu vrijednost udaljenosti bez korekcije
    float dist = 0.0;
    // varijabla za izracunatu vrijednost udaljenosti s korekcijom
    float distCorrected = 0.0;
    // brzina zvuka
    float vSound = 343.0;
    // varijabla za korektiranu brzinu zvuka s obzirom na temperaturu
    float vSoundCorrected;
    // varijabla za vrijednost temperature dobivene ADC sa senz. LM35
    float temp = 0.0;
    // varijabla za pohranu vrijednosti AD pretvorbe na kanalu ADC0
    float valADC = 0;

    while (1)
    {
        // ako nije poslan trigger i vremenski istek nije aktivan
        if(!HCSR04_triggerSent && timeout == 0) {
            HCSR04_trigger();   // funkcija za slanje trigger pulsa
        }

        if (HCSR04_measured)    // zastavica -> zavrseno mjerenje
        {
```

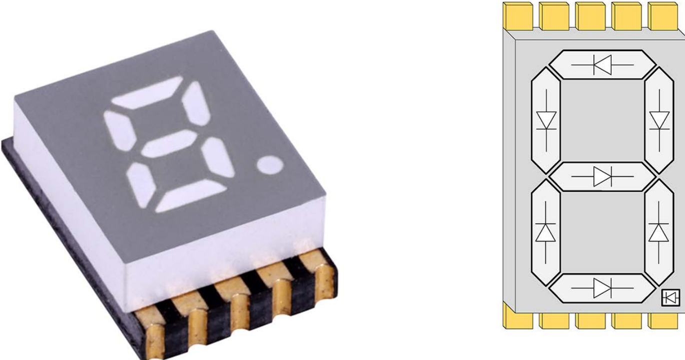
```
            // izracun proteklog vremena da se puls vrati
            pulseTime = pulseCount * 8.0 / F_CPU;
            // izracun udaljenosti pomocu brzine zvuka i vremena
            dist = pulseTime / 2.0 * vSound * 100.0;

            valADC = adcRead(ADC0);     // ADC izlaza senzora LM35
            temp = valADC * 5.0 / 1024 * 100;   // izracun temperature
            // izracun korigirane brzine zvuka uz vrijednost temperature
            vSoundCorrected = 331.0 + 0.6 * temp;

            // izracun korigirane udaljenosti s korigiranom vr. brzine zv.
            distCorrected = pulseTime / 2.0 * vSoundCorrected * 100.0;

            lcdClrScr();    // ocisti prethodni ispis na LCD
            lcdHome();      // postavi pokazivac LCD na koordinate 1,1
            // u prvi redak LCD displeja ispisi udaljenost bez korekcije
            lcdprintf("Dist: %.2fcm\n", dist);
            // u drugi redak LCD displeja ispisi udaljenost s korekcijom
            lcdprintf("DistCorr: %.2fcm", distCorrected);

            // postavi zastavicu za signaliziranje mjerenja na false
            HCSR04_measured = false;
            // postavi zastavicu za signaliziranje poslanog trigger signala na false
            HCSR04_triggerSent = false;
        }
        _delay_ms(150);
    }

    return 0;
}
```

Prevedite datoteku vjezba133.cpp u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Uz pomoć ravnala ili nekog drugog kalibriranog mjerila testirajte točnost mjerenja udaljenosti ultrazvučnim senzorom HC-SR04. Komentirajte utjecaj korekcije brzine zvuka na krajnji rezultat. Testirajte mjerni opseg senzora udaljenosti. Nakon što ste testirali vježbu, zatvorite datoteku vjezba133.cpp i onemogućite prevođenje ove datoteke.

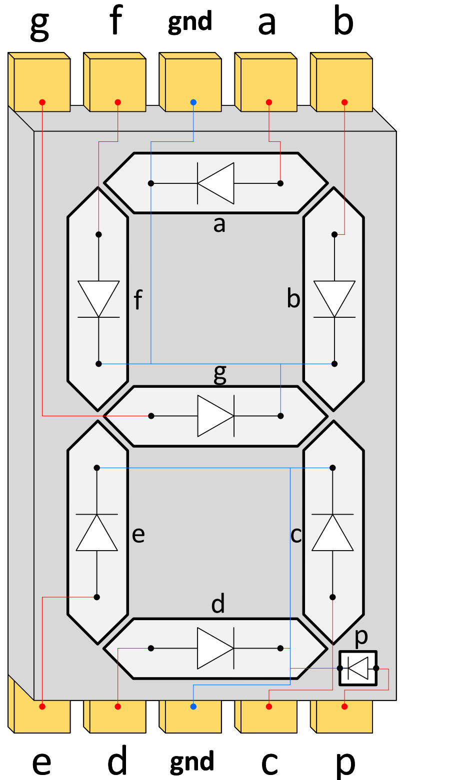
13.4 Numerički displej i posmačni registri

Numerički displej ili segmentni displej sastoji se od 7 LED dioda. Zbog toga se često još naziva i 7-segmentni displej. Diode su postavljene u strukturu pomoću koje je moguće vizualizirati sve znamenke dekadskog ili heksadekadskog sustava. Svaka LED dioda je nazvana segment jer kada svijetli čini segmentni dio neke znamenke. Da bi se prikazivale brojeve veće od 10, potrebno je koristiti više displeja. Za prikaz realnih brojeva u upotrebu ulazi i dodatna osma LED dioda koja ima značenje decimalne točke. Numerički displej pripada kategoriji opto-elektroničkih komponenti i obično se nalazi u plastičnom kućištu pravokutnog oblika. Na slici 13.10 prikazana je uobičajena komponenta numeričkog displeja i 3D model iste.

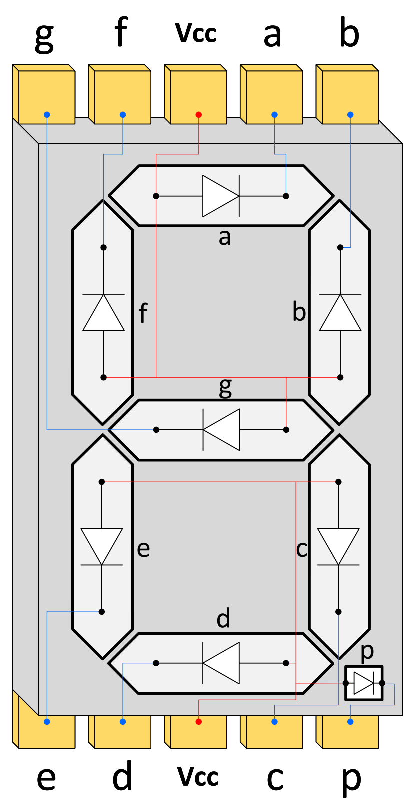
Numerički displej sastoji se od LED dioda koje kao i svaka dioda, imaju svoje priključnice: anodu i katodu. Zbog niske vrijednosti najveće moguće struje koja može proći kroz strujni krug u kojem je pin mikroupravljača, potrebno je dodati izvana otpornik u seriju sa svakim segmentom displeja kako bi se ta struja ograničila. Ušteda izlaznih pinova na numeričkom displeju ostvarena je zajedničkim spajanjem svih anoda ili katoda. Prema toj karakteristici razlikujemo dvije izvedbe numeričkih displeja koje su prikazane na slici 13.11.



Slika 13.10: Numerički sedam segmentni displej i 3D model



(a) Numerički displej u spoju zajedničke katode



(b) Numerički displej u spoju zajedničke anode

Slika 13.11: Izvedbe numeričkog 7-segmentnog displeja

Na izvedbama su plavom bojom označene linije koje su na niskoj razini signal (0 V). Crvenom linijom su označene linije visoke razine signala (5 V). Kada se usporede izvedbe na slici, moguće je zaključiti:

- **Spoj zajedničke katode:** sve katode su spojene na nisku razinu signala (gnd), dok će se segmenti uključivati preko pinova priključenjem na visoku razinu signala (5 V) i
- **Spoj zajedničke anode:** sve anode su spojene na visoku razinu signala (Vcc), dok će se segmenti uključivati preko pinova priključenjem na nisku razinu signala (0 V).

Svaki segment označen je odgovarajućim slovom engleske abecede koje odgovara istim redoslijedom binarnoj kombinaciji pojedinog znaka dekadskog sustava za koji su kombinacije prikazane tablicom stanja 13.1. Vizualni prikaz znamenaka dekadskog sustava na

komponenatama numeričkog segmentnog displeja prikazan je na slici 13.12:

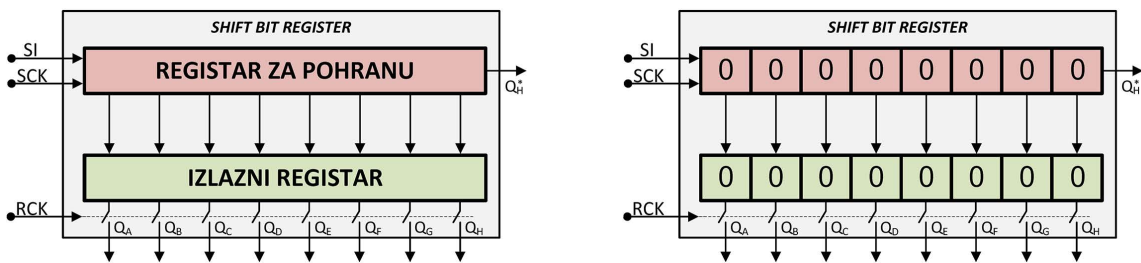
Slika 13.12: Vizualni prikaz znamenaka dekadskog sustava na numeričkom displeju

Tablica 13.1: Stanja pojedinih segmenata za prikaz znamenki dekadskog sustava

Znamenka	Segmenti numeričkog displeja							
	a	b	c	d	e	f	g	p
0	uključen	uključen	uključen	uključen	uključen	uključen	isključen	x
1	isključen	uključen	uključen	isključen	isključen	isključen	isključen	x
2	uključen	uključen	isključen	uključen	uključen	isključen	uključen	x
3	uključen	uključen	uključen	uključen	isključen	isključen	uključen	x
4	isključen	uključen	uključen	isključen	isključen	uključen	uključen	x
5	uključen	isključen	uključen	uključen	isključen	uključen	uključen	x
6	uključen	isključen	uključen	uključen	uključen	uključen	uključen	x
7	uključen	uključen	uključen	isključen	isključen	isključen	isključen	x
8	uključen	uključen	uključen	uključen	uključen	uključen	uključen	x
9	uključen	uključen	uključen	uključen	isključen	uključen	uključen	x

Za upravljanje numeričkim displejima koriste se mikroupravljači. Ukoliko pogledamo numerički displej na slici 13.10, možemo vidjeti da je potrebno 7 izlaznih pinova mikroupravljača za kontrolu svih segmenata numeričkog displeja. Uobičajeno je spajanje više numeričkih displeja u ugradbenom sustavu. Ukoliko na razvojnom okruženju imamo 4 numerička displeja, od kojih svakih koristi 7 pinova mikroupravljača, ukupni utrošak pinova za numeričke displeje je 28. S obzirom na resurse korištenog mikroupravljača ATmega328P na razvojnom okruženju, nije moguće koristiti 4 numerička displeja zbog pomanjkanja broja pinova. Općenito je to značajan broj pinova za mikroupravljače jer obično u ugradbenom sustavu postoje još dodatno senzori, aktuatori, komunikacijski moduli i slično pa tako visok broj pinova samo za numerički displej

je u većini slučajeva prevelik utrošak resursa. Postoji rješenje za ovaj česti problem, drugim riječima postoji rješenja potrebe za povećanjem broja izlaznih pinova mikroupravljača. U tu namjenu koriste se elektroničke komponente nazvane - posmačni registri (engl. *Shift Bit Register; SBR*) širine 8 bitova. Što se tiče utroška pinova mikroupravljača, za jedan numerički displej je potrebno 3 pina mikroupravljača umjesto 7. Ukoliko se gleda na grupu od 4 numerička displeja, tada je utrošak pinova 3 u usporedbi s 28 ako se ne koriste posmačni registri (bez računanja pina za točku). Na razvojnom okruženju korištenom u ovoj knjizi, odabrani su posmačni registri kataloškog broja 74HC595D čija je blokovska shema koja prikazuje princip rada prikazana na slici 13.13.



Slika 13.13: Blokovska shema posmačnog registra 74HC595D

Posmačni registri rade na principu serijskog ulaza na koji se dovode binarne kombinacije (podatak - binarna reprezentacija istog). Na vanjski takt iste se upisuju u registar za pohranu. Nakon upisivanja binarnog podatka, na vanjski takt, vrijednosti zapisane u registru preslikavaju se na fizičke izlaz, odnosno pinove posmačnog registra. Ako pogledamo blokovnu shemu na slici 13.13 možemo vidjeti sljedeće elemente posmačnog registra:

- `SI` (engl. *Serial Input*) - ulaz u posmačni registar koji se koristi za dovođenje visokog ili niskog stanja signala, odnosno binarne kombinacije koja predstavlja podatak koji se želi zapisati u posmačni registar,

- `SCK` (engl. *Serial Clock*) - ulaz u posmačni registar na koji se dovodi vanjski signal koji predstavlja takt za zapisivanje trenutnog podatka na `SI` ulazu,

- `RCK` (engl. *Register Clock*) - ulaz u posmačni registar na koji se također dovodi takt, na koji će posmačni registar preslikati vrijednosti iz `Registar za pohranu` u `izlazni registar`,

- $Q_A - Q_H$ - fizički izlazni pinovi posmačnog registra izravno spojeni na `Izlazni registar`,

- Q_H^* - nakon 8 taktova dovedenih na ulaz `SCK`, na ovaj izlaz prosljeđuje se zadnji bit poruke Q_H^*,

- `Registar za pohranu` - registar širine 8 bitova u koji se pohranjuje podatak doveden na `SI` ulaz na taktove dovedene na ulaz `SCK` i

- `Izlazni registar` - registar širine 8 bitova na koji su izravno priključeni fizički pinovi. Sadržaj registra se preslikava iz `Registar za pohranu` taktom dovedenim na `RCK` ulaz.

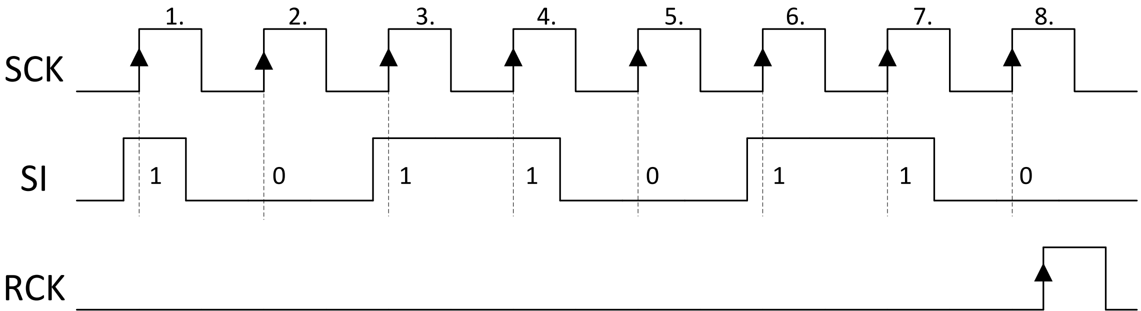
Slijed punjenja posmačnog registra širine 8 bitova prikazan je na slici 13.14 i može se opisati u nekoliko koraka:

- odabere se neki podatak širine 8 bitova, na primjer znamenka 5 iz dekadskog brojevnog sustava koja odgovara binarnoj kombinaciji za numerički displej 10110110,

- dovodi se prvi bit najmanje važnosti (engl. *Least Significant Bit; LSB*) na `SI` ulaz

posmačnog registra,

- dovodi se kratak impuls (takt) na `SCK` ulaz posmačnog registra kako bi se LSB binarne kombinacije upisao u `Registar za pohranu` posmačnog registra,
- binarna kombinacija se bitovno pomakne u desno za jedno mjesto, kako bi sada sljedeći bit bio na mjestu LSB,
- sada se taj bit dovodi na `SI` ulaz posmačnog registra i dovodi se vanjski takt na `SCK` ulaz posmačnog registra kako bi se taj bit upisao u `Registar za pohranu`,
- prethodni koraci se ponavljaju sve dok nije upisano svih 8 bitova binarne kombinacije 10110110 te tada
- dovodi se vanjski takt (impuls) na `RCK` ulaz posmačnog registra, kako bi se pohranjena binarna kombinacija iz `Registar za pohranu` preslikala u `Izlazni registar`.

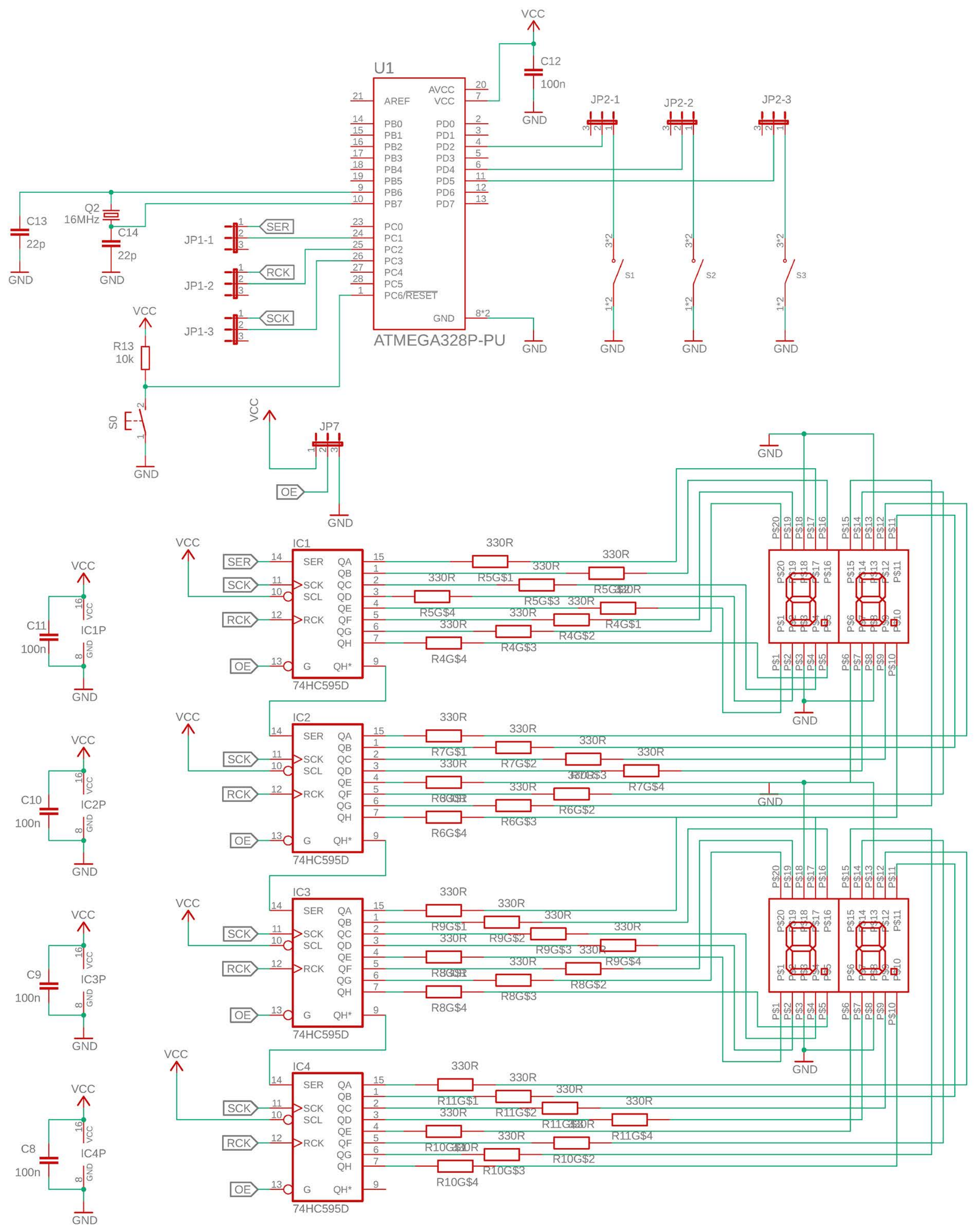


Slika 13.14: Slijed punjenja posmačnog registra širine 8 bitova

Kao što je ranije rečeno, moguće je spojiti više numeričkih displeja, jedan do drugog, kako bi se mogli prikazivati brojevi s više znamenaka. Kako bismo smanjili utrošak pinova, posmačne registre je moguće spojiti u seriju na sljedeći način:

- `SI` ulaz 1. posmačnog registra potrebno je spojiti na pin mikroupravljača namijenjen za ulogu generiranja podatka za pohranu u `Registar za pohranu`,
- Q_H^* izlaz 1. posmačnog registra spojiti na ulaz `SI` 2. posmačnog registra,
- Q_H^* izlaz 2. posmačnog registra spojiti na ulaz `SI` 3. posmačnog registra,
- Q_H^* izlaz 3. posmačnog registra spojiti na ulaz `SI` 4. posmačnog registra,
- sve `SCK` ulaze posmačnih registara potrebno je spojiti na pin mikroupravljača namijenjen za ulogu generiranja serijskog takta za upisivanje vrijednosti i
- sve `RCK` ulaze posmačnih registara potrebno je spojiti na pin mikroupravljača namijenjen za ulogu generiranja takta za preslikavanje podataka iz `Registar za pohranu` u `Izlazni registar`.
- Pinove posmačnih registara, odnosno spojne točke `SI`, `SCK` i `RCK` spojiti na pinove mikroupravljača.

Shema spajanja 4 numerička displeja preko 4 posmačna registra 74HC595D s razvojnim okruženjem upravljanim mikroupravljačem ATmega328P prikazana je na slici 13.15:



Slika 13.15: Shema spajanja 4 numerička displeja i posmačna registra 74HC595D s mikroupravljačem ATmega328P

Vježba 13.4

Potrebno je napisati program koji će upravljati numeričkim displejom preko posmačnih registara ispisujući na sva 4 numerička displeja znamenku 5 dekadskog sustava. U proširenju

vježbe, potrebno je ispisati znamenke 5, 6, 7 i 8 istim redoslijedom na sva 4 displeja. U drugom proširenju vježbe, potrebno je ispisati vrijednost napona i digitalne riječi dobivene AD pretvorbom kanala `ADC0` na koji je spojen potenciometar na numeričke displeje. Postupno proširite kod prema zadanim ispisima. Počnite sa ispisom sve 4 iste znamenke (broj 5). Shema spajanja 4 numerička displeja preko 4 posmačna registra s razvojnim okruženjem upravljanim mikroupravljačem ATmega328P prikazana je na slici 13.15.

U projektnom stablu otvorite datoteku `vjezba134a.cpp`. Omogućite samo prevođenje datoteke `vjezba134a.cpp`. Početni sadržaj datoteke `vjezba134a.cpp` prikazan je programskim kodom 13.19.

Programski kod 13.19: Početni sadržaj datoteke `vjezba134a.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"

void init() {
    // inic. serijskog ulaza (SI) SBR
    // inic. register clock ulaza (RCK) SBR
    // inic. serial clock ulaza (SCK) SBR
}

int main(void) {

    init();

    uint8_t data = 0xB6;    // testni podatak

    // 4x ispis -> ispis na sva 4 displeja
    for (uint8_t i = 0; i < 4; i++)
    {
        // provjera svih 8 bitova podatka
        for (uint8_t j = 0; j < 8; j++)
        {
            if (data & 0x01)    // maskiranje LSB
            {
                // SI - true
            }
            else
            {
                // SI - false
            }

            data = data >> 1;   // posmak za 1 >>

            // SCK puls - 1 us

        }

        data = 0xFF;    // reinicijalizacija podatka

        // RCK puls - 1us

    }
}
```

Sva četiri numerička displeja spojena su preko posmačnih registara na mikroupravljač. Iste je moguće upravljati sa samo 3 pina mikroupravljača koje je potrebno inicijalizirati. Funkcije pinova su sljedeće:

- pin PC1 - serijski ulaz SI u posmačni registar,
- pin PC2 - takt RCK za upis u Izlazni registar i
- pin PC3 - takt SCK za upis u Registar za pohranu.

Inicijalizacijska funkcija init() prikazana je programskim kodom 13.20. Osvježite funkciju init() u datoteci vjezba134a.cpp kako bi bila istovjetna programskom kodu 13.20.

Programski kod 13.20: Inicijalizacijska funkcija init()

```
void init() {
    // inic. serijskog ulaza (SI) SBR
    DDRC |= (1 << PC1);
    // inic. register clock ulaza (RCK) SBR
    DDRC |= (1 << PC2);
    // inic. serial clock ulaza (SCK) SBR
    DDRC |= (1 << PC3);
}
```

U glavnoj programskoj funkciji main() pozvana je inicijalizacijska funkcija init(). Sljedeće je inicijaliziran podatak koji će se koristiti za ispis na sva 4 numerička displeja. U ovoj vježbi izabrana je dekadska znamenka 5. Sve kombinacije segmenata za prikaz dekadskih znamenaka i pripadnih heksadekadskih vrijednosti prikazane su tablicom 13.2.

Tablica 13.2: Kombinacije segmenata numeričkog displeja

Znamenka	Segmenti								HEX
	a	b	c	d	e	f	g	p	
0	1	1	1	1	1	1	0	0	0xFC
1	0	1	1	0	0	0	0	0	0x60
2	1	1	0	1	1	0	1	0	0xDA
3	1	1	1	1	0	0	1	0	0xF2
4	0	1	1	0	0	1	1	0	0x66
5	1	0	1	1	0	1	1	0	0xB6
6	1	0	1	1	1	1	1	0	0xBE
7	1	1	1	0	0	0	0	0	0xE0
8	1	1	1	1	1	1	1	0	0xFE
9	1	1	1	1	0	1	1	0	0xF6

Prema načinu rada posmačnog registra, može se zaključiti da generiranjem 8 taktova na SCK ulaz posmačnog registra, upisati će se 8 bitova podatka dovedenog na SI ulaz. Podatak se u programskom kodu pomiče u desno metodom posmaka bitova (engl. *Bit Shifting*), gdje dolazi do preljeva *LSB* bita i tada taj bit postaje trajno izgubljen. U vježbi je zadano da isti broj treba ispisati na sva 4 numerička displeja. Na ovaj način ispisao bi se samo prvi displej, dok na svim drugima nebi svijetlio niti jedan segment, jer ne postoji više podatak zbog preljeva (stalno nisko stanje signala na SI ulazu). Zbog ovog logičkog problema, uvedene su dvije ugniježđene *for* petlje (engl. *Nested For Loops*). Nakon prvih 8 bitova, podatak se reinicijalizira na početnu vrijednost, kako bi iduća iteracija slanja 8 bitova podatka imala dostupne sve bitova podatka kao i prethodna. Prilikom svakog upisivanja podatka dovedenog na SI ulaz, potrebno je dovesti kratki impuls na SCK ulaz posmačnog registra kako bi se dovedeni podatak (trenutni *LSB* bit podatka), upisao u Registar za pohranu. Nakon upisanih svih 4 podatka na sva 4 posmačna registra, iste je

potrebno prepisati u `Izlazni registar`, kako bi podaci postali vidljivi na numeričim displejima (uključeni/isključeni segmenti numeričkog displeja). Preslikavanje podataka (bitova) iz `Registar za pohranu` u `Izlazni registar` zadaje se impulsom na ulaz `RCK` posmačnih registara. Potpuni sadržaj datoteke `vjezba134a.cpp` prikazan je programskim kodom 13.21.

Programski kod 13.21: Potpuni sadržaj datoteke `vjezba134a.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"

void init() {
    // inic. serijskog ulaza (SI) SBR
    DDRC |= (1 << PC1);
    // inic. register clock ulaza (RCK) SBR
    DDRC |= (1 << PC2);
    // inic. serial clock ulaza (SCK) SBR
    DDRC |= (1 << PC3);
}

int main(void) {

    init();

    uint8_t data = 0xB6;    // testni podatak

    // 4x ispis -> ispis na sva 4 displeja
    for (uint8_t i = 0; i < 4; i++)
    {
        // provjera svih 8 bitova podatka
        for (uint8_t j = 0; j < 8; j++)
        {
            if (data & 0x01)        // maskiranje LSB
            {
                PORTC |= (1 << PC1);    // SI - true
            }
            else
            {
                PORTC &= ~(1 << PC1);   // SI - false
            }

            data = data >> 1;       // posmak za 1 >>

            PORTC |= (1 << PC3);    // SCK puls - 1 us
            _delay_us(1);
            PORTC &= ~(1 << PC3);
            _delay_us(1);
        }

        data = 0xB6;    // reinicijalizacija podatka

        PORTC |= (1 << PC2);    // RCK puls - 1us
        _delay_us(1);
        PORTC &= ~(1 << PC2);
        _delay_us(1);
    }
}
```

Prevedite datoteku `vjezba134a.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Na sva 4 numerička displeja trebao bi biti ispisan broj 5. Uz pomoć slike 13.12 i tablice 13.1 definirajte vlastiti znak na numeričkom displeju te isti ispišite na sva 4 numerička displeja. Nakon što ste testirali vježbu, zatvorite datoteku `vjezba134a.cpp` i onemogućite prevođenje ove

datoteke.

Potrebno je proširiti programski kod iz prethodne vježbe na način da će sada na numeričkim displejima biti ispisane znamenke 5, 6, 7 i 8 dekadskog sustava. Potrebno je kreirati novu datoteku `vjezba134b.cpp` i omogućiti prevođenje iste. Početni sadržaj datoteke `vjezba134b.cpp` koji je potrebno napisati prikazan je programskim kodom 13.22:

Programski kod 13.22: Početni sadržaj datoteke `vjezba134b.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"
#include "Peripheral/SBR_7seg.h"

void init() {
    // inicijalizacija SBR/7seg.
}

int main(void) {
    // inic. mikroupravljaca

    // posalji 4B podataka odjednom
    // na sva cetiri 7-seg. displeja
}
```

Ukoliko pažljivo promotrimo zaglavlje programskog koda 13.22 možemo vidjeti da je uključena u prevođenje jedna nova biblioteka nazvana `SBR_7seg.h`. Kako bi se pojednostavilo inicijaliziranje posmačnih registara definirani su konstantama registar `DDR` i registar `PORT` te pin. Konstante se definiraju na jednom mjestu uz nove varijable korištene u biblioteci. Svrha je ta da kod svake reinicijalizacije pinova ne treba mijenjati višestruke linije programskog koda na različitim mjestima u biblioteci. Tako definirani registri i pinovi prikazani su programskim kodom 13.23:

Programski kod 13.23: Definirani registri i pinovi posmačnih registara

```
#define SI_DDR DDRC
#define RCK_DDR DDRC
#define SCK_DDR DDRC

#define SI_PORT PORTC
#define RCK_PORT PORTC
#define SCK_PORT PORTC

#define SI_PIN PC1
#define RCK_PIN PC2
#define SCK_PIN PC3
```

Ranije korištena tablica u ovoj vježbi prikazuje da nije izravno i jednostavno očitati heksadekadsku kombinaciju nekog dekadskog broja koji se prikazuje na numeričkom displeju. Stoga, uvedene su konstante za sve kombinacije segmenata dekadskog sustava, uključujući segment `p` koji označava točku na numeričkom displeju. Znamenke dekadskog sustava definirane su uz pomoć polja zbog jednostavnosti indeksiranja pojedinih kombinacija pomoću brojevima dekadskog sustava, koji se često koriste za brojače i petlje u glavnoj pogramskoj petlji. Oba načina definiranja konstanti segmenata dekadskog sustava prikazani su programskim kodom 13.24.

Programski kod 13.24: Konstante segmenata dekadskog sustava

```
#define NUM_DOT 0x01
#define NUM_0 0xFC
#define NUM_1 0x60
#define NUM_2 0xDA
```

```
#define NUM_3 0xF2
#define NUM_4 0x66
#define NUM_5 0xB6
#define NUM_6 0xBE
#define NUM_7 0xE0
#define NUM_8 0xFE
#define NUM_9 0xF6

const uint8_t charset[10] = {
    0xFC,   // 0
    0x60,   // 1
    0xDA,   // 2
    0xF2,   // 3
    0x66,   // 4
    0xB6,   // 5
    0xBE,   // 6
    0xE0,   // 7
    0xFE,   // 8
    0xF6    // 9
};
```

U biblioteci `SBR_7seg.h` deklarirane su sljedeće funkcije:

- `sbr_init()` - funkcija koja inicijalizira posmačni registar, odnosno konfigurira digitalne izlaze mikroupravljača spojene na ulaze posmačnog registra SI, SCK i RCK,
- `sbr_serial_clock()` - funkcija za generiranje impulsa širine 1 μs na ulazu SCK,
- `sbr_register_clock()` - funkcija za generiranje impulsa širine 1 μs na ulazu RCK,
- `sbr_send_1B(uint8_t data)` - funkcija za slanje jednog bajta podataka (`data`) na posmačni registar, odnosno na ulaz SI,
- `sbr_send_4B(uint8_t d1, uint8_t d2, uint8_t d3, uint8_t d4)`- funkcija za slanje četiriju bajtova (`d1, d2, d3, d4`) na posmačni registar, odnosno na ulaz SI, redoslijedom argumenata funkcije.

U datoteci `SBR_7seg.cpp` definirane su funkcije deklarirane u datoteci `SBR_7seg.h`.

Funkcija `sbr_init()` zamjenjuje inicijalizaciju posmačnih registara u smislu definiranja pinova za ulaze `SI`, `SCK` i `RCK` u posmačne registre. Napisana je tako da koristi konstante definirane u programskom kodu 13.23. Definicija funkcije `sbr_init()` prikazana je programskim kodom 13.25.

Programski kod 13.25: Definicija funkcije `sbr_init()`

```
void sbr_init(void) {
    // inicijalizacija serijskog ulaza (SI) u SBR
    SI_DDR |= (1 << SI_PIN);
    // inicijalizacija register clock ulaza (RCK) u SBR
    RCK_DDR |= (1 << RCK_PIN);
    // inicijalizacija serial clock ulaza (SCK) u SBR
    SCK_DDR |= (1 << SCK_PIN);
}
```

Da bi se doveden podatak na `SI` ulaz posmačnog registra upisao u `Registar za pohranu`, potrebno je na ulazu `SCK` generirati kratak impuls u trajanju od najmanje 1 μs. Kako bi se pojednostavilo generiranje takvog impulsa bez brige o trajanju i pinu na kojem će biti generiran, napisana je funkcija `sbr_serial_clock()` čija je definicija prikazana programskim kodom 13.26.

Programski kod 13.26: Definicija funkcije `sbr_serial_clock()`

```
void sbr_serial_clock(void) {
    SCK_PORT |= (1 << SCK_PIN);
    _delay_us(1);
    SCK_PORT &= ~(1 << SCK_PIN);
    _delay_us(1);
}
```

Kada se upišu svi podaci u `Registar za pohranu`, potrebno je iste prepisati u `Izlazni registar`, koji je povezan s izlaznim pinovima posmačnog registra i time će se podaci ispisati na numerički displej koji je priključen na posmačni registar. Preslikavanje vrijednosti registara aktivira se generiranjem kratkog impulsa u trajanju od najmanje 1 μs na `RCK` ulaz posmačnog registra. Za tu namjenu napisana je funkcija `sbr_register_clock()` čija je definicija prikazana programskim kodom 13.27.

Programski kod 13.27: Definicija funkcije `sbr_register_clock()`

```
void sbr_register_clock(void) {
    RCK_PORT |= (1 << RCK_PIN);
    _delay_us(1);
    RCK_PORT &= ~(1 << RCK_PIN);
    _delay_us(1);
}
```

U prethodnom primjeru opisano je ispisivanje nekog podatka na numerički displej posredno preko posmačnog registra. Da ponovimo ukratko, potrebno je poslati podati na `SI` ulaz posmačnog registra, pomoću kratkog impulsa na `SCK` ulaz posmačnog registra taj podatak se upisuje u `Registar za pohranu`. Da bi se vrijednosti preslikale na fizičke izlazne pinove posmačnog registra, potrebno je vrijednosti iz `Registar za pohranu` prepisati u `Izlazni registar` na čije su pinove izravno spojeni pinovi numeričkog displeja. Posmačni registar je širine 8 bitova, stoga i pohranjeni podatak je širine 8 bitova, što odgovara i broju segmenata numeričkog displeja (7 segmenata i segment `p` - *točka*). Naprikladniji programski način kako upisati 8 bitova podatka je *for* petlja koja se izvrši 8 puta. Tako imamo potpunu kontrolu što se događa u jednoj iteraciji *for* petlje. Za upis podatka širine 8 bitova ili 1 bajt, napisana je funkcija `sbr_send_1B()` čija je definicija prikazana programskim kodom 13.28.

Programski kod 13.28: Definicija funkcije `sbr_send_1B()`

```
void sbr_send_1B(uint8_t data)
{
    for (uint8_t i = 0; i < 8; i++)
    {
        if (data & 0x01)
        {
            SI_PORT |= (1 << SER_PIN);
        }
        else
        {
            SI_PORT &= ~(1 << SER_PIN);
        }

        sbr_serial_clock();

        data = data >> 1;
    }
}
```

Kako bi se pojednostavio ispis na sva 4 displeja dostupna na razvojnom okruženju napisana je funkcija `sbr_send_4B()`. Funkcija ispisuje u jednom retku programskog koda znamenke i vrijednosti na sva 4 displeja. Funkcija je jednostavni četverostruki poziv funkcije `sbr_send_1B()` ranije definirane. Na kraju definicije funkcije generiran je impuls za preslikavanje vrijednosti

registra u **Izlazni registar**, pomoću ranije definirane funkcije `sbr_register_clock()`. Funkcija za ulazne argumente prima cjelobrojne vrijednosti bez predznaka podatkovnog tipa `uint8_t`. Definicija funkcije `sbr_send_4B()` za slanje 4 byte podataka odjednom prikazana je programskim kodom 13.29:

Programski kod 13.29: Definicija funkcije `sbr_send_4B()`

```
void sbr_send_4B(uint8_t d1, uint8_t d2, uint8_t d3, uint8_t d4)
{
    sbr_send_1B(d4);
    sbr_send_1B(d3);
    sbr_send_1B(d2);
    sbr_send_1B(d1);

    sbr_register_clock();
}
```

Promatrajući prethodnu vježbu `vjezba143a.cpp` prikazanu programskim kodom 13.21 može se uvidjeti mogućnost značajnog pojednostavljena programskog koda s očuvanjem potpuno iste namjene - ispis znamenaka dekadskog sustava na sva 4 numerička displeja. U ovoj vježbi zadan je bio ispis znamenki 5, 6, 7 i 8 tim redoslijedom na sva 4 numerička displeja. Konačan sadržaj datoteke `vjezba134b.cpp` prikazan je programskim kodom 13.30:

Programski kod 13.30: Konačan sadržaj datoteke `vjezba134b.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"
#include "Peripheral/SBR_7seg.h"

void init() {
    sbr_init();     // inicijalizacija SBR/7seg.
}

int main(void) {
    init();     // inic. mikroupravljaca

    // posalji 4B podataka odjednom
    // na sva cetiri 7-seg. displeja
    sbr_send_4B(NUM_5, NUM_6, NUM_7, NUM_8);
}
```

Prevedite datoteku `vjezba134b.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Na 4 numerička displeja trebalo bi redom pisati znamenke: 5, 6, 7, 8. Uz pomoć slike 13.12 i tablice 13.1 definirajte vlastiti četveroznamenkasti broj te isti ispišite na 4 numerička displeja. Nakon što ste testirali vježbu, zatvorite datoteku `vjezba134b.cpp` i onemogućite prevođenje ove datoteke.

Na kraju, potrebno je proširiti programski kod iz prethodog primjera na način da će sada na numeričkim displejima biti ispisan rezultat AD pretvorbe kanala `ADC0` na koji je spojen potenciometar. Potrebno je u razmacima od 1 sekunde ispisivati rezultat AD pretvorbe - digitalnu riječ u intervalu od 0 do 1023 i vrijednost koja odgovara naponu na pinu mikroupravljača `PC0` (`ADC0`). Napon treba ispisivati na 3 decimalna mjesta, što znači da na prvom numeričkom displeju treba biti ispisana i točka uz odgovarajuću znamenku. Koristite biblioteku `SBR_7seg.h`.

Potrebno je kreirati novu datoteku `vjezba134c.cpp` i omogućiti prevođenje iste. Početni sadržaj datoteke `vjezba134c.cpp` koji je potrebno napisati prikazan je programskim kodom 13.31.

Programski kod 13.31: Početni sadržaj datoteke `vjezba134c.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"
#include "Peripheral/SBR_7seg.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {
    sbr_init(); // inic. SBR
    adcInit();  // inic. ADC
}

int main(void) {

    init();     // inic. mikroupravljaca

    // varijabla - rezultat AD pretvorbe
    uint16_t adcVal = 0;
    // varijabla - izracunat napon - ADC0
    uint16_t vVal = 0;

    // varijable - ispis rezultata 7SEG
    uint8_t adcNum1 = 0;
    uint8_t adcNum2 = 0;
    uint8_t adcNum3 = 0;
    uint8_t adcNum4 = 0;

    // varijable - ispis napona - ADC0
    uint8_t vNum1 = 0;
    uint8_t vNum2 = 0;
    uint8_t vNum3 = 0;
    uint8_t vNum4 = 0;

    while (1)
    {
        // AD pretvorba
        adcVal = adcRead(ADC0);
        // izracuna napona na ADC0 pinu
        vVal = adcVal * 5000.0 / 1023.0;

        // izdvajanje znamenki rezultata ADC

        // izdvajanje znamenki izrac. napona

        // ispis rezultata ADC na 7SEG

        _delay_ms(1000);

        // ispis izrac. napona na 7SEG

        _delay_ms(1000);
    }
}
```

U inicijalizacijskoj funkciji `init()` pozvane su dvije funkcije. Funkcija za inicijalizaciju posmačnih registara `sbr_init()` i funkcija za inicijalizaciju AD pretvorbe `adc_init()`. U glavnoj programskoj funkciji `main()` pozvana je inicijalizacijska funkcija `init()` nakon koje su deklarirane pomoćne lokalne varijable koje će nam služiti za ispis rezultata AD pretvorbe i napona.

U glavnoj beskonačnoj programskoj petlji `while()`, u svakoj iteraciji izvršava se AD pretvorba na kanalu `ADC0` koji je spojen na analogni pin `PC0` (potenciometar). Nakon očitanja rezultata AD

pretvorbe, potrebno ga je skalirati da bi se dobila vrijednost napona na pinu mikroupravljača `PC0`. Da bi bilo moguće ispisati znamenke na svaki od 4 dostupna numerička displeja na razvojnom okruženju, potrebno je izdvojiti znamenke u posebne varijable, kako za rezultat AD pretvorbe, tako i za izračunatu vrijednost napona. Izdvajanje znamenaka izvedeno je pomoću cjelobrojnog dijeljenja i operatora *modulo*, što je i prikazano programskim kodom 13.32:

Programski kod 13.32: Izdvajanje znamenaka AD pretvorbe i izračunatog napona

```
// izdvajanje znamenki rezultata ADC
adcNum1 = adcVal / 1000;
adcNum2 = (adcVal / 100) % 10;
adcNum3 = (adcVal / 10) % 10;
adcNum4 = adcVal % 10;

// izdvajanje znamenki izrac. napona
vNum1 = vVal / 1000;
vNum2 = (vVal / 100) % 10;
vNum3 = (vVal / 10) % 10;
vNum4 = vVal % 10;
```

Nakon izdvajanja znamenaka AD pretvorbe i izračunatog napona, preostalo ih je ispisati na numeričke displeje. Korištena je funkcija `sbr_send_4B()` iz biblioteke `SBR_7seg.h` kojoj su proslijeđene vrijednosti polja `charset[]`, koje za argument prima izdvojene znamenke. Tim načinom se posredno dostupa do heksadekadskih vrijednosti segmenata za pojedinu znamenku, odnosno argument. Na numerički displej na kojem želimo prikazati točku uz znamenku, potrebno je uz argument dodati logičku operaciju *ILI* s konstatnom segmenta nazvanom `NUM_DOT`. Potpuni sadržaj datoteke `vjezba134c.cpp` prikazan je programskim kodom 13.33:

Programski kod 13.33: Potpuni sadržaj datoteke `vjezba134c.cpp`

```
#include "AVR/avrlib.h"
#include "ADC/adc.h"
#include "Peripheral/SBR_7seg.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {
    sbr_init(); // inic. SBR
    adcInit();  // inic. ADC
}

int main(void) {

    init();     // inic. mikroupravljaca

    // varijabla - rezultat AD pretvorbe
    uint16_t adcVal = 0;
    // varijabla - izracunat napon - ADC0
    uint16_t vVal = 0;

    // varijable - ispis rezultata 7SEG
    uint8_t adcNum1 = 0;
    uint8_t adcNum2 = 0;
    uint8_t adcNum3 = 0;
    uint8_t adcNum4 = 0;

    // varijable - ispis napona - ADC0
    uint8_t vNum1 = 0;
    uint8_t vNum2 = 0;
    uint8_t vNum3 = 0;
    uint8_t vNum4 = 0;
```

```
    while (1)
    {
        // AD pretvorba
        adcVal = adcRead(ADC0);
        // izracuna napona na ADC0 pinu
        vVal = adcVal * 5000.0 / 1023.0;

        // izdvajanje znamenki rezultata ADC
        adcNum1 = adcVal / 1000;
        adcNum2 = (adcVal / 100) % 10;
        adcNum3 = (adcVal / 10) % 10;
        adcNum4 = adcVal % 10;

        // izdvajanje znamenki izrac. napona
        vNum1 = vVal / 1000;
        vNum2 = (vVal / 100) % 10;
        vNum3 = (vVal / 10) % 10;
        vNum4 = vVal % 10;

        // ispis rezultata ADC na 7SEG
        sbr_send_4B(charset[adcNum1], charset[adcNum2], charset[adcNum3],
                                                      charset[adcNum4]);
        _delay_ms(1000);

        // ispis izrac. napona na 7SEG
        sbr_send_4B(charset[vNum1] | NUM_DOT, charset[vNum2], charset[vNum3],
                                                           charset[vNum4]);
        _delay_ms(1000);
    }
}
```

Prevedite datoteku `vjezba134c.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P.

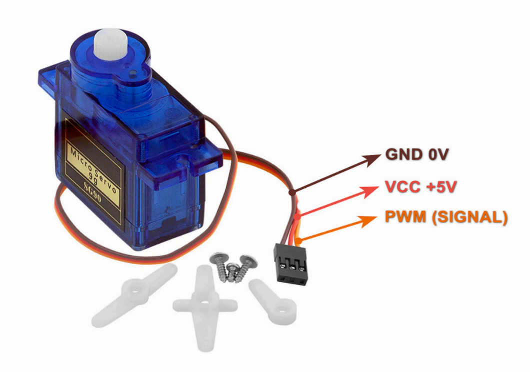
Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Na 4 numerička displeja trebale bi se izmjenjivati dvije četveroznamenkaste vrijednosti. Ukoliko je potenciometar odvrnut do krajnjeg položaja u smjeru suprotnom kazaljci na satu, trebale bi se izmjenjivati vrijednosti 1023 i 5,000. Ukoliko je potenciometar odvrnut do krajnjeg položaja u smjeru kazaljke na satu, vrijednosti koje se izmjenjuju trebale bi biti 0000 i 0,0000. Pokušajte odrediti otprilike sredinu zakreta potenciometra te provjeriti da li su vrijednosti koje se izmjenjuju približno jednake 511 i 2,500.

Ukoliko su testovi uspješni, zatvorite datoteku `vjezba134c.cpp` i onemogućite prevođenje ove datoteke.

13.5 Servomotor

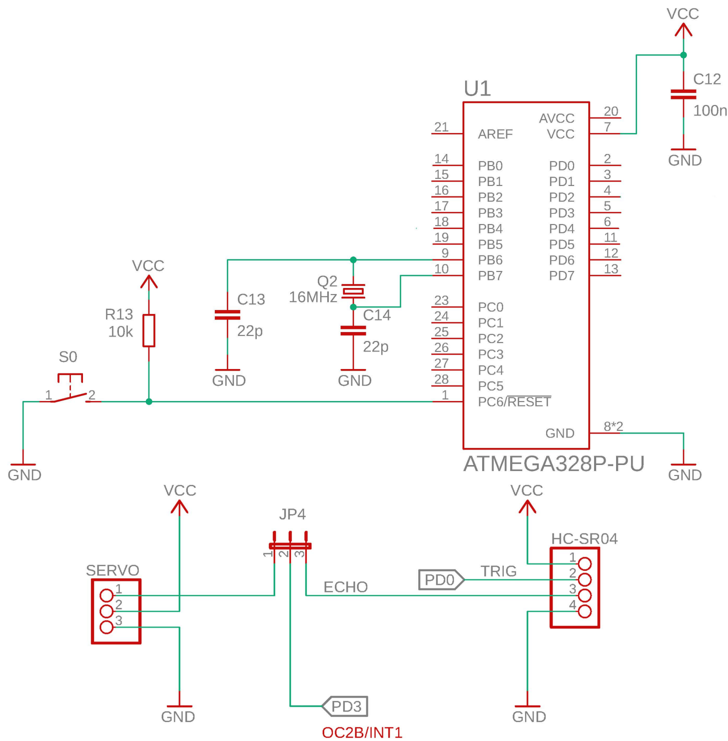
Servomotor je vrsta elektromotora upravljana po poziciji. Sastoji se od kućišta u kojem se nalazi uobičajeni istosmjerni motor s pripadajućim enkoderom u povratnoj vezi, elektronika za generiranje upravljačkog signala (engl. *driver*), prijenosna kutija zupčastog prijenosa i izlazna osovine. Servo motori se koriste za namjene gdje je potrebno vrlo precizno pozicioniranje izlazne osovine motora, kao što su upravljanja mehanizmima, vozilima, letjelicama i slično. Potrebno je napomenuti da servomotore odlikuje dobro držanje zadane pozicije i velik moment na izlaznoj osovini. Razlikujemo servomotore malih dimenzija (hobi servomotori) i industrijske servomotore. Zbog jednostavno upravljačke elektronike i signala potrebnih za upravljanje, u ovoj vježbi koristiti će se hobi servomotor malih dimenzija generičke oznake *SG90* (90 dolazi od 90 grama). Takvi motori obično su napajani naponom *TTL* razina i upravljani *PWM* signalom istih razina. Servomotor SG90 ima tri žice (slika 13.16):

- narančasta (nekad žuta) žica je signalna žica (*PWM* signal) za upravljanje kutem zakreta,
- smeđa žica spaja se na GND ili 0 V i
- crvena žica spaja se na VCC ili +5 V.



Slika 13.16: Servo motor SG90 s pripadajućim spojnim žicama

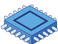
Upravljanje pozicijom izlazne osovine servomotora zahtijeva posebnu vrstu upravljačkog signala, nazvanog pulsno širinski moduliran signal (engl. *Pulse Width Modulation; PWM*), koji je ranije objašnjen u poglavlju `Pulsno širinska modulacija`. Takav signal je vremenski određen i generira se pomoću sklopa mikroupravljača *Timer/Counter*. Izlazni signal je proslijeđen na točno određene pinove koji mogu imati takvu alternativnu funkciju. Shema spajanja servomotora na mikroupravljač ATmega328P prikazana je na slici 13.17.



Slika 13.17: Shema spajanja servomotora na razvojno okruženje s mikroupravljačem ATmega328P

Na neki način potrebno je dinamički zadavati referencu kuta zakreta servomotora. Jedan

od načina je upotreba potenciometra, koji je potrebno uključiti na razvojnom okruženju postavljanjem kratkospojnika JP6 tako da spaja srednji trn i trn nazvan POT. Da bi servomotor imao napajanje preko razvojnog okruženja, potrebno je postaviti kratkospojnik JP4 tako da spaja srednji trn i trn nazvan SERVO. Potrebno je pripaziti na orijentaciju priključnih žica servomotora na razvojno okruženje (s lijeva na desno prvi trn je GND, ako se gleda natpis SERVO na tiskanoj pločici razvojnog okruženja).



Vježba 13.5

Potrebno je napraviti program koji će upravljati zakretom izlazne osovine servomotora od 0° do 180°. Referencu je potrebno izračunati iz rezultata AD pretvorbe na kanalu ADC0 na koji je spojen potenciometar. Zakretom potenciometra servomotor u što kraćem vremenu treba postići zadani kut zakreta izlazne osovine. Na LCD displeju potrebno je ispisivati referentnu vrijednost kuta i vrijednost *Duty Cycle* širine visokog stanja upravljačkog *PWM* signala. U daljnjem tijeku vježbe potrebno je napisati funkciju koja će u jednoj liniji programskog koda izvršiti pozicioniranje servomotora, umjesto do sada u nekoliko linija programskog koda. Shema spajanja servomotora i mikroupravljača ATmega328P prikazana je na slici 13.17. Potrebno je koristiti sklop *Timer/Counter2* s pripadajućim izlaznim kanalom OC2B.

U projektnom stablu otvorite datoteku vjezba135a.cpp. Omogućite samo prevođenje datoteke vjezba135a.cpp. Početni sadržaj datoteke vjezba135a.cpp prikazan je programskim kodom 13.34.

Programski kod 13.34: Početni sadržaj datoteke vjezba135a.cpp

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "ADC/adc.h"
#include <avr/io.h>
#include <util/delay.h>

// pokusno dobivene ref. vr. DC za servomotor

void init()
{
    // inicijalizacija LCD
    // inicijalizacija ADC

    // timer 2 nacin rada phase correct u mode 5 (OCR2A = top)
    // djelitelj frekvencije 1024
    // top vrijednost do koje timer broji
    // OC2B kanal izlazni PWM signala, nalazi se na pinu PD3

}

int main(void){

    // inicijalizacija mikroupravljaca

    // varijabla ADC
    // varijable: kut, duty cycle

    while(1) {

        // ADC ch ADC0 (POT)
```

```
        // skaliranje ADC (0 do 1023) -> kut (0 do 180)

        if(rotAngle <= 90.0) {      // ako je kut < 90

            // skaliraj duty cycle da krece od 4.3 i ide do 11.7

        }
        else {      // ako je kut > 90

            // skaliraj duty cycle da ide krece od 11.7 i ide do 20.22

        }

            // azuriranje DC za PWM signal pin OC2B

            // ocisti prethodni ispis na LCD
            // postavljanje pokazivaca LCD na koordinate 1,1
            // ispis DC 1.red LCD
            // ispis kuta 2.red LCD

        _delay_ms(150);
    }
}
```

U programskom kodu 13.34 potrebno je funkcijom `init()` inicijalizirati LCD displej i AD pretvornik. Za upravljanje servomotorom koristit će se sklop *Timer/Counter2*. Generirani *PWM* signal bit će proslijeđen na izlaz `OC2B` koji je spojen na fizički pin `PD3`. Sklop će raditi u *Phase Correct, PWM* načinu rada s djeliteljem frekvencije radnoga takta iznosa 1024. Signal će biti neinvertiran. Za upravljanje servomotorom potrebna je frekvencija iznosa 50 Hz, stoga je potrebno izračunati gornju (*TOP*) vrijednost do koje će sklop *Timer/Counter2* brojiti. Gornja (*TOP*) vrijednost može se izračunati prema relaciji 13.7.

$$f_{OC2B} = \frac{f_{CPU}}{prescaler \cdot 2 \cdot TOP} \rightarrow TOP = \frac{f_{CPU}}{prescaler \cdot 2 \cdot f_{OC2B}} \tag{13.7}$$

gdje su:

- TOP - vrijednost do koje će brojiti sklop *Timer/Counter2*,
- f_{CPU} - frekvencija mikroupravljača,
- $prescaler$ - djelitelj frekvencije mikroupravljača,
- f_{OC2B} - frekvencija *PWM* signala na pinu `OC2B`.

Izračunata vrijednost je 156,25 koja se zaokruži na konačnu cjelobrojnu vrijednost 156. Potrebno je pohraniti istu u registar `OCR2A`, jer u načinu rada *Phase Correct, PWM, mode 5* sklopa *Timer/Counter2* vrijednost do koje broji sklop pohranjuje se u `OCR2A` 8-bitni registar. Pin `PD3` definiran je kao izlazni pin. Da bi prekidne rutine bile dostupne, potrebno ih je globalno omogućiti pomoću funkcije `interruptEnable()`. Potpuno napisana funkcija `init()` za inicijalizaciju mikroupravljača prikazana je programskim kodom 13.35. Osvježite funkciju `init()` u datoteci `vjezba135a.cpp` kako bi bila istovjetna programskom kodu 13.35.

Programski kod 13.35: Programski kod inicijalizacijske funkcije `init()`

```
void init()
{
    lcdInit();  // inicijalizacija LCD
    adcInit();  // inicijalizacija ADC
```

```
    // timer 2 nacin rada phase correct u mode 5 (OCR2A = top)
    timer2PhaseCorrectPWM_mode5();
    // djelitelj frekvencije 1024
    timer2SetPrescaler(TIMER2_PRESCALER_1024);
    // top vrijednost do koje timer broji
    OCR2A = 156;
    // OC2B kanal izlazni PWM signala, nalazi se na pinu PD3
    DDRD |= (1<<PD3);
}
```

U `main()` funkciji pozvana je prethodno opisana funkcija `init()` za inicijalizaciju mikroupravljača. Nakon toga deklarirana je 16-bitna pomoćna varijabla `varPot` za pohranu rezultata AD pretvorbe na kanalu `ADC0` na koji je spojen potenciometar. Definirane su dodatne dvije varijable, `rotAngle` za pohranu referentne vrijednosti kuta zakreta i `duty` za pohranu vrijednosti izračunate širine visokog stanja *PWM* signala - *Duty Cycle*, kao što je to prikazano programskim kodom 13.36. Osvježite funkciju `main()` u datoteci `vjezba135a.cpp` s programskim kodom 13.36.

Programski kod 13.36: Poziv funkcija i deklaracije pomoćnih varijabli

```
init(); // inicijalizacija mikroupravljaca
uint16_t varPot = 0;     // varijabla ADC
float rotAngle, duty;    // varijable: kut, duty cycle
```

U glavnoj beskonačnoj petlji `while()` izvršava se AD pretvorba u svakoj iteraciji. Rezultat se sprema u varijablu `varPot`. S obzirom na zakret osovine potenciometra, spremljena vrijednost se nalazi u intervalu od 0 do 1023, jer je AD pretvornik rezolucije 10 bitova ($2^{10} - 1 = 1023$). Tu vrijednost je potrebno pretvoriti u vrijednost koja se nalazi u intervalu od 0° do 180°. Metoda pretvorbe je jednostavno skaliranje prema relaciji 13.8.

$$kut(rotAngle) = \frac{ADC0}{1023.0} \cdot 180^\circ \tag{13.8}$$

gdje su:

- $kut(rotAngle)$ - zadana/referentna vrijednost kuta zakreta izlazne osovine servomotora i
- $ADC0$ - vrijendost AD pretvorbe dobivena na kanalu `ADC0`.

Rezultat će biti vrijednost korištena kao referenca kuta zakreta servomotora. Kao što je to prethodno opisano, servomotor zahtijeva pulsno širinski moduliran *PWM* signal, frekvencije 50 Hz. Takva frekvencija uključuje da period signala treba biti trajanja 20 ms, što je moguće potvrditi relacijom 13.9:

$$T_{PWM}\,[s] = \frac{1}{f_{PWM}\,[Hz]} \tag{13.9}$$

gdje je:

- T_{PWM} - vrijeme trajanja perioda *PWM* signala i
- f_{PWM} - frekvencija *PWM* signala.

Referentni kutevi zakreta izlazne osovine servomotora, odgovaraju sljedećima širinama visokog stanja *PWM* signala - *Duty Cycle*:

- širina visokog stanja *PWM* signala = 1 ms (*Duty Cycle* iznosi 5 %), tada će izlazna osovina servomotora biti zakrenuta za 0°,

- širina visokog stanja *PWM* signala = 1,5 ms (*Duty Cycle* iznosi 7,5 %), tada će izlazna osovina servomotora biti zakrenuta za 90°,
- širina visokog stanja *PWM* signala = 2 ms (*Duty Cycle* iznosi 10 %), tada će izlazna osovina servomotora biti zakrenuta za 180°.

Međutim, te vrijednosti na stvarnom primjeru upravljanja servomotorom ne daju točne kuteve zakreta. Odstupanja su učestala pojava zbog nesavršenosti izrade servomotora i enkodera. Utjecaj ima i nemogućnost dovoljno preciznog definiranja svojstava *PWM* signala, obično zbog nedostatka visoko preciznih resursa na mikroupravljaču (16-bitni sklopovi *Timer/Counter*). Iskustveno, pokusom su dobivene sljedeće vrijednosti:

- širina visokog stanja *PWM* signala = 0.86 ms (*Duty Cycle* iznosi 4.3 %), tada će izlazna osovina servomotora biti zakrenuta za 0°,
- širina visokog stanja *PWM* signala = 2,34 ms (*Duty Cycle* iznosi 11.7 %), tada će izlazna osovina servomotora biti zakrenuta za 90°,
- širina visokog stanja *PWM* signala = 40,44 ms (*Duty Cycle* iznosi 20.22 %), tada će izlazna osovina servomotora biti zakrenuta za 180°.

Dobivene konstantne trebaju biti vidljive i dostupne cijelom programskom kodu, uključujući prekidne rutine i funkcije. Da bi to tako bilo, potrebno ih je napisati odmah ispod pretprocesorskih naredbi kao konstante. Dodijeljena imena konstantama i pripadne vrijednosti istih prikazane su programskim kodom 13.37. Osvježite datoteku `vjezba135a.cpp` s programskim kodom 13.37.

Programski kod 13.37: Konstantne širine visokog stanja PWM signala - *Duty Cycle*

```
// pokusno dobivene ref. vr. DC za servomotor
#define DUTY_MIN 4.3        // za kut 0
#define DUTY_MAX 20.22      // za kut 180
#define DUTY_MID 11.7       // za kut 90
```

Ako je referenca kuta zakreta manja od 90°, potrebno je skalirati širinu visokog stanja *PWM* signala tako da se nalazi u intervalu od 4,3% (`DUTY_MIN`) do 11,7% (`DUTY_MID`), što ujedno predstavlja zakret osovine servomotora od 0° do 90°. Izračun slijedi prema relaciji 13.10.

$$dutyCycle = DUTY_MIN + (DUTY_MID - DUTY_MIN)/90.0 \cdot rotAngle \tag{13.10}$$

gdje je:

- $DUTY_MIN$ - vrijednost *Duty Cycle* za položaj servomotora = 0°,
- $DUTY_MID$ - vrijednost *Duty Cycle* za položaj servomotora = 90°,
- $rotAngle$ - referentna pozicija kuta zakreta izlazne osovine servomotora i
- $dutyCycle$ - izračunata širina visokog stanja *Duty Cycle* - *PWM* signala.

Inače, ako je referenca kuta veća od 90°, potrebno je skalirati širinu visokog stanja *PWM* signala tako da se nalazi u intervalu od 11,7% (`DUTY_MID`) do 20,22 %(`DUTY_MAX`), što ujedno predstavlja zakret osovine servomotora od 90° do 180°. Izračun slijedi prema relaciji 13.11.

$$dutyCycle = DUTY_MID + (DUTY_MAX - DUTY_MID)/90.0 \cdot (rotAngle - 90.0) \tag{13.11}$$

gdje je:

- $DUTY_MAX$ - vrijednost *Duty Cycle* za položaj servomotora = 180°.

Pripremljenu vrijednost širine PWM signala pohranjenu u varijablu `duty`, potrebno je pohraniti u registar OCR2B. Time je definirana širina visokog stanja PWM signala na fizičkom izlazu odnosno pinu PD3, na koji je spojena signalna žica za upravljački signal servomotora. Opisani primjer upotrebe servomotora na razvojnom okruženju upravljanom mikroupravljačem ATmega328P prikazan je programskim kodom 13.38. Osvježite datoteku `vjezba135a.cpp` kako bi bila istovjetna programskom kodu 13.38.

Programski kod 13.38: Potpuni sadržaj datoteke `vjezba135a.cpp`

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "ADC/adc.h"
#include <avr/io.h>
#include <util/delay.h>

// pokusno dobivene ref. vr. DC za servomotor
#define DUTY_MIN 4.3        // za kut 0
#define DUTY_MAX 20.22      // za kut 180
#define DUTY_MID 11.7       // za kut 90

void init()
{
    lcdInit();  // inicijalizacija LCD
    adcInit();  // inicijalizacija ADC

    // timer 2 nacin rada phase correct u mode 5 (OCR2A = top)
    timer2PhaseCorrectPWM_mode5();
    // djelitelj frekvencije 1024
    timer2SetPrescaler(TIMER2_PRESCALER_1024);
    // top vrijednost do koje timer broji
    OCR2A = 156;
    // OC2B kanal izlazni PWM signala, nalazi se na pinu PD3
    DDRD |= (1<<PD3);
}

int main(void){

    init(); // inicijalizacija mikroupravljaca

    uint16_t varPot = 0;    // varijabla ADC
    float rotAngle, duty;   // varijable: kut, duty cycle

    while(1) {

        varPot = adcRead(ADC0); // ADC ch ADC0 (POT)
        // skaliranje ADC (0 do 1023) -> kut (0 do 180)
        rotAngle = varPot / 1023.0 * 180.0;

        if(rotAngle <= 90.0) {      // ako je kut < 90
            // skaliraj duty cycle da krece od 4.3 i ide do 11.7
            duty = DUTY_MIN+(DUTY_MID-DUTY_MIN)/90.0*rotAngle;
        }
        else {      // ako je kut > 90
            // skaliraj duty cycle da ide krece od 11.7 i ide do 20.22
            duty = DUTY_MID+(DUTY_MAX-DUTY_MID)/90.0*(rotAngle-90.0);
        }
```

```
        OCR2B = (uint8_t)duty;  // azuriranje DC za PWM signal pin OC2B

        lcdClrScr();    // ocisti prethodni ispis na LCD
        lcdHome();      // postavljanje pokazivaca LCD na koordinate 1,1
        lcdprintf("duty = %d\n",  (uint8_t)duty); // ispis DC 1.red LCD
        lcdprintf("angle = %0.3f\n", rotAngle); // ispis kuta 2.red LCD

        _delay_ms(150);
    }
}
```

Prevedite datoteku `vjezba135a.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Isprobajte precizno postavljanje reference kuta zakreta servomotora potenciometrom. Izbrojite dosegljive pozicije izlazne osovine servomotora. Usporedite i komentirajte utjecaj rezolucije korištenog sklopa *Timer/Counter2* i sklopa *Timer/Counter1* na broj dosegljivih pozicija kuta zakreta servomotora. Nakon što ste testirali vježbu, zatvorite datoteku `vjezba135a.cpp` i onemogućite prevođenje ove datoteke.

Kako bi se postigla veća jednostavnost i čitljivost programskog koda, uvedene su funkcije koje zamjenjuju više linija koda obično jednom linijom koda. Taj princip je koristan u ovoj vježbi i primijeniti ćemo ga na izračun širine visokog stanja *PWM* signala. Prethodno opisan programski kod treba zamijeniti funkcijom `setServoPosition()`. Funkcija će u jednoj liniji koda izvršiti zakret osovine servomotora na zadanu vrijednost kuta zakreta, a da pritom se izračun širine visokog stanja PWM signala nalazi u skrivenom sloju i nije vidljiv glavnoj programskoj funkciji `main()`. Unutar postojećeg projekta `Moduli` u programskom alatu *Microchip Studio 7*, napravite novu datoteku imena i ekstenzije `vjezba135b.cpp`. U novo izrađenoj datoteci potrebno je napisati izmijenjeni programski kod s funkcijom `setServoPosition()`. Opisani primjer upotrebe servomotora na razvojnom okruženju upravljanom mikroupravljačem ATmega328P korištenjem funkcije `setServoPosition()` u glavnoj programskoj funkciji prikazan je programskim kodom 13.39. Osvježite datoteku `vjezba135b.cpp` kako bi bila istovjetna programskom kodu 13.39.

Programski kod 13.39: Potpuni sadržaj datoteke `vjezba135b.cpp`

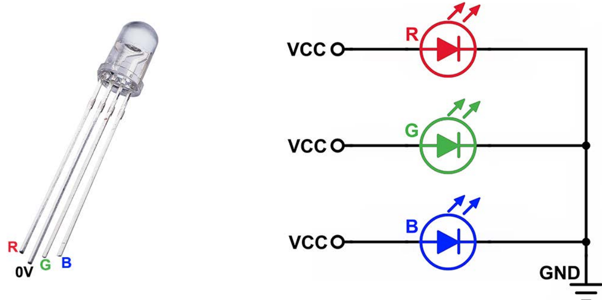
```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "ADC/adc.h"
#include <avr/io.h>
#include <util/delay.h>

// pokusno dobivene ref. vr. DC za servomotor
#define DUTY_MIN 4.3        // za kut 0
#define DUTY_MAX 20.22      // za kut 180
#define DUTY_MID 11.7       // za kut 90

void init()
{
    lcdInit();  // inicijalizacija LCD
    adcInit();  // inicijalizacija ADC

    // timer 2 nacin rada phase correct u mode 5 (OCR2A = top)
    timer2PhaseCorrectPWM_mode5();
    // djelitelj frekvencije 1024
    timer2SetPrescaler(TIMER2_PRESCALER_1024);
    // top vrijednost do koje timer broji
    OCR2A = 156;
    // OC2B kanal izlazni PWM signala, nalazi se na pinu PD3
```

```
    DDRD |= (1<<PD3);
}

void setServoPosition(float angle)
{
    uint8_t dutyCycle = 0;

    if(angle <= 90.0) {     // ako je kut < 90
        // skaliraj duty cycle da krece od 4.3 i ide do 11.7
        dutyCycle = DUTY_MIN+(DUTY_MID-DUTY_MIN)/90.0*angle;
    }
    else {      // ako je kut > 90
        // skaliraj duty cycle da ide krece od 11.7 i ide do 20.22
        dutyCycle = DUTY_MID+(DUTY_MAX-DUTY_MID)/90.0*(angle-90.0);
    }

    OCR2B = (uint8_t)dutyCycle; // azuriranje DC za PWM signal pin OC2B
}

int main(void){

    init(); // inicijalizacija mikroupravljaca

    uint16_t varPot = 0;    // varijabla ADC
    float rotAngle, duty;   // varijable: kut, duty cycle

    while(1) {

        varPot = adcRead(ADC0); // ADC ch ADC0 (POT)
        // skaliranje ADC (0 do 1023) -> kut (0 do 180)
        rotAngle = varPot / 1023.0 * 180.0;

        setServoPosition(rotAngle); // fnc - servomotor position update

        lcdClrScr();    // ocisti prethodni ispis na LCD
        lcdHome();      // postavljanje pokazivaca LCD na koordinate 1,1
        lcdprintf("duty = %d\n",  (uint8_t)duty); // ispis DC 1.red LCD
        lcdprintf("angle = %0.3f\n", rotAngle); // ispis kuta 2.red LCD

        _delay_ms(150);
    }
}
```

Prevedite datoteku `vjezba135b.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Isprobajte precizno postavljanje reference kuta potenciometrom. Usporedite ima li razlike u programskom kodu koji koristi funkciju za pozicioniranje servomotora s prethodnim kodom koji ne koristi tu funkciju. Zatvorite datoteku `vjezba135b.cpp` i onemogućite prevođenje ove datoteke.

13.6 RGB dioda

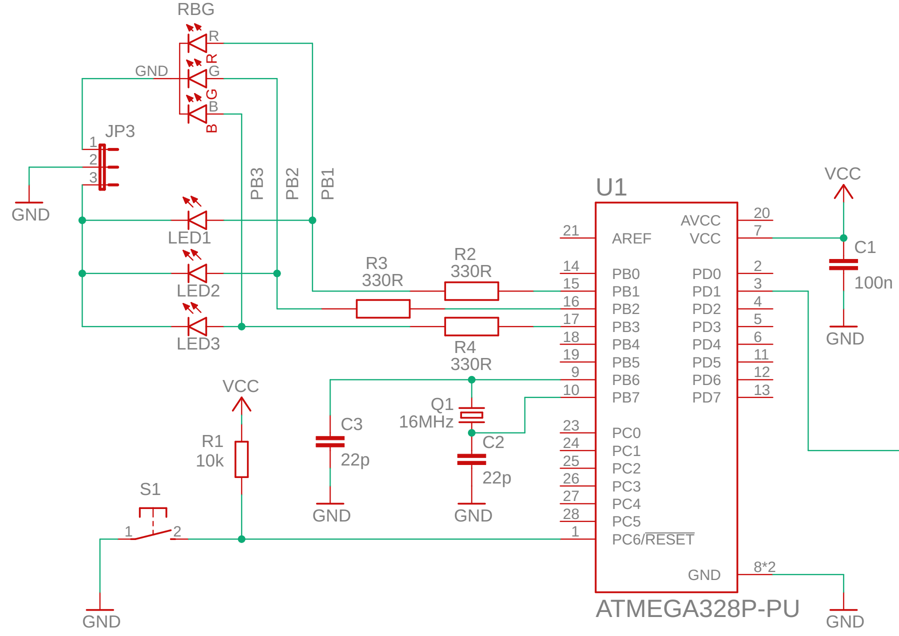
RGB dioda je nalik običnoj svjetlećoj diodi (LED) prozirne boje kao što se može vidjeti na slici 13.18.



Slika 13.18: RGB dioda - komponenta i shema

Obična LED dioda može emitirati svjetlo jedne nepromjenjive boje, dok RGB dioda može prikazati tri boje istodobno. Može se reći da je RGB dioda kombinacija triju LED dioda u jednoj. Svaka od te tri LED diode ima svoju boju, kojoj je moguće postavljati intenzitet svjetlosti. Kanali boje su R (engl. *Red*) za crvenu, G (engl. *Green*) za zelenu i B (engl. *Blue*) za plavu boju, što čini osnovne komponente RGB spektra boja. RGB boje u kombinaciji bi trebale moći ostvarivati većinu boja iz osnovne palete boja. RGB dioda prikazana na slici 13.18 tip je diode sa zajedničkom katodom, što znači da će se pojedina komponenta uključivati visokom razinom signala, dok zajednički kraj je spojen na masu ili nisku razinu signala (0 V).

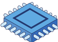
RGB dioda zahtijeva poseban način spajanja, jer za upravljanje koristi izlazne *PWM* signale *Timer/Counter* sklopova. Izlazni kanali s takvim generiranim signalima su alternativna funkcija određenih pinova mikroupravljača i nije moguće koristiti takve funkcije na ostalim pinovima. Stoga, potrebno je spojiti RGB diodu na mikroupravljač ATmega328P prema shemi na slici 13.19.



Slika 13.19: Shema spajanja RGB diode na mikroupravljač ATmega328P

Kratkospojnik JP3 potrebno je postaviti između srednjeg trna i trna nazvanog RGB, kako bi RGB dioda imala napajanje i bila spojena na pinove mikroupravljača PB1, PB2 i PB3. Kao što je uobičajeno da se kod izravnog spajanja LED diode na pin mikroupravljača spoji u seriju otpornik iznosa 330 kΩ, tako je i ovdje za svaki pojedini kanal spojen i pripadajući otpornik u seriju s kanalom RGB diode.

Mikroupravljač ATmega328P kao i većina drugih mikroupravljača, nema integriran pretvornik digitalnog u analogni signal (engl. *digital-to-analog; DAC*). Uz pomoć digitalnog signala koji je pulsno širinski moduliran, moguće dobiti analogni signal, odnosno srednju efektivnu vrijednost signala kroz vrijeme. Visina napona će biti određena širinom visokog stanja *PWM* signala. RGB dioda je dovoljno spor optoelektronički element, odnosno naše oko dovoljno sporo reagira na promjene signala, kako bi ta svjetlost djelovala neprekidno u našoj vizualnoj percepciji.

 Vježba 13.6

Potrebno je napraviti program koji će postaviti vrijednosti intenziteta svjetlosti R, G i B komponenti RGB diode. Vrijednosti trebaju biti postavljene trajno u programskom kodu na sljedeće: R=10, G=10 i B=10. Koristite *Timer/Counter1* i *Timer/Counter2* sklopove te njihove pripadajuće izlazne pinove za prosljeđivanje generiranih *PWM* signala. Izlazni pinovi su OC1A, OC1B i OC2A. Spojeni su na pinove PB1, PB2 i PB3 istim redoslijedom. Potrebno je ispisati na prvi redak LCD displeja vrijednosti R, G i B komponenata RGB diode, na primjer R10 G10 B10. Shema spajanja RGB diode na mikroupravljač prikazana je na slici 13.19.

U projektnom stablu otvorite datoteku vjezba136a.cpp. Omogućite samo prevođenje datoteke vjezba136a.cpp. Početni sadržaj datoteke vjezba136a.cpp prikazan je programskim kodom 13.40.

Programski kod 13.40: Početni sadržaj datoteke vjezba136a.cpp

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {

    // inicijalizacija LCD displeja
    // djelitelj frekvencije
    // djelitelj frekvencije
    // timer 1 - phase correct u 8bit
    // timer 2 - phase correct
    // neinvertirajuci PWM pin OC1A
    // neinvertirajuci PWM pin OC1B
    // neinvertirajuci PWM pin OC2A
    // pinovi izlazni PWM
    // globalno omoguci prekidne rutine
}

int main(void) {

    init();     // poziv funkcije za inicijalizaciju mikroracunala

    uint8_t RGB[3] = {10, 10, 10};      // polje RGB kanala LED diode
```

```
    while(1) {

        timer1OC1ADutyCycle(RGB[0]);    // PWM signal na kanalu OC1A - PB1
        timer1OC1BDutyCycle(RGB[1]);    // PWM signal na kanalu OC1B - PB2
        timer2OC2ADutyCycle(RGB[2]);    // PWM signal na kanalu OC2A - PB3

        lcdClrScr();    // ocisti LCD displej
        lcdHome();      // postavljanje pokazivaca LCD displeja
        lcdprintf("R%u G%u B%u", RGB[0], RGB[1], RGB[2]);   // ispis

        _delay_ms(250);

    }
}
```

U programskom kodu 13.40 pomoću funkcije `init()` potrebno je inicijalizirati LCD displej i sklopove mikroupravljača *Timer/Counter1* i *Timer/Counter2*. Za generiranje *PWM* signala za upravljanje R, G i B komponentama RGB diode, koristit će se izlazni pinovi `PB1`, `PB2` i `PB3`, koji su spojeni na izlazne *PWM* kanale `OC1A`, `OC1B` i `OC2A`. *Timer/Counter1* i *Timer/Counter2* sklopovi radit će u *Phase Correct, PWM* načinu rada s djeliteljem frekvencije radnoga takta iznosa 64. Na pinovima `OC1A`, `OC1B` i `OC2A` omogućeni su neinvertirani *PWM* signali. Također, pinovi `OC1A` (`PB1`), `OC1B` (`PB2`) i `OC2A` (`PB3`) postavljeni su kao izlazni pinovi. Da bi prekidne rutine bile dostupne za korištenje, potrebno ih je globalno omogućiti pomoću funkcije `interruptEnable()`. Potpuno napisana inicijalizacijska funkcija `init()` prikazana je programskim kodom 13.41. Osvježite funkciju `init()` u datoteci `vjezba136a.cpp` kako bi bila istovjetna programskom kodu 13.41.

Programski kod 13.41: Potpuno napisana inicijalizacijska funkcija `init()`

```
void init() {

    lcdInit();      // inicijalizacija LCD displeja

    timer1SetPrescaler(TIMER1_PRESCALER_64);    // djelitelj frekvencije
    timer2SetPrescaler(TIMER2_PRESCALER_64);    // djelitelj frekvencije

    timer1PhaseCorrectPWM8bit();    // timer 1 - phase correct u 8bit
    timer2PhaseCorrectPWM();        // timer 2 - phase correct

    timer1OC1AEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC1A
    timer1OC1BEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC1B
    timer2OC2AEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC2A

    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);   // pinovi izlazni PWM

    interruptEnable();      // globalno omoguci prekidne rutine
}
```

U glavnoj programskoj funkciji `main()`, pozvana je prethodno opisana funkcija `init()` za inicijalizaciju mikroupravljača. Nakon toga, deklarirano je cjelobrojno 8-bitno polje bez predznaka, u koje će biti pohranjene vrijednosti R, G i B komponenata RGB diode. U glavnoj beskonačnoj programskoj petlji `while()`, postavljaju se vrijednosti širine visokog stanja PWM signala - (engl. *Duty Cycle; DC*), pomoću sljedećih funkcija koje kao argument primaju vrijednosti RGB polja:

- `timer1OC1ADutyCycle(RGB[0]);`
- `timer1OC1BDutyCycle(RGB[1]);`
- `timer2OC2ADutyCycle(RGB[2]);`

Nakon postavljanja visokog stanja *PWM* signala na kanalima OC1A, OC1B i OC2A, iste je potrebno ispisati na prvi redak LCD displeja.

PWM signal moguće je snimiti uređajima za analiziranje signala, kao što su osciloskop ili digitalni analizator. Preporuča se snimanje signala u edukacijske svrhe. Specifikacije *PWM* signala su objašnjene potanko u poglavlju `Pulsno širinska modulacija`.

Potpuni sadržaj datoteke `vjezba136a.cpp` gdje je inicijaliziran mikroupravljač s potpunim algoritmom za postavljanje rada RGB diode nepromjenjivim intenzitetom, prikazan je programskim kodom 13.42.

Programski kod 13.42: Potpuni sadržaj datoteke `vjezba136a.cpp`

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {

    lcdInit();      // inicijalizacija LCD displeja

    timer1SetPrescaler(TIMER1_PRESCALER_64);    // djelitelj frekvencije
    timer2SetPrescaler(TIMER2_PRESCALER_64);    // djelitelj frekvencije

    timer1PhaseCorrectPWM8bit();    // timer 1 - phase correct u 8bit
    timer2PhaseCorrectPWM();        // timer 2 - phase correct

    timer1OC1AEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC1A
    timer1OC1BEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC1B
    timer2OC2AEnableNonInvertedPWM();   // neinvertirajuci PWM pin OC2A

    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);   // pinovi izlazni PWM

    interruptEnable();      // globalno omoguci prekidne rutine
}

int main(void) {

    init();     // poziv funkcije za inicijalizaciju mikroracunala

    uint8_t RGB[3] = {10, 10, 10};      // polje RGB kanala LED diode

    while(1) {

        timer1OC1ADutyCycle(RGB[0]);    // PWM signal na kanalu OC1A - PB1
        timer1OC1BDutyCycle(RGB[1]);    // PWM signal na kanalu OC1B - PB2
        timer2OC2ADutyCycle(RGB[2]);    // PWM signal na kanalu OC2A - PB3

        lcdClrScr();    // ocisti LCD displej
        lcdHome();      // postavljanje pokazivaca LCD displeja
        lcdprintf("R%u G%u B%u", RGB[0], RGB[1], RGB[2]);   // ispis

        _delay_ms(250);

    }
}
```

Prevedite datoteku `vjezba136a.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Prilikom kombinacije R=10, G=10 i B=10, RGB dioda treba svijetliti slabim intenzitetom i

moguće je vidjeti zasebno R, G i B svjetla unutar diode. Za potpunu provjeru ispravnosti napisanog koda, postavite u 3 iteracije snimanja programa na mikroupravljač sljedeće kombinacije:

- R na 100% intenziteta - R=100; G=0; B=0,
- G na 100% intenziteta - R=0; G=100; B=0 i
- B na 100% intenziteta - R=0; G=0; B=100.

Nakon što ste testirali vježbu, zatvorite datoteku `vjezba136a.cpp` i onemogućite prevođenje ove datoteke.

Potrebno je proširiti programski kod iz prethodnog primjera, na način da će se sada vrijednosti R, G i B komponenata RGB dioda generirati pomoću rotacijskog enkodera. Potrebno je izmijeniti ispis na LCD displeju. U prvi redak ispišite postotak širine visokog stanja *PWM* signala - *Duty Cycle*, na primjer `PWM: 53/100%`. Drugi redak LCD displeja treba prikazivati odabrani kanal RGB diode za koji se u prvom retku trenutno namješta vrijednost *Duty Cycle* od 0 do 100. Kada se pritisne tipkalo na rotacijskom enkoderu, vrijednost *Duty Cycle* treba pohraniti u registar za postavaljanje određenog kanala *PWM* signala. Također, pritisak tipkala treba prebaciti izbornik na LCD displeju na sljedeći kanal. Shema spajanja RGB diode na mikroupravljač prikazana je na slici 13.19.

U projektnom stablu otvorite datoteku `vjezba136b.cpp`. Omogućite samo prevođenje datoteke `vjezba136b.cpp`. Početni sadržaj datoteke `vjezba136b.cpp` prikazan je programskim kodom 13.43.

Programski kod 13.43: Početni sadržaj datoteke `vjezba136b.cpp`

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "Peripheral/rotEnc.h"
#include "Peripheral/rgbLED.h"
#include <avr/io.h>
#include <util/delay.h>

char rgbCH = 'R';        // globalna varijabla trenutne vrijednosti RGB
uint8_t encValue = 0;    // globalna varijabla trenutne vrijednosti RE

ISR(INT0_vect){      // prekidna rutina za azuriranje vrijednosti RE
    encValue = rotEncUPDATEnREAD(encValue);     // azuriranje \& citanje RE
}

void init() {        // inicijalizacija mikroracunala

    // inicijalizacija rotacijskog enkodera
    // inicijalizacija RGB LE diode
    // inicijalizacija LCD displeja
    // omogucavanje globalnih prekidnih rutina

    // inicijalno postavljanje ispisa na LCD

    // prikaz vrijednosti RE

    // prikaz kanala RGB diode

}
```

```
int main(void) {

    init();      // poziv funkcije za inicijalizaciju mikroupravljaca

    uint8_t RGB[3] = {0, 0, 0};      // polje trenutne vrijednosti RGB kanala

    while (1) {

        if(rotEncBTN()) // tipkalo padajuci detekcija - promjeni RGB
        {
            switch(rgbCH){
                case 'R':
                    // spremi vrijednost RE kao R kanal RGB
                    // sljedeci kanal RGB diode
                    break;

                case 'G':
                    // spremi vrijednost RE - G kanal RGB
                    // sljedeci kanal RGB diode
                    break;

                case 'B':
                    // spremi vrijednost RE - B kanal RGB
                    // sljedeci kanal RGB diode
                    break;

                default:
                    asm("nop");      // no operation
            }

            while(!get_pin(PIND, PD5));      // cekaj otpustastanje tipkala
        }

        // azuriranje vrijednosti kanala RGB

        // ocisti LCD displej
        // postavljanje pokazivaca displeja na koordinate 1, 1
        // prikaz vrijednosti dobivene RE
        // postavljanje pokazivaca displeja na koordinate 2, 1
        // prikaz trenutnog kanala RGB diode

        _delay_ms(100);
    }
}
```

U programskom kodu 13.43 uljučene su u prevođenje biblioteke koje će omogućiti jednostavniju izradu programskog koda za upravljanje RGB diodom. U prevođenje je uključena biblioteka `rotEnc.h` koja sadrži korisne funkcije za upotrebu rotacijskog enkodera. Sljedeća biblioteka je `rgbLED.h` koja se koristi za jednostavniju izradu programskog koda koji upravlja RGB diodom, a sadrži sljedeće funkcije `rgbINIT()` i `rgbCHwrite()`.

Definicija funkcije za inicijalizaciju RGB diode pisana je bez funkcija iz drugih biblioteka. Svrha funkcije je ista kao i dio programskog koda vezanog za inicijalizaciju *Timer/Counter* sklopova i pripadajućih izlaza u inicijalizacijskoj funkciji iz prethodnog primjera. Inicijalizirani su sklopovi *Timer/Counter1* i *Timer/Counter2*. Djelitelji frekvencije su postavljeni na 64. Oba sklopa rade u *Phase Correct PWM* načinu rada. Omogućeni su fizički izlazi navedenih sklopova kao *PWM* izlazni pinovi `OC1A`, `OC1B` i `OC2A`. Potpuna definicija funkcije `rgbINIT()` prikazana je programskim kodom 13.44.

Programski kod 13.44: Definicija funkcije `rgbINIT()`

```
void rgbINIT()      // funkcija za inicijalizaciju RGB LED diode
{
    // timer 1 - djelitelj frekvencije postavljen na 64
    TCCR1B \&= ~((1 << CS12) | (1 << CS11) | (1 << CS10));
    TCCR1B |= ((0 << CS12) | (1 << CS11) | (1 << CS10));

    // timer 2 - djelitelj frekvencije postavljen na 64
    TCCR2B \&= ~((1 << CS22) | (1 << CS21) | (1 << CS20));
    TCCR2B |= ((1 << CS22) | (0 << CS21) | (0 << CS20));

    // timer 1 - phase correct nacin rada - 8-bit counting
    TCCR1A \&= ~((1 << WGM11) | (1 << WGM10));
    TCCR1B \&= ~((1 << WGM13) | (1 << WGM12));
    TCCR1A \&= ~((1 << COM1A1)|(1 << COM1A0)|(1 << COM1B1)|(1 << COM1B0));
    TCCR1A |= (0 << WGM11) | (1 << WGM10);
    TCCR1B |= (0 << WGM13) | (0 << WGM12);

    // timer 2 - phase correct nacin rada - 8-bit counting
    TCCR2A \&= ~((1 << WGM21) | (1 << WGM20));
    TCCR2B \&= ~(1 << WGM22);
    TCCR2A \&= ~((1 << COM2A1)|(1 << COM2A0)|(1 << COM2B1)|(1 << COM2B0));
    TCCR2A |= (0 << WGM21) | (1 << WGM20);
    TCCR2B |= (0 << WGM22);

    //timer 1 - OC1A neinvertirajuci PWM izlaz omogucen
    TCCR1A \&= ~((1 << COM1A1) | (1 << COM1A0));
    TCCR1A |= (1 << COM1A1) | (0 << COM1A0);

    //timer 1 - OC1B neinvertirajuci PWM izlaz omogucen
    TCCR1A \&= ~((1 << COM1B1) | (1 << COM1B0));
    TCCR1A |= (1 << COM1B1) | (0 << COM1B0);

    //timer 2 - OC2A neinvertirajuci PWM izlaz omogucen
    TCCR2A \&= ~((1 << COM2A1) | (1 << COM2A0));
    TCCR2A |= (1 << COM2A1) | (0 << COM2A0);

    DDRB |= (1 << PB1) | (1 << PB2) | (1 << PB3);   // OC1A, OC1B, OC2A
}
```

Definicija funkcije za ažuriranje vrijednosti visokog stanja *PWM* kanala RGB diode `rgbCHwrite()` prikazana je programskim kodom 13.45.

Programski kod 13.45: Definicija funkcije `rgbCHwrite()`

```
void rgbCHwrite(uint8_t chR, uint8_t chG, uint8_t chB)
{
    // timer 1 - OC1A - Duty Cycle skaliran s 100 na 255
    OCR1A = chR * 255 / 100;

    /// timer 1 - OC1B - Duty Cycle skaliran s 100 na 255
    OCR1B = chG * 255 / 100;

    /// timer 2 - OC2A - Duty Cycle skaliran s 100 na 255
    OCR2A = chB * 255 / 100;
}
```

U funkciji `rgbCHwrite()` postavljene su vrijednosti registara `OC1A`, `OC1B` i `OC2A`. Argumenti funkcije mogu poprimiti vrijednosti od 0 do 100. Postavljanjem 8-bitnih vrijednosti u navedene registre, određuje se širina visokog stanja *PWM* signala na izlaznim kanalima kojima odgovoraju fizički pinovi mikroupravljača `PB1`, `PB2` i `PB3`. Za sva tri kanala RGB diode napravljeno je jednostavno skaliranje *Duty Cycle* vrijednosti. Uz pomoć enkodera zadaje se neka vrijednost

u intervalu od 0 do 100, koju je potrebno skalirati na interval od 0 do 255, jer je maksimalna vrijednost 8-bitnog registra širine $2^8 - 1 = 255$.

Ažuriranje i čitanje vrijednosti enkodera te pohrana iste u globalnu varijablu, napisano je u prekidnoj rutini zbog asinkrone pojave zahtjeva za promjenu vrijednosti kanala RGB diode. Korištena je funkcija iz biblioteke `rotEnc.h`, koja je uključenja u prevođenje zbog jednostavnosti korištenja rotacijskog enkodera te sadrži sljedeće funkcije:

- `rotEncINIT();`
- `rotEncBTN();`
- `rotEncDIR();`
- `rotEncUPDATEnREAD();`

U funkciji `rotEncINIT()` omogućen je vanjski ulazni signal na pinu `PD2`, koji odgovara vanjskom prekidu `INT0`. Resetirane su na 0 vrijednosti kontrolnog registra `EICRA` na mjestima bitova `ISC00` i `ISC01`. Isti bitovi postavljeni su u binarnu kombinaciju koja označava da će vanjska prekidna rutina biti aktivirana na padajući brid signala na pinu `PD2`. S obzirom da je rotacijski enkoder spojen na pinove `PD2`, `PD4` i `PD5`, ti isti pinovi definirani su kao ulazni te su uključeni pripadajući pritezni otpornici. Definicija funkcije `rotEncINIT()` prikazana je programskim kodom 13.46.

Programski kod 13.46: Definicija funkcije `rotEncINIT()`

```
void rotEncINIT()
{
    EIMSK |= (1 << INT0);   // INT0 na pinu PD2 je omogucen

    EICRA &= ~((1 << ISC01) | (1 << ISC00));

    // padajuci brid signala na pinu PD2 ce aktivirati prekidnu rutinu
    EICRA |= (1 << ISC01) | (0 << ISC00);

    DDRD &= ~(1 << PD2);    // inicijalizacija pina PD2 kao ulaznog pina
    DDRD &= ~(1 << PD4);    // inicijalizacija pina PD4 kao ulaznog pina
    DDRD &= ~(1 << PD5);    // inicijalizacija pina PD5 kao ulaznog pina

    PORTD |= (1 << PD2);    // ukljucivanje pull-up otpornika za pin PD2
    PORTD |= (1 << PD4);    // ukljucivanje pull-up otpornika za pin PD4
    PORTD |= (1 << PD5);    // ukljucivanje pull-up otpornika za pin PD5
}
```

U funkciji `rotEncBTN()` provjera se stanje tipkala na rotacijskom enkoderu, na pojednostavljen način. Definicija funkcije je prikazana programskim kodom 13.47.

Programski kod 13.47: Definicija funkcije `rotEncBTN()`

```
bool rotEncBTN()    // fnc za detekciju stanja tipkala rotacijskog enkodera
{
    if(isFallingEdge(D5))   // na padajuci brid pina PD5 vrati bool = 1
        return 1;
    else
        return 0;           // inace bool = 0
}
```

Smjer vrtnje rotacijskog enkodera je bitna informacija za povećanje ili umanjenje referentne vrijednosti koju planiramo koristiti za namještanje postavki nekog sklopa. Pomoću kombinacije padajućih i rastućih bridova signala, dobivenih analizom stanja A i B kanala rotacijskog enkodera, odlučuje se smjer vrtnje. Neposredno nakon pojave padajućeg brida signala na A kanalu

rotacijskog enkodera, potrebno je promotriti stanje B kanala te zaključiti smjer vrtnje. Stanje B kanala promatra se na fizičkom pinu mikroupravljača `PD3`, na koji je spojen izlazni B kanal rotacijskog enkodera. Definicija funkcije `rotEncDIR()` koja odlučuje o smjeru vrtnje rotacijskog enkodera prikazana je programskim kodom 13.48.

Programski kod 13.48: Definicija funkcije `rotEncDIR()`

```
bool rotEncDIR()    // fnc - detekcija smjera rotacijskog enkodera
{
    if (get_pin(PIND, PD4))      // ako je kanal B enkodera u visokom stanju
        return 1;                // smjer je u smjeru kazaljke na satu
    else
        return 0;                // smjer suprotan kazaljci na satu
}
```

Ažuriranje, čitanje i pohrana trenutne vrijednosti upravljane rotacijskim enkoderom, pojednostavljeno je funkcijom `rotEncUPDATEnREAD()`. Navedena funkcija provjerava koji je smjer vrtnje rotacijskog enkodera trenutan i na temelju tog podatka odlučuje da li će se varijabla za pohranu trenutne vrijednosti rotacijskog enkodera uvećati ili umanjiti za neki iznos (u ovoj vježbi je to korak iznosa 10). Funkcija će vratiti 8-bitnu trenutnu vrijednost rotacijskog enkodera. Definicija funkcije prikazana je programskim kodom 13.49:

Programski kod 13.49: Definicija funkcije `rotEncUPDATEnREAD()`

```
// azuriranje i citanje vrijednosti RE u varijablu
uint8_t rotEncUPDATEnREAD(uint8_t enc)
{
    if (rotEncDIR() == 1)   // smjer u smjeru kazaljke na satu
    {
        // trenutna vrijednost enkodera < najvece moguce
        if (enc < ENCODER_HIGH_THRESHOLD)
            enc = enc + 10;     // uvecaj trenutnu vrijednost za 1
    }

    else
    {
        // ako je trenutna vrijednost enkodera veca od najmanje moguce
        if (enc > ENCODER_LOW_THRESHOLD)
            enc = enc - 10; // smanji trenutnu vrijednost za 1
    }

    return enc;
}
```

Inicijalizacijska funkcija `init()` uključuje sljedeće:

- Inicijalizacija rotacijskog enkodera - funkcija `rotEncINIT()`,
- Inicijalizacija RGB diode - funkcija `rgbINIT()` i
- Inicijalizacija LCD displeja - funkcija `lcdInit()`.

Pomoću funkcije `interruptEnable()` omogućene su globalne prekidne rutine. Postavljen je inicijalni ispis na LCD displeju u zadanom formatu.

U glavnoj programskoj funkciji `main()` pozvana je prethodno opisana funkcija `init()` za inicijalizaciju mikroupravljača. Nakon toga deklarirano je cjelobrojno 8-bitno polje bez predznaka, u koje će biti pohranjene vrijednosti R, G i B komponenata RGB diode. U glavnoj beskonačnoj petlji `while()` provjerava se periodično da li je pritisnuto tipkalo rotacijskog enkodera. Ukoliko jest, promijeniti će se izabrani trenutni kanal RGB diode za koju se postavlja vrijednost *Duty Cycle* pripadnog *PWM* signala. Nakon provjere stanja tipkala rotacijskog

enkodera, ažurira se vrijednost *PWM* signala RGB diode pomoću funkcije `rgbCHwrite()`. Nakon ažuriranja RGB diode, ispisuju se vrijednosti na LCD displeju u prethodno opisanom ispisnom formatu. Brzina ažuriranja beskonačne programske petlje postavljena je pomoću funkcije `_delay_ms(100)` na 100 ms, iz potrebe usporavanja izvršavanja programa zbog oku prebrzog ažuriranja LCD displeja.

Da bi bilo moguće koristiti rotacijski enkoder, potrebno je postaviti kratkospojnik `JP2` tako da kratko spaja srednja 3 pina i gornja 3 pina nazvana `ENK`.

Potpuni sadržaj datoteke `vjezba136b.cpp` gdje je inicijaliziran mikroupravljač s promjenjivim algoritmom za postavljanje intenziteta svjetla kanala RGB diode pomoću rotacijskog enkodera, prikazan je programskim kodom 13.50.

Programski kod 13.50: Potpuni sadržaj datoteke `vjezba136b.cpp`

```
#include "AVR/avrlib.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "Peripheral/rotEnc.h"
#include "Peripheral/rgbLED.h"
#include <avr/io.h>
#include <util/delay.h>

char rgbCH = 'R';        // globalna varijabla trenutne vrijednosti RGB
uint8_t encValue = 0;    // globalna varijabla trenutne vrijednosti RE

ISR(INT0_vect){      // prekidna rutina za azuriranje vrijednosti RE
    encValue = rotEncUPDATEnREAD(encValue);     // azuriranje & citanje RE
}

void init() {        // inicijalizacija mikroracunala

    rotEncINIT();    // inicijalizacija rotacijskog enkodera
    rgbINIT();       // inicijalizacija RGB LE diode
    lcdInit();       // inicijalizacija LCD displeja
    interruptEnable();  // omogucavanje globalnih prekidnih rutina

    lcdClrScr();     // inicijalno postavljanje ispisa na LCD
    lcdHome();
    lcdprintf("PWM: %d/100%c", encValue, 37);   // prikaz vrijednosti RE
    lcdGotoXY(2,1);
    lcdprintf("RGB CH: %c", rgbCH);     // prikaz kanala RGB diode

}

int main(void) {

    init();     // poziv funkcije za inicijalizaciju mikroupravljaca

    uint8_t RGB[3] = {0, 0, 0};     // polje trenutne vrijednosti RGB kanala

    while (1) {

        if(rotEncBTN()) // tipkalo padajuci detekcija - promjeni RGB
        {
            switch(rgbCH){
                case 'R':
                    RGB[0] = encValue;  // spremi vrijednost RE - R kanal RGB
                    rgbCH = 'G';        // sljedeci kanal RGB diode
                    break;

                case 'G':
```

```
                    RGB[1] = encValue;  // spremi vrijednost RE - G kanal RGB
                    rgbCH = 'B';        // sljedeci kanal RGB diode
                    break;

                case 'B':
                    RGB[2] = encValue;  // spremi vrijednost RE - B kanal RGB
                    rgbCH = 'R';        // sljedeci kanal RGB diode
                    break;

                default:
                    asm("nop");     // no operation
            }

            while(!get_pin(PIND, PD5));     // cekaj otpustastanje tipkala
        }

        rgbCHwrite(RGB[0], RGB[1], RGB[2]); // azuriranje vrijednosti RGB

        lcdClrScr();    // ocisti LCD displej
        lcdHome();      // postavljanje pokazivaca LCD na koordinate 1, 1
        lcdprintf("PWM: %d/100%c", encValue, 37);   // prikaz vrijednosti RE
        lcdGotoXY(2,1);     // postavljanje pokazivaca LCD na koordinate 2, 1
        lcdprintf("RGB CH: %c", rgbCH);     // prikaz trenutnog kanala RGB

        _delay_ms(100);
    }
}
```

Prevedite datoteku `vjezba136b.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Postavite vrijednost R, G i B komponenata RGB diode pomoću rotacijskog enkodera. Za potpunu provjeru ispravnosti napisanog koda, postavite u rotacijskim enkoderom sljedeće kombinacije:

- R na 100% intenziteta - R=100; G=0; B=0,
- G na 100% intenziteta - R=0; G=100; B=0 i
- B na 100% intenziteta - R=0; G=0; B=100.

Nakon što ste testirali vježbu, zatvorite datoteku `vjezba136b.cpp` i onemogućite prevođenje ove datoteke.

Potrebno je izmijeniti programski kod iz prethodnog primjera, na način da će se sada vrijednosti R, G i B komponenata RGB dioda izdvojiti iz komplementne poruke dobivene putem serijske *UART* komunikacije. Poruka treba biti poslana iz programskog alata *Tera Term* na osobnom računalu. Potrebno je izmijeniti ispis na LCD displeju. U prvi redak ispišite dobivenu poruku u izvornom obliku. U drugi redak LCD displeja ispišite izdvojene vrijednosti iz poruke potrebne za postavljanje R, G i B komponenti RGB diode. Shema spajanja RGB diode na mikroupravljač prikazana je na slici 13.19.

U projektnom stablu otvorite datoteku `vjezba136c.cpp`. Omogućite samo prevođenje datoteke `vjezba136c.cpp`. Početni sadržaj datoteke `vjezba136c.cpp` prikazan je programskim kodom 13.51.

Programski kod 13.51: Početni sadržaj datoteke `vjezba136c.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"
#include "Peripheral/rgbLED.h"
```

```
#include "UART/uart.h"
#include <avr/io.h>
#include <util/delay.h>

void init() {       // inicijalizacija mikroracunala

    // inicijalizacija RGB LE diode
    // inicijalizacija LCD displeja
    // inicijalizacija serijske komunikacije
    // globalne prekidne rutine

}

int main(void) {

    init();     // poziv inicijalizacije mikroupravljaca

    uint8_t RGB[3] = {0, 0, 0};     // polje vrijednosti RGB kanala

    while (1) {

    if(uartIsMessageRecived()) {        // ako je pristigla poruka

        lcdClrScr();        // ocisti LCD
        lcdHome();      // pokazivac LCD postavi koordinate 1,1
        lcdprintf("%s\n", uartBuffer);  // ispisi poruku na LCD izvorno

        if(uartBuffer[0] == 'L') {  // prvi znak poruke L ->  RGB

            RGB[0] = 0;     // postavi R kanal RGB diode na 0
            RGB[1] = 0;     // postavi G kanal RGB diode na 0
            RGB[2] = 0;     // postavi B kanal RGB diode na 0

            // izdvajanje informacija poruke oblika npr. L25-50-75e
            // (R kanal: 25%, G kanal: 50%, B kanal: 75%)

            for(uint8_t i = 1, j = 0; uartBuffer[i] != 'e'; i++) {
                if (uartBuffer[i] != '-') {     // znak za razdvajanje
                    // spajanje znamenki
                    RGB[j] = RGB[j] * 10  + (uartBuffer[i] - 48);
                }
                else {
                    j++;    // iduca vrijednost
                }
            }
                    // azuriranje vrijednosti kanala RGB
            rgbCHwrite(RGB[0], RGB[1], RGB[2]);
                    // ispis izdvojenih R, G i B
            lcdprintf("%u %u %u e", RGB[0], RGB[1], RGB[2]);

            _delay_ms(100);
        }
        }
    }
}
```

U programskom kodu 13.51 u funkciji `init()` inicijalizirana je RGB dioda pomoću funkcije `rgbINIT()`, LCD displej pomoću funkcije `lcdInit()` i serijska *UART* komunikacija pomoću funkcije `uartInit(9600)`, kojoj je i proslijeđen argument 9600 koji označava brzinu prijenosa podataka putem serijske komunikacije (engl. *Baud Rate*), iznosa 9600 bps. Omogućene su i globalne prekidne rutine. Potpuna definicija funkcije `init()` prikazana je programskim kodom 13.52. Osvježite funkciju `init()` u datoteci `vjezba136c.cpp` kako bi bila istovjetna programskom

kodu 13.52.

Programski kod 13.52: Potpuna definicija funkcije `init()`

```
void init() {        // inicijalizacija mikroracunala

    rgbINIT();       // inicijalizacija RGB LE diode
    lcdInit();       // inicijalizacija LCD displeja
    uartInit(9600); // inicijalizacija serijske komunikacije
    interruptEnable();  // globalne prekidne rutine

}
```

U glavnoj programskoj funkciji `main()` pozvana je prethodno opisana funkcija `init()` za inicijalizaciju mikroupravljača. Nakon toga deklarirano je cjelobrojno 8-bitno polje bez predznaka, u koje će biti pohranjene vrijednosti R, G i B komponenata RGB diode. U glavnoj beskonačnoj programskoj petlji `while()`, neprekidno se provjerava da li je pristigla poruka putem serijske *UART* komunikacije, koristeći funkciju `uartIsMessageRecived()`. Ukoliko je poruka pristigla i u cijelosti je pročitana, prosljeđuje se na daljnju obradu, odnosno izdvajanje korisnih informacija. Slijedi ispisivanje poruke na prvi redak LCD displeja. Ukoliko je prvi znak pristigle poruke slovo `L`, tada je poruka namijenjena za postavljanje vrijednosti RGB diode. Potrebno je pohraniti znakove iz poruke u određene lokalne varijable, za korištenje tijekom daljnjeg programskog koda. Pomoću jednostavne *for* petlje čitaju se svi znakovi, sve dok petlja ne dođe do znaka, odnosno slova `e` koji označava kraj poruke. Tijekom izvršavanja *for* petlje, znak crtica `-` ima ulogu znaka za razdvajanje (engl. *delimiter*) i označava novi element u poruci, odnosno novu vrijednost komponente RGB diode. Svaki element lokalnog polja RGB predstavlja širinu visokog stanja *PWM* signala za pojedini kanal RGB diode. Ako poruka koja stigne putem serijske komunikacije ima oblik `L25-50-75e`, tada je iznos DC(R)=25%, DC(G)=50% i DC(B)=75%. Nakon uspješnog čitanja i izdvajanja informacija iz poruke, iste se dohvaćaju iz polja podataka. Za postavljanje vrijednosti širine visokog stanja *PWM* signala koristi se već poznata, ranije opisana funkcija `rgbCHwrite()`. Potpuno napisana glavna programska funkcija `main()` prikazana je programskim kodom 13.53. Osvježite funkciju `main()` u datoteci `vjezba136ac.cpp` kako bi bila istovjetna programskom kodu 13.53.

Programski kod 13.53: Potpuna glavna programska funkcija `main()`

```
int main(void) {

    init();     // poziv inicijalizacije mikroupravljaca

    uint8_t RGB[3] = {0, 0, 0};     // polje vrijednosti RGB kanala

    while (1) {

        if(uartIsMessageRecived()) {        // ako je pristigla poruka

            lcdClrScr();        // ocisti LCD
            lcdHome();      // pokazivac LCD postavi koordinate 1,1
            lcdprintf("%s\n", uartBuffer);  // ispisi poruku na LCD izvorno

            if(uartBuffer[0] == 'L') {  // prvi znak L -> RGB

                RGB[0] = 0;     // postavi R kanal RGB diode na 0
                RGB[1] = 0;     // postavi G kanal RGB diode na 0
                RGB[2] = 0;     // postavi B kanal RGB diode na 0

                // izdvajanje informacija iz poruke npr. L25-50-75e
                // (R kanal: 25%, G kanal: 50%, B kanal: 75%)

                for(uint8_t i = 1, j = 0; uartBuffer[i] != 'e'; i++) {
```

```
                    if (uartBuffer[i] != '-') {     // znak za razdvajanje
                        // spajanje znamenki
                        RGB[j] = RGB[j] * 10  + (uartBuffer[i] - 48);
                    }
                    else {
                        j++;    // iduca vrijednost
                    }
                }
                        // azuriranje vrijednosti kanala RGB
                rgbCHwrite(RGB[0], RGB[1], RGB[2]);
                        // ispis izdvojenih R, G i B
                lcdprintf("%u %u %u e", RGB[0], RGB[1], RGB[2]);

                _delay_ms(100);
            }
        }
    }
}
```

Prevedite datoteku `vjezba136c.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačom ATmega328P. Postavite vrijednost R, G i B komponenata RGB diode pomoću serijske komunikacije. Za potpunu provjeru ispravnosti napisanog koda, postavite sljedeće kombinacije pomoću poruka poslanih serijskom komunikacijom:

- R na 100% intenziteta - `L100-0-0e`,
- G na 100% intenziteta - `L0-100-0e` i
- B na 100% intenziteta - `L0-0-100e`.

Nakon što ste testirali vježbu, zatvorite datoteku `vjezba136c.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Microchip Studio*.

Poglavlje 14

Ostale značajke mikroupravljača

U ovom poglavlju kratko ćemo proučiti ostale značajke mikroupravljača ATmega328P:

- EEPROM memorija,
- Whatchdog tajmer,
- Sleep modovi rada i upravljanje potrošnjom energije,
- Analogni komparator,
- I2C komunikacija.

Za svaku vježbu u ovom poglavlju dat ćemo konačno programsko rješenje mikroupravljača i ukratko ga objasniti.

S mrežne stranice `https://vub.hr/atmega328p` skinite datoteku `Ostalo.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati `Vaše Ime i Prezime` ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Pero Perić, datoteka koju ćete stvoriti zvat će se `Pero Peric`. Datoteku `Ostalo.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `Mikroupravljaci.atsln` u datoteci `\\Vanjski prekidi\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju `Ostale značajke mikroupravljača`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

14.1 EEPROM memorija

SRAM podatkovna memorija ima svojstvo da gubitkom napajanja gubi sadržaj zapisan u njoj. EEPROM memorija (engl. *Electrically Erasable Programmable Read-Only Memory*) vrsta je permanentne nepromjenjive memorije koja za čuvanje pohranjenih podataka ne treba električno napajanje. Tipična primjena EEPROM memorije jest za čuvanje kalibracijskih parametara sustava, regulacijskih parametara sustava i općenito parametara koji se ne smiju izgubiti nestankom napajanja, a istovremeno ih je povremeno moguće promijeniti. Nedostatak EEPROM memorije je broj ciklusa pisanja i čitanja koji iznosi 100 000. Pri korištenju EEPROM memorije potrebno je voditi računa da se pristup memorije ne odvija neprestano jer bi se na takav način EEPROM memorija uništila. Mikroupravljač ATmega328P ima 1 kB EEPROM memorije koja je bajtovno orjentirana.

Funkcije kojima se pristupa EEPROM memoriji deklarirane su u zaglavlju `avr/eeprom.h`. Ove funkcije su napisane od strane proizvođača mikroupravljača ATmega328P i možemo ih

koristiti u obliku kako su definirane. Za čitanje podataka iz EEPROM memorije i pisanje podataka u EEPROM memoriju koriste se sljedeće funkcije:

- `uint8_t eeprom_read_byte(const uint8_t * adr)` - funkcija kojom se čita cjelobrojni podatak širine 8 bitova s adrese `adr`. Funkcija vraća podatkovni tip `uint8_t`, no moguće je pročitati i podatkovni tip `int8_t` ako je tako pohranjen u EEPROM memoriju. Funkcija prima `uint8_t` pokazivač na adresu `adr`.

- `uint16_t eeprom_read_word(const uint16_t * adr)` - funkcija kojom se čita cjelobrojni podatak širine 16 bitova s adrese `adr`. Funkcija vraća podatkovni tip `uint16_t`, no moguće je pročitati i podatkovni tip `int16_t` ako je tako pohranjen u EEPROM memoriju. Funkcija prima `uint16_t` pokazivač na adresu `adr`.

- `uint32_t eeprom_read_dword(const uint32_t * adr)` - funkcija kojom se čita cjelobrojni podatak širine 32 bitova s adrese `adr`. Funkcija vraća podatkovni tip `uint32_t`, no moguće je pročitati i podatkovni tip `int32_t` ako je tako pohranjen u EEPROM memoriju. Funkcija prima `uint32_t` pokazivač na adresu `adr`.

- `float eeprom_read_float(const float * adr)` - funkcija kojom se čita realan podatak širine 32 bitova s adrese `adr`. Funkcija vraća podatkovni tip `float`. Funkcija prima `float` pokazivač na adresu `adr`.

- `void eeprom_write_byte(uint8_t * adr, uint8_t value)` - funkcija kojom se zapisuje cjelobrojni podatak `value` širine 8 bitova na adresu `adr`. Funkcija prima `uint8_t` pokazivač na adresu `adr` i podatak `value` tipa `uint8_t`.

- `void eeprom_write_word(uint16_t * adr, uint16_t value)` - funkcija kojom se zapisuje cjelobrojni podatak `value` širine 16 bitova na adresu `adr`. Funkcija prima `uint16_t` pokazivač na adresu `adr` i podatak `value` tipa `uint16_t`.

- `void eeprom_write_dword(uint32_t * adr, uint32_t value)` - funkcija kojom se zapisuje cjelobrojni podatak `value` širine 32 bitova na adresu `adr`. Funkcija prima `uint32_t` pokazivač na adresu `adr` i podatak`value` tipa `uint32_t`.

- `void eeprom_write_float(float * adr, float value)` - funkcija kojom se zapisuje realan podatak `value` širine 32 bitova na adresu `adr`. Funkcija prima `float` pokazivač na adresu `adr` i podatak `value` tipa `float`.

EEPROM memorija je, kako smo naveli, bajtovno orijentirana. Prema tome, ako u EEPROM memoriju spremamo podatkovni tip `uint16_t` na adresu 10, podatak će zauzeti adrese 10 i 11 jer podatkovni tip `uint16_t` zauzima 2 bajta memorije.

U nastavku ćemo prikazati nekoliko primjere korištenja navedenih funkcija:

- `eeprom_write_byte((uint8_t*)0, 100);` - u EEPROM memoriju na adresu 0 pohranjuje se broj 100 koji je 8-bitni cijeli broj bez predznaka. Ovaj podatak zauzima jedan bajt EEPROM memorije pa je sljedeća slobodna adresa 1.

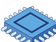
- `eeprom_write_word((uint16_t*)1, -20000);` - u EEPROM memoriju na adresu 1 (prva slobodna adresa) pohranjuje se broj -20000 koji je 16-bitni cijeli broj s predznakom. Ovaj podatak zauzima dva bajta EEPROM memorije (adrese 1 i 2) pa je sljedeća slobodna adresa 3.

- `eeprom_write_dword((uint32_t*)3, 200000);` - u EEPROM memoriju na adresu 3 (prva slobodna adresa) pohranjuje se broj 200000 koji je 32-bitni cijeli broj bez predznaka. Ovaj

podatak zauzima četiri bajta EEPROM memorije (adrese 3, 4, 5 i 6) pa je sljedeća slobodna adresa 7.

- `eeprom_write_float((float*)7, 3.14159);` - u EEPROM memoriju na adresu 7 (prva slobodna adresa) pohranjuje se broj 3.14159 koji je 32-bitni realan broj. Ovaj podatak zauzima četiri bajta EEPROM memorije (adrese 7, 8, 9 i 10) pa je sljedeća slobodna adresa 11.
- `uint8_t data8 = eeprom_read_byte((uint8_t*)0);` - s adrese 0 u EEPROM memoriji čita se 8-bitni cijeli broj bez predznaka. Sadržaj varijable `data8` bit će jednak 100 sukladno prethodnim funkcijama.
- `uint16_t data16 = eeprom_read_word((uint16_t*)1);` - s adrese 1 u EEPROM memoriji čita se 16-bitni cijeli broj s predznakom. Sadržaj varijable `data16` bit će jednak -20000 sukladno prethodnim funkcijama.
- `uint32_t data32 = eeprom_read_dword((uint32_t*)3);` - s adrese 3 u EEPROM memoriji čita se 32-bitni cijeli broj bez predznaka. Sadržaj varijable `data32` bit će jednak 200000 sukladno prethodnim funkcijama.
- `float real = eeprom_read_float((float*)7);` - s adrese 7 u EEPROM memoriji čita se 32-bitni realan broj. Sadržaj varijable `real` bit će jednak 3.14159 sukladno prethodnim funkcijama.

Primijetite da za pojedinu adresu, funkcije primaju pokazivače na tu adresu. Na primjer, ako želimo pročitati cjelobrojni podatak širine 16 bitova s adrese 5, funkciji `eeprom_read_word()` se prosljeđuje pokazivač na cjelobrojni tip podatka širine 16 bitova (`(uint16_t*)5`).

 Vježba 14.1

Napišite program koji će tri parametra sustava K_1 (`int8_t`), K_2 (`uint16_t`) i K_3 (`float`) čuvati u EEPROM memoriji. Početna vrijednost parametara jest $K_1 = -15$, $K_2 = 300$ i $K_3 = 14.5665$. Ove vrijednosti parametara potrebno je zapisati u EEPROM memoriju samo na početku vježbe, a nakon toga se dio programa koji zapisuje parametre u EEPROM mora obrisati te ponovno snimiti program na mikroupravljač. Pritiskom na tipkalo T1, vrijednost parametara se mijenja u $K_1 = -10$, $K_2 = 260$ i $K_3 = 11.709$. U programu realizirajte izračun sljedeće relacije:

$$x(adc0) = \frac{adc0}{K_2} K_3 + K_1 \tag{14.1}$$

Vrijednost $adc0$ u relaciji (14.1) je rezultat AD prevorbe na pinu ADC0. Pritiskom na tipkalo T2 izmjenjuje se ispis relacije (14.1) i parametara K_1, K_2 i K_3 na LCD displeju. Prema shemi na slici 5.2, tipkalo T1 spojeno je na digitalni ulaz PD4, a tipkalo T2 spojeno je na digitalni ulaz PD2 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1. Potenciometar je prema slici 7.2 spojen na pin ADC0 pomoću kratkospojnika `JP6` koji postavite između trnova 2 i 3.

U projektnom stablu otvorite datoteku `vjezba141.cpp`. Omogućite prevođenje datoteke `vjezba141.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba141.cpp` prikazan je programskim kodom 14.1.

Programski kod 14.1: Početni sadržaj datoteke `vjezba141.cpp`

```
#include "AVR/avrlib.h"
```

```
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include <avr/eeprom.h>

bool parametriUpdate = false;
bool lcdUpdate = true;
// lcdPrikaz = false - parametri
// lcdPrikaz = true - formula
bool lcdPrikaz = true;

// prekidna rutina za PCINT2
ISR(PCINT2_vect) {
    // detekcija promjene brida na portu D
    uint8_t PINDChanged = pcintPINDOld ^ PIND;
    // ako se dogodio brid na pinu PD2 (tipkalo T2)
    if (PINDChanged & (1 << PD2)) {
        // padajući brid (pritisnuto tipkalo)
        if (get_pin(PIND, PD2) == false) {
            lcdUpdate = true;
            lcdPrikaz = !lcdPrikaz;
        }
    }
    // ako se dogodio brid na pinu PD4 (tipkalo T1)
    if (PINDChanged & (1 << PD4)) {
        // padajući brid (pritisnuto tipkalo)
        if (get_pin(PIND, PD4) == false) {
            parametriUpdate = true;
        }
    }
    // osvježenje stare vrijednosti registra PIND
    pcintPINDOld = PIND;
}
// osvježenje EEPROM memorije
void eepromUpdate(int8_t K1, uint16_t K2, float K3) {
    eeprom_write_byte((uint8_t*)0, K1);
    eeprom_write_word((uint16_t*)1, K2);
    eeprom_write_float((float*)3, K3);
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    interruptEnable(); // globalno omogućenje prekida
    // pin PD2 konfiguriran kao ulaz + uključen pritezni otpornik
    pinMode(D2, INPUT_PULLUP);
    // pin PD4 konfiguriran kao ulaz + uključen pritezni otpornik
    pinMode(D4, INPUT_PULLUP);
    // omogući PCINT prekid na pinovima iz skupine PCINT23..16
    pcintEnable23to16();
    pcintPinEnable23to16(PCINT18); //omogući PCINT prekid na pinu PCINT18(PD2)
    pcintPinEnable23to16(PCINT20); //omogući PCINT prekid na pinu PCINT20(PD4)
    // spremanje inicijalnih vrijednosti registara PINB, PINC i PIND
    pcintInit();
    // inicijalne vrijednosti eeprom memorije
    eepromUpdate(-15, 300, 14.5665);
}

int main(void) {

    init(); // inicijalizacija mikroupravljača
    // parametri
    int8_t K1;
```

```
    uint16_t K2;
    float K3;

    uint16_t adc0, adc0Old = 0;
    float x; // rezultat formule
    // citanje parametara iz EEPROM memorije
    K1 = eeprom_read_byte((uint8_t*)0);
    K2 = eeprom_read_word((uint16_t*)1);
    K3 = eeprom_read_float((float*)3);

    while (1) {
        // AD pretvorba i izračun forumle
        adc0 = adcRead(ADC0);
        x = adc0 * K3 / K2 + K1;

        // omogući ispis ako se promjeni ADC0
        if (adc0Old != adc0) {
            if (lcdPrikaz) {
                lcdUpdate = true;
            }
            adc0Old = adc0;
        }
        // ispis na LCD
        if (lcdUpdate) {
            if (lcdPrikaz == false) {
                lcdClrScr();
                lcdprintf("K1=%d K2=%u\n", K1, K2);
                lcdprintf("K3=%.4f", K3);
                lcdUpdate = false;
            }
            else {
                lcdClrScr();
                lcdprintf("x(%d)=%.4f", adc0, x);
                lcdUpdate = false;
                _delay_ms(500);
            }

        }
        // novi parametri u EEPROM memoriji
        if (parametriUpdate) {
            K1 = -10;
            K2 = 260;
            K3 = 11.709;
            eepromUpdate(K1, K2, K3);
            parametriUpdate = false;
            lcdUpdate = true;
        }

    }
}
```

Prevedite datoteku `vjezba141.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Nakon toga izbrišite naredbu `eepromUpdate(-15, 300, 14.5665);` koja se nalazi u inicijalizacijskoj funkciji `init()`. Ponovno prevedite datoteku `vjezba141.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P.

Na početku glavne funkcije iz EEPROM memorije učitavaju se parametri K_1, K_2 i K_3 sljedećim naredbama:

- `K1 = eeprom_read_byte((uint8_t*)0);` - čitanje 8-bitnog cjelobrojnog podatka s adrese 0,

- `K2 = eeprom_read_word((uint16_t*)1);` - čitanje 16-bitnog cjelobrojnog podatka s adrese 1,
- `K3 = eeprom_read_float((float*)3);` - čitanje 32-bitnog realnog podatka s adrese 3.

Primijetite da parametri K_1, K_2 i K_3 nisu inicijalizirani nakon deklaracije. Kod prvog snimanja programa na mikroupravljač ATmega328P naredbom `eepromUpdate(-15, 300, 14.5665);` smo u EEPROM memoriju učitali vrijednosti parametara K_1, K_2 i K_3. Funkcija `eepromUpdate(int8_t K1, uint16_t K2, float K3)` prima tri argumenta koji se upisuju u EEPROM memoriju pomoću sljedećih naredaba:

- `eeprom_write_byte((uint8_t*)0, K1);` - zapisivanje parametra K_1 (8-bita, cijeli broj) na adresu 0,
- `eeprom_write_word((uint16_t*)1, K2);` - zapisivanje parametra K_2 (16-bita, cijeli broj) na adresu 1,
- `eeprom_write_float((float*)3, K3);` - zapisivanje parametra K_3 (32-bita, realan broj) na adresu 3.

Funkcije kojima se pristupa EEPROM memoriji deklarirane su u zaglavlju `avr/eeprom.h` koje je potrebno uključiti u programski kod naredbom `#include <avr/eeprom.h>`.

Kada se jednom podaci zapišu u EEPROM memoriju, oni tamo ostaju trajno dok se ne promjene. Nakon što smo obrisali naredbu `eepromUpdate(-15, 300, 14.5665);`, parametri K_1, K_2 i K_3 se i dalje učitavaju iz EEPROM memorije i imaju vrijednosti $K_1 = -15$, $K_2 = 300$ i $K_3 = 14.5665$.

Obrada tipkala T1 i T2 realizirana je pomoću PCINT prekida. Kada se pojavi padajući brid tipkala T2 (PD2) mijenja se stanje varijable `lcdPrikaz` kojom se u beskonačnoj `while` petlji izmjenjuje prikaz relacije (14.1) i parametra K_1, K_2 i K_3 uz uvjet da varijabla `lcdUpdate` ima vrijednost `true`. Varijabla `lcdUpdate` postavlja se u vrijednost `true` ako se pritisne tipkalo T2 ili ako se promijeni rezultat AD pretvorbe na pinu ADC0.

Promijenite poziciju potenciometra! Kada se promijeni rezultat AD pretvorbe na pinu ADC0, na LCD displeju se prikazuje izračun relacije (14.1) koji je proveden u beskonačnoj `while` petlji. Pritisnite tipkalo T2! Na LCD displeju se sada ispisuju parametri. Odspojite napajanje mikroupravljača ATmega328P, a nakon toga vratite napajanje. Ponovno pritisnite tipkalo T2! Na LCD displeju se i dalje ispisuju parametri koje smo učitali u EEPROM memoriju na početku vježbe.

Kada se pojavi padajući brid tipkala T1 (PD4) varijabla `parametriUpdate` postavlja se u vrijednost `true`. U beskonačnoj `while` petlji se pomoću varijable `parametriUpdate` u EEPROM memoriju postavljaju novi parametri sukladno zahtjevima vježbe. Pritisnite tipkalo T1, a zatim pritisnite tipkalo T2! Na LCD displeju se sada ispisuju novi parametri. Odspojite napajanje mikroupravljača ATmega328P, a nakon toga vratite napajanje. Ponovno pritisnite tipkalo T2! Na LCD displeju se i dalje ispisuju novi parametri koje smo osvježili u EEPROM memoriju. Dakle, nestankom napajanja mikroupravljača ATmega328P, podaci u EEPROM memoriji se ne brišu.

Važno je osigurati da se čitanje EEPROM memorije i pisanje u EEPROM memoriju događa asinkrono i vrlo rijetko kako bismo osigurali da ne prekoračimo 100 000 ciklusa pisanja i čitanja EEPROM memorije.

Zatvorite datoteku `vjezba141.cpp` i onemogućite prevođenje ove datoteke.

14.2 Watchdog tajmer

Watchdog tajmer se koristi za praćenje izvršavanja programskog koda na mikroupravljaču. Osnovna ideja ovog tajmera jest (ukoliko je omogućen) da broji unutar zadanog perioda vremena. Ako zadani period vremena istekne, *Watchdog* tajmer pokreće prekidnu rutinu (generira prekid) i/ili resetiranje programa u mikroupravljaču. Da bismo izbjegli resetiranje mikroupravljača, u programskom kodu brojanje *Watchdog* tajmer moramo vratiti (resetirati) na početak zadanog perioda vremena. Uloga *Watchdog* tajmera jest da se programski kod u beskonačnoj `while` petlji izvrši unutar zadanog vremena. Ukoliko se programski kod u beskonačnoj `while` petlji ne izvrši unutar zadanog vremena, pretpostavka jest da se program zaglavio što može uzrokovati neispravan rad mikroupravljača. Stoga će ga *Watchdog* tajmer resetirati ili će se pokrenuti prekidna rutina kojom će se eliminirati neispravan rad mikroupravljača.

Mikroupravljač ATmega328P ima mogućnost konfiguriranja *Watchdog* tajmera za:

- generiranje prekida,
- resetiranje mikroupravljača,
- generiranje prekida i resetiranje mikroupravljača.

Navedeni načini konfiguracije definiraju koji će se mehanizam pokrenuti kada vrijeme koje *Watchdog* tajmer broji istekne. Vremena nakon kojih *Watchdog* tajmer pokreće jedan od gore navedenih mehanizama mogu biti: 16 ms, 32 ms, 64 ms, 0.125 s, 0.25 s, 0.5 s, 1.0 s, 2.0 s, 4.0 s i 8.0 s. Ova vremena konfiguriraju se u registru `WDTCSR` prema tablici 10-3 u literaturi [2].

Osnovne makronaredbe kojima se konfigurira *Watchdog* tajmer nalaze se u zaglavlju `avr/wdt.h`. Ove makronaredbe su napisane od strane proizvođača mikroupravljača ATmega328P. Autor je napisao zaglavlje `watchdog.h` u kojima se nalaze funkcije s proširenim mogućnostima konfiguracije *Watchdog* tajmera u odnosu na makronaredbe koje se nalaze u zaglavlju `avr/wdt.h`. Funkcije koje su deklarirane u zaglavlju `watchdog.h` u programski kod uključuje se naredbom `#include "AVR/watchdog.h"`. Konstante za odabir vremena nakon kojeg će *Watchdog* tajmer pokrenuti resetiranje mikroupravljača i/ili prekid prikazane su programskim kodom 14.2, a nalaze se u zaglavlju `watchdog.h`.

Programski kod 14.2: Konstante za odabir vremena nakon kojeg će *Watchdog* tajmer pokrenuti resetiranje mikroupravljača i/ili prekid

```
// definicije vremena za watchdog - registar WDTCSR
#define WATCHDOG_TIME_16MS      ((0 << WDP3) |(0 << WDP2) | (0 << WDP1) | (0 << WDP0))
#define WATCHDOG_TIME_32MS      ((0 << WDP3) |(0 << WDP2) | (0 << WDP1) | (1 << WDP0))
#define WATCHDOG_TIME_64MS      ((0 << WDP3) |(0 << WDP2) | (1 << WDP1) | (0 << WDP0))
#define WATCHDOG_TIME_125MS     ((0 << WDP3) |(0 << WDP2) | (1 << WDP1) | (1 << WDP0))
#define WATCHDOG_TIME_250MS     ((0 << WDP3) |(1 << WDP2) | (0 << WDP1) | (0 << WDP0))
#define WATCHDOG_TIME_500MS     ((0 << WDP3) |(1 << WDP2) | (0 << WDP1) | (1 << WDP0))
#define WATCHDOG_TIME_1S        ((0 << WDP3) |(1 << WDP2) | (1 << WDP1) | (0 << WDP0))
#define WATCHDOG_TIME_2S        ((0 << WDP3) |(1 << WDP2) | (1 << WDP1) | (1 << WDP0))
#define WATCHDOG_TIME_4S        ((1 << WDP3) |(0 << WDP2) | (0 << WDP1) | (0 << WDP0))
#define WATCHDOG_TIME_8S        ((1 << WDP3) |(0 << WDP2) | (0 << WDP1) | (1 << WDP0))
```

Za konfiguraciju *Watchdog* tajmera koriste se sljedeće funkcije iz zaglavlja `watchdog.h`:

- `void watchdogOff()` - funkcija kojom se isključuje *Watchdog* tajmer. Ovu funkciju potrebno je uključiti odmah po uključenju mikroupravljača kako bi se onemogućio rad *Watchdog* tajmera tijekom konfiguracije mikroupravljača. Ako se to ne napravi na samom početku, potencijalno mikroupravljač može zaglaviti u beskonačnoj petlji resetiranja.
- `void watchdogSystemResetOn(uint8_t watchdogTime)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za resetiranje mikroupravljača. Ova funkcija kao

argument prima konstantu kojom se definira vrijeme nakon kojeg će *Watchdog* tajmer pokrenuti resetiranje mikroupravljača. Konstante su definirane u programskom kodu 14.2. Ovom se funkcijom pokreće mjerenja vremena pomoću *Watchdog* tajmera.

- `void watchdogInterruptAndSystemResetOn(uint8_t watchdogTime)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za generiranje prekida i resetiranje mikroupravljača. Najprije se generira prekid kojim se poziva prekidna rutina `ISR(WDT_vect)`, a nakon što se prekidna rutina izvrši, pokreće se resetiranje mikroupravljača. Unutar prekidne rutine moguće je pohraniti ključne parametre sustava kako se ne bi izgubili tijekom resetiranja. Ova funkcija kao argument prima konstantu kojom se definira vrijeme nakon kojeg će *Watchdog* tajmer pokrenuti resetiranje mikroupravljača. Konstante su definirane u programskom kodu 14.2. Ovom se funkcijom pokreće mjerenja vremena pomoću *Watchdog* tajmera.

- `void watchdogInterruptOn(uint8_t watchdogTime)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za generiranje prekida kojim se poziva prekidna rutina `ISR(WDT_vect)`. Unutar prekidne rutine moguće je provesti željene naredbe. Teoretski, u prekidnoj rutini moguće je napraviti softversko resetiranje mikroupravljača, odnosno pokretanje programskog koda od početka pozivom instrukcije u asembleru `asm("jmp 0");`. Ova funkcija kao argument prima konstantu kojom se definira vrijeme nakon kojeg će *Watchdog* tajmer pokrenuti resetiranje mikroupravljača. Konstante su definirane u programskom kodu 14.2. Ovom se funkcijom pokreće mjerenja vremena pomoću *Watchdog* tajmera.

- `watchdogReset()` - funkcija kojom se resetira mjerenje vremena *Watchdog* tajmera. Ova funkcija se uvijek nalazi na kraju beskonačne `while` petlje. Ako se ova funkcija pozove prije isteka vremena koje mjeri *Watchdog* tajmer, neće se pokrenuti resetiranje mikroupravljača niti će se pozvati prekidne rutine `ISR(WDT_vect)`.

U nastavku ćemo prikazati nekoliko primjere korištenja navedenih funkcija:

- `watchdogSystemResetOn(WATCHDOG_TIME_250MS)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za resetiranje mikroupravljača koje će se dogoditi ako se funkcija `watchdogReset()` na kraju beskonačne `while` petlje ne pozove unutar 250 ms od pokretanja *Watchdog* tajmera.

- `watchdogInterruptAndSystemResetOn(WATCHDOG_TIME_1S)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za generiranje prekida i resetiranje mikroupravljača koji će se dogoditi ako se funkcija `watchdogReset()` na kraju beskonačne `while` petlje ne pozove unutar 1 sekunde od pokretanja *Watchdog* tajmera.

- `watchdogInterruptOn(WATCHDOG_TIME_4S)` - funkcija kojom se *Watchdog* tajmer mikroupravljača konfigurira za generiranje prekida koji će se dogoditi ako se funkcija `watchdogReset()` na kraju beskonačne `while` petlje ne pozove unutar 4 sekunde od pokretanja *Watchdog* tajmera.

Vježba 14.2

Napišite program koji će koristiti *Watchdog* tajmer za generiranje prekida i resetiranje mikroupravljača ako mikroupravljač ne završi ciklus beskonačne `while` petlje unutar 2 sekunde. Neposredno prije resetiranja mikroupravljača potrebno je u EEPROM memoriju pohraniti stanje jedne 8-bitne cjelobrojne varijable bez predznaka. Pomoću tipkala T2 simulirajte zaglavljenje

programa u beskonačnoj `while` petlji. Prema shemi na slici 5.2, tipkalo T2 spojeno je na digitalni ulaz PD2 mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba142.cpp`. Omogućite prevođenje datoteke `vjezba142.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba142.cpp` prikazan je programskim kodom 14.3.

Programski kod 14.3: Početni sadržaj datoteke `vjezba142.cpp`

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "LCD/lcd.h"
#include "AVR/watchdog.h"
#include <avr/eeprom.h>

uint8_t watchdogBroj;

// watchdog prekidna rutina
ISR(WDT_vect){
    lcdHome();
    lcdprintf("WDT reset\n");
    lcdprintf("Spremam podatke");
    // spemi varijablu watchdogBroj prije reseta
    eeprom_write_byte((uint8_t*)0, watchdogBroj);
}

void init() {
    watchdogOff(); // isključi Watchdog tajmer
    _delay_ms(2000);
    lcdInit(); // inicijalizacija LCD displeja
    // pin PD2 konfiguriran kao ulaz + uključen pritezni otpornik
    pinMode(D2, INPUT_PULLUP);
    watchdogInterruptAndSystemResetOn(WATCHDOG_TIME_2S);
    // ucitaj spremljeni podatak iz EEPROM memorije
    watchdogBroj = eeprom_read_byte((uint8_t*)0);
    interruptEnable(); // globalno omogućenje prekida
}

int main(void) {

    init(); // inicijalizacija mikroupravljača

    while (1) {
        // ispis na LCD displej
        lcdClrScr();
        lcdprintf("%u", watchdogBroj++);
        _delay_ms(1000);

        // dodatno kašnjenje koje ce pokrenuti watchdog tajmer
        if(isFallingEdge(D2)) {
            _delay_ms(60000);
        }
        // resetiranje watchdoga za novi ciklus
        watchdogReset();
    }
}
```

U ovoj vježbi koristi se *Watchdog* tajmer za generiranje prekida i resetiranje mikroupravljača. U inicijalizacijskoj funkciji `init()` provedene su sljedeće konfiguracije i pozvane su sljedeće funkcije:

- pomoću funkcije `watchdogOff()` isključen je *Watchdog* tajmer kako bi se onemogućio rad *Watchdog* tajmera tijekom konfiguracije mikroupravljača. Funkciju `watchdogOff()` je obavezno pozvati na samom početku glavne funkcije `main()` što je ujedno i početak inicijalizacijske funkcije `init()`.

- provedeno je kašnjenje od 2 sekunde kako bi se prilikom generiranja prekida vidio sadržaj LCD displeja,

- inicijaliziran je LCD displej,

- pin PD2 je konfiguriran kao ulaz i na njemu je uključen pritezni otpornik,

- pomoću funkcije `watchdogInterruptAndSystemResetOn(WATCHDOG_TIME_2S)` *Watchdog* tajmer mikroupravljača konfiguriran je za generiranje prekida i resetiranje mikroupravljača kako je navedeno zahtjevima vježbe. Kao argument ova funkcija prima konstantu `WATCHDOG_TIME_2S` kojom se definira vrijeme do pokretanja mehanizma prekida i resetiranja mikroupravljača u iznosu od 2 sekunde. Pozivom ove funkcije, *Watchdog* tajmer počinje mjeriti vrijeme.

- iz EEPROM memorije pročitan je 8-bitni cjelobrojni podatak s adrese 0 u varijablu `watchdogBroj`. Osvježenje cjelobrojnog podatka na adresi 0 u EEPROM memoriji dešava se kod svakog poziva prekidne rutine *Watchdog* tajmera.

- globalno je omogućen prekid.

U beskonačnoj `while` petlji na LCD displeju se ispisuje varijabla `watchdogBroj` i istovremeno se uvećava za 1 svakih 1000 ms. Na kraju beskonačne `while` petlje pozvali smo funkciju `watchdogReset()` kojom se resetira mjerenje vremena *Watchdog* tajmera. Neposredno prije poziva ove funkcije, nalazi se uvjetovano `if` grananje kojim se ispituje da li se dogodio padajući brid tipkala T2 koji je spojen na pin PD2. Ako je ovaj uvjet zadovoljen (ako je pritisnuto tipkalo T2), u iteraciju beskonačne `while` petlje dodat će se kašnjenje u iznosu 60 sekundi.

Ako tipkalo T2 nije pritisnuto, ukupno kašnjenje u beskonačnoj `while` petlji je 1000 ms (jedna sekunda). Ukupno kašnjenje `while` petlje manje je od vremena koje smo definirali konstantom `WATCHDOG_TIME_2S`. Prema tome, pozivom funkcije `watchdogReset()` će se mjerenje vremena pomoću *Watchdog* tajmera iznova resetirati što neće rezultirati generiranjem prekida i resetiranja mikroupravljača.

Ako pritisnemo tipkalo T2, kašnjenje u beskonačnoj `while` petlji iznosi 61 sekundu. Već nakon isteka 2 sekunde, *Watchdog* tajmer će pokrenuti generiranje prekida i resetiranje mikroupravljača. Najprije se generira prekid koji poziva prekidnu rutinu `ISR(WDT_vect)`. Unutar prekidne rutine, na LCD displej ispisuju se dvije poruke jedna ispod druge: `WDT reset` i `Spremam podatke`. Potom se u EEPROM memoriju sprema iznos 8-bitne cjelobrojne varijable `watchdogBroj` na adresu 0. Kada se završi prekidna rutina, pokreće se resetiranje mikroupravljača i programski kod se izvodi iznova. Pritiskom na tipkalo T2 simulirali smo zaglavljenje beskonačne `while` petlje. Kod ponovnog pokretanja programa na mikroupravljaču, iz EEPROM memorije učitava se iznos varijable `watchdogBroj` koju smo pthodno pohranili.

Prevedite datoteku `vjezba142.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Promatrajte ispis na LCD displeju i u proizvoljnom trenutku pritisnite tipkalo T2. Komentirajte ponašanje mikroupravljača nakon pritiska tipkala T2.

Zatvorite datoteku `vjezba142.cpp` i onemogućite prevođenje ove datoteke.

14.3 Sleep modovi rada i upravljanje potrošnjom energije mikroupravljača

Smanjenje potrošnje energije vrlo je bitno brojnim ugradbenim sustavima s mikroupravljačima. Dva su osnovna mehanizma koja omogućuju smanjenje potrošnje energije u AVR mikroupravljačima:

- *Sleep* modovi mikroupravljača (engl. *Sleep Modes*),
- upravljanje potrošnjom energije mikroupravljača (engl. *Power Management*).

Mikroupravljač ATmega328P ima šest *Sleep* modova:

- *Idle Mode* - konstanta za konfiguraciju je `SLEEP_MODE_IDLE`,
- *ADC Noise Reduction* - konstanta za konfiguraciju je `SLEEP_MODE_ADC`,
- *Power Down* - konstanta za konfiguraciju je `SLEEP_MODE_PWR_DOWN`,
- *Power Save* - konstanta za konfiguraciju je `SLEEP_MODE_PWR_SAVE`,
- *Standby* - konstanta za konfiguraciju je `SLEEP_MODE_STANDBY`,
- *Extended Standby* - konstanta za konfiguraciju je `SLEEP_MODE_EXT_STANDBY`.

Opis pojedinog *Sleep* moda mikroupravljača ATmega328P nalazi se u literaturi [2]. *Sleep* modovi omogućuju isključenje pojedinih modula mikroupravljača koji se ne koriste, a sve s ciljem smanjenja potrošnje energije. Mikroupravljači služe za opću namjenu stoga su bogati raznovrsnim hardverom (modulima). U velikom broju primjena mikroupravljača koristi se oko 20 % značajki mikroupravljača. Korištenjem *Sleep* modova, moguće je tako reducirati potrošnju energije jer se brojni moduli mikroupravljača isključuju. Na primjer, *Power Down* mod isključuje sve module mikroupravljača osim vanjskih prekida, I2C komunikaciju i *Watchdog* tajmer. Moduli koji u pojedinom *Sleep* modu ostaju uključeni omogućuju buđenje (engl. *wake up*) mikroupravljača i provedbu svih bitnih zadataka.

Konfiguracija *Sleep* modova provodi se u registru `SMCR`. Osnovne makronaredbe kojima se konfiguriraju *Sleep* modovi mikroupravljača nalaze se u zaglavlju `avr/sleep.h`. Ove makronaredbe su napisane od strane proizvođača mikroupravljača ATmega328P.

Za konfiguraciju *Sleep* modova koriste se sljedeće makronaredbe iz zaglavlja `avr/sleep.h`:

- `set_sleep_mode(mode)` - makronaredba koja prima konstantu na mjestu argumenta `mode` kojom se konfigurira jedan od 6 *Sleep* modova mikroupravljača. Moguće konstante koje se mogu dodijeliti na mjestu argumenta `mode` su: `SLEEP_MODE_IDLE`, `SLEEP_MODE_ADC`, `SLEEP_MODE_PWR_DOWN`, `SLEEP_MODE_PWR_SAVE`, `SLEEP_MODE_STANDBY` i `SLEEP_MODE_EXT_STANDBY`. Naziv konstanti intuitivan je i povezan sa *Sleep* modovima.
- `sleep_mode()` - makronaredba koja pokreće *Sleep* mehanizam pozivom sljedeće tri makronaredbe:
  - `sleep_enable()` - makronaredba kojom se u registru `SMCR` omogućuje *Sleep* mehanizam,
  - `sleep_cpu()` - makronaredba kojom se poziva SLEEP instrukcija mikroupravljača (`asm("sleep")`). Ova instrukcija će mikroupravljač poslati na "spavanje", odnosno pokrenuti *Sleep* mehanizam. Ovisno o konfiguriranom *Sleep* modu, isključit će se

pojedini moduli mikroupravljača. Nakon što se mikroupravljač "probudi" (na primjer kada se pojavi vanjski prekid na INT0), program se nastavlja izvoditi na mjestu poziva makronaredbe `sleep_cpu()`.

– `sleep_disable()` - makronaredba kojom se u registru `SMCR` onemogućuje *Sleep* način rada.

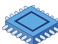
Makronaredbe `set_sleep_mode()` i `sleep_mode(mode)` dostatne su da se na mikroupravljaču pokrene *Sleep* mehanizam, odnosno da se mikroupravljač pošalje na "spavanje". Makronaredbe iz zaglavlja `avr/sleep.h` uključuje se u programski kod naredbom `#include <avr/sleep.h>`.

AVR mikroupravljači omogućuju smanjenje potrošnje energije na način da isključe radni takt za odabrane module (na primjer za tajmere, UART komunikaciju, ...). Upravljanje potrošnjom energije za pojedine module provodi se u registru `PRR`. Osnovne makronaredbe kojima se može isključiti ili uključiti radni takt za pojedine module mikroupravljača nalaze se u zaglavlju `avr/power.h`. Ove makronaredbe su napisane od strane proizvođača mikroupravljača ATmega328P.

U nastavku ćemo prikazati samo nekoliko makronaredbi iz zaglavlja `avr/power.h`:

- `power_timer0_disable()` - makronaredba kojom se isključuje sklop *Timer/Counter0* kako u pozadini ne bi trošio energiju ako se ne koristi,
- `power_timer0_enable()` - makronaredba kojom se ponovno uključuje sklop *Timer/Counter0*,
- `power_timer1_disable()` - makronaredba kojom se isključuje sklop *Timer/Counter1* kako u pozadini ne bi trošio energiju ako se ne koristi,
- `power_timer1_enable()` - makronaredba kojom se ponovno uključuje sklop *Timer/Counter1*,
- `power_usart0_disable()` - makronaredba kojom se isključuje USART modul (za UART komunikaciju) kako u pozadini ne bi trošio energiju ako se ne koristi,
- `power_usart0_enable()` - makronaredba kojom se ponovno uključuje USART modul. Nakon ponovnog uključenja USART modula potrebno ga je ponovno inicijalizirati.
- `power_adc_disable()` - makronaredba kojom se isključuje modul za analogno-digitalnu pretvorbu kako u pozadini ne bi trošio energiju ako se ne koristi,
- `power_adc_enable()` - makronaredba kojom se uključuje modul za analogno-digitalnu pretvorbu.

Makronaredbe iz zaglavlja `avr/power.h` uključuje se u programski kod naredbom `#include <avr/power.h>`.

 Vježba 14.3

Napišite program kojim ćete testirati *Power Down Sleep* mod mikroupravljača ATmega328P. Pomoću tipkala T2 potrebno je "probuditi" mikroupravljač. Dodatno, testirajte upravljanje potrošnjom energije tako da isključite sklop *Timer/Counter1*. LCD displej je potrebno koristiti za ispis statusnih poruka. Prema shemi na slici 5.2, crvena LED dioda spojena je na pin PB1, žuta LED dioda spojena je na pin PB2, a tipkalo T2 spojeno je na digitalni ulaz PD2 (INT0) mikroupravljača ATmega328P. Shema povezivanja LCD displeja na razvojno okruženje

s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba143.cpp`. Omogućite prevođenje datoteke `vjezba143.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba143.cpp` prikazan je programskim kodom 14.4.

Programski kod 14.4: Početni sadržaj datoteke `vjezba143.cpp`

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include <avr/sleep.h>
#include <avr/power.h>

volatile uint8_t brojac = 1;

// prekidna rutina INT0 za wake up
ISR(INT0_vect){
    lcdClrScr();
    lcdprintf("Probudio sam se\n");
    lcdprintf("%d. puta!!!!", brojac++);
    // brojac se osvježava nakon wake up
}

// sklop Timer1 koji mijenja stanje žute LED diode
ISR(TIMER1_OVF_vect){
    timer1SetValue(3096);
    digitalToggle(B2);
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    pinMode(B2, OUTPUT); // PB2 konfiguriran kao izlazni pin (žuta LED dioda)
    // pin PD2 konfiguriran kao ulaz + ukljucen pritezni otpornik
    pinMode(D2, INPUT_PULLUP);
    interruptEnable(); // globalno omogucenje prekida
    int0Enable(); // omoguci INT0 prekide
    int0FallingEdge(); // INT0 prekida na padajuci brid
    // postavljanje normalnog načina rada
    timer1NormalMode();
    timer1SetPrescaler(TIMER1_PRESCALER_64); // F_CPU/64
    // omogućenje prekida preljevom za timer1
    timer1InterruptOVFEnable();
    // početna vrijednost TCNT1 za 250 ms
    timer1SetValue(3096);
    // konfiguracija sleep moda
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    // isključenje sklopa Timer/Counte1
    //power_timer1_disable();
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    // ispis statusne poruke
    lcdprintf("Sleep modovi!");

    while (1) {
        // sekvenca za crvenu LED diodu
        digitalWrite(B1, 1);
```

```
        _delay_ms(1000);
        digitalWrite(B1, 0);
        _delay_ms(1000);
        digitalWrite(B1, 1);
        _delay_ms(1000);
        digitalWrite(B1, 0);
        // ispis statusne poruke
        lcdClrScr();
        lcdprintf("Spavam %d. puta", brojac);
        // slanje mikroupravljaca u sleep
        sleep_mode();
    }
}
```

U ovoj vježbi testirat ćemo *Power Down Sleep* mod mikroupravljača te isključenje sklopa *Timer/Counter1*. U inicijalizacijskoj funkciji `init()` provedene su sljedeće konfiguracije:

- inicijaliziran je LCD displej,
- pinovi PB1 i PB2 konfigurirani su kao izlazi,
- pin PD2 je konfiguriran kao ulaz i na njemu je uključen pritezni otpornik,
- globalno je omogućen prekid.
- vanjski prekid INT0 konfiguriran je tako da generira prekide na padajući brid (kada se pritisne tipkalo T2),
- sklop *Timer/Counter1* konfiguriran je u normalnom načinu rada (vrijeme između poziva prekidne rutine je 250 ms),
- konfiguriran je *Power Down Sleep* mod.

U beskonačnoj `while` petlji najprije se izvodi sekvenca od dva uključenja i isključenja crvene LED diode s kašnjenjima iznosa 1000 ms. Nakon sekvence crvene LED diode, na LCD displeju ispisuje se poruka `Spavam x. puta`. Makronaredbom `sleep_mode()` se pokreće *Sleep* mehanizam.

U programskom kodu 14.4 dvije su prekidne rutine:

- `ISR(INT0_vect)` - ispisuje tekst na LCD diplej `Probudio sam se x. puta!!!!` i uvećava varijablu `brojac` za jedan. Pozivom ove prekidne rutine, mikroupravljač se vraća iz *Sleep* moda u radni mod (*wake up*).
- `ISR(TIMER1_OVF_vect)` - mijenja stanje žute LED diode.

Prevedite datoteku `vjezba143.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Prilikom uključenja mikroupravljača na LCD displeju se ispisuje tekst `Sleep modovi!`. Paralelno s time temeljem programa u beskonačnoj `while` petlji, crvena LED dioda četiri puta promijeni stanje s vremenom između promjene stanja iznosa 1000 ms. Paralelno, od uključenja mikroupravljača sklop *Timer/Counter1* u prekidnom načinu rada izmjenjuje stanje žute LED diode svakih 250 ms. Neposredno prije slanja mikroupravljač u *Sleep* mod, na LCD displeju se ispisuje tekst `Spavam 1. puta`.

U trenutku kada se izvede makronaredba `sleep_mode()`, mikroupravljač ulazi u *Sleep* mod. Za vrijeme kada je mikroupravljač u *Sleep* modu prestaje se izvoditi beskonačna `while` petlja (crvena LED dioda prestaje izmjenjivati stanje), a s radom prestaje i sklop *Timer/Counter1* (žuta

LED dioda prestaje izmjenjivati stanje). Mikroupravljač će ostati u *Sleep* modu do trenutka kada neki od modula koji su aktivni u *Power Down Sleep* modu generiraju prekid.

Pritisnite tipkalo T2. Onog trenutka kada ste pritisnuli tipkalo, na LCD displeju ispisat će se tekst `Probudio sam se 1. puta!!!!` (pozvana je prekidna rutina `ISR(INT0_vect)`). Mikroupravljač je izašao iz *Sleep* moda i počeo s normalnim radom jer se dogodio vanjski prekid INT0. Program u beskonačnoj `while` petlji nastavlja s izvođenjem, što znači da će crvena LED dioda ponoviti sekvencu. Također, sklop *Timer/Counte1* je ponovno uključen i izmjenjuje stanje žute LED diode. Ponovnim izvođenjem makronaredbe `sleep_mode()`, mikroupravljač ulazi drugi puta u *Sleep* mod.

U programskom kodu 14.4 uklonite komentar ispred makronaredbe `power_timer1_disable()` koja se nalazi u inicijalizacijskoj funkciji `init()`. Ponovno prevedite datoteku `vjezba143.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Primijetite da žuta LED dioda ne izmjenjuje stanje jer smo makronaredbom `power_timer1_disable()` isključili sklop *Timer/Counte1* radi uštede energije.

Pokušajte testirati i ostale *Sleep* načine rada. Zatvorite datoteku `vjezba143.cpp` i onemogućite prevođenje ove datoteke.

14.4 Analogni komparator

Analogni komparator uspoređuje dva analogna signala tako da ih međusobno oduzima. Mikroupravljač ATmega328P omogućuje usporedbu analognih signala na pozitivnom ulazu (pinu) AIN0 (PD6) i negativnom ulazu (pinu) AIN1 (PD7). Kao negativni ulaz analognog komparatora mogu se odabrati i ostali analogni ulazi mikroupravljača (ADC0 - ADC7). Analogni komparator može generirati prekid u sljedećim slučajevima:

- kada napon na pinu AIN0 postane veći od napona na pinu AIN1,
- kada napon na pinu AIN1 postane veći od napona na pinu AIN0,
- kada napon na pinu AIN0 postane veći od napona na pinu AIN1 ili kada napon na pinu AIN1 postane veći od napona na pinu AIN0.

Jedna od primjena analognog komparatora je njegovo korištenje u svrhu prekostrujne zaštitu motora. Pretpostavimo da se struja motora mjeri analognim senzorom koji na izlazu linearno mijenja napon u ovisnosti o promjeni struje prema koeficijentu proporcionalnosti 1V /1A (ako senzor na izlazu generira napon iznosa 3 V, struja koju mjeri iznosi 3 A). Senzor struje spojimo na pin AIN1. Pretpostavimo da je maksimalna struja koja smije teći motorom jednaka 4 A. Na ulaz AIN0 možemo spojiti potenciometar (naponsko dijelilo) na način da na ulazu AIN0 bude napon 4 V. Onog trenutka kada struja motora postane veća od 4 A, senzor koji je spojen na ulaz AIN1 generirat će napon veći od 4 V. U tom trenutku, napon na ulazu AIN1 biti će veći od napona na AIN0 pa će analogni komparator generirati prekid kojim se poziva prekidna rutina `ISR(ANALOG_COMP_vect)`. Unutar prekidne rutine možemo isključiti motor i tako ga sačuvati od izgaranja. Konfiguracija analognog komparatora provodi se u registru `ACSR` prema literaturi [2].

Vježba 14.4

Napravite program kojim ćete simulirati prethodno opisanu prekostrujnu zaštitu motora. Razvojno okruženje sa slike 2.1 ne posjeduje elektroničke module za kojim bi se izvela prekostrujna zaštita motora pomoću analognog komparatora. Postavu za prekostrujnu zaštitu moguće je spojiti žicama izvana.

U projektnom stablu otvorite datoteku `vjezba144.cpp`. Omogućite prevođenje datoteke `vjezba144.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba144.cpp` prikazan je programskim kodom 14.5.

Programski kod 14.5: Početni sadržaj datoteke `vjezba144.cpp`

```
#include "AVR/avrlib.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

volatile bool prekidAnaComp = false;

//prekidna rutina analognog komparatora (AIN0(PD6) - AIN1(PD7))
ISR(ANALOG_COMP_vect){
    // iskljuci PWM/motor
    timer1OC1ADisable();
    timer1OC1ADutyCycle(0);
    lcdClrScr();
    lcdprintf("Motor OFF!");
    // onemoguci prekid analognog komparatora
    ACSR &= ~(1 << ACIE);
    prekidAnaComp = true;
}

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    adcInit(); // inicijalizacija AD pretvorbe
    pinMode(B1, OUTPUT); // PB1 konfiguriran kao izlazni pin (crvena LED dioda)
    // postavljanje Phase Correct PWM načina rada za timer1 - 10 bit
    timer1PhaseCorrectPWM10bit();
    // djelitelj frekvencije F_CPU / 8
    timer1SetPrescaler(TIMER1_PRESCALER_8);
    // neinvertirajući PWM signal na PB1 (OC1A)
    timer1OC1AEnableNonInvertedPWM();
    // omoguci prekid analognog komparatora
    ACSR |= (1 << ACIE);
    // prekid kada napon na pinu AIN1 postane veci od napona na pinu AIN0
    ACSR |= (1 << ACIS1) | (0 << ACIS0);
    interruptEnable(); // lobalno omogucenje prekida
}

int main(void) {

    init(); // inicijalizacija mikroupravljaca
    float D;
    lcdprintf("Motor ON!");
    while (1) {
        // promjena brzine vrtnje motora
        D = adcReadScale0To100(ADC0);
        // ako se nije dogodio prekid mijenjaj faktor vodenja
        if (prekidAnaComp == false) {
            timer1OC1ADutyCycle(D);
        }
        _delay_ms(1);
    }
}
```

U ovoj vježbi simulirat ćemo prekostrujnu zaštitu istosmjernog motora. S obzirom da na

razvojnom okruženju nemamo postavu za provedbu prekostrujne zaštite (može se spojiti na razvojno okruženje izvana ako imate senzor struje, DC-DC pretvarač i istosmjerni motor), napravit ćemo sljedeće pretpostavke:

- istosmjerni motor spojen je na DC-DC pretvarač koji je upravljan PWM signalom generiranim pomoću sklopa *Timer/Counter1* na pinu PB1 (OC1A),
- potenciometar kojim se postavlja referentna vrijednost za maksimalnu struju motora spojen je na pin AIN0 (PD6),
- senzor struje kojim se mjeri struja motora spojen je na pin AIN1 (PD7),
- promjena brzine vrtnje motora može se ostvariti pomoću potenciometra koji je spojen na pin ADC0.

Senzor struje na izlaznom pinu generira napon U koji je proporcionalan struji i koju mjeri senzor s konstantom proporcionalnosti koja iznosi 1 V/1A, prema relaciji:

$$U = i\frac{\mathtt{V}}{\mathtt{A}}. \tag{14.2}$$

Analogni komparator cijelo vrijeme oduzima naponske razine na pinovima AIN0 (PD6) i AIN1 (PD7) računajući razliku napona prema relaciji:

$$\Delta U = U_{AIN0} - U_{AIN1}. \tag{14.3}$$

Pretpostavimo da smo na pinu AIN0 (PD6) namjestili konstantan iznos napona $U_{AIN0} = 4$ V. Zadaća prekostrujne zaštite jest da se isključi istosmjerni motor ako struja na njemu postane veća od 4 A. S obzirom da se u ovoj situaciji mora djelovati trenutno, koristimo prekidnu rutinu analognog komparatora kako bismo bez zadrške isključili istosmjerni motor.

Ako je struja motora jednaka 3.5 A, senzor struje će na svom izlazu generirati napon 3.5 V. Napon na pinu AIN1 (PD7) bit će $U_{AIN1} = 3.5$ V. Prema relaciji (14.3), razlika napona će biti $\Delta U = 0.5$ V. U ovoj situaciji prekostrujna zaštita ne treba poduzimati nikakve akcije. Pretpostavimo da je struja motora u jednom trenutku narasla na 4.1 A. Napon na pinu AIN1 (PD7) bit će $U_{AIN1} = 4.1$ V. Prema relaciji (14.3), razlika napona će biti $\Delta U = -0.1$ V. U ovom slučaju prekostrujna zaštita mora reagirati i isključiti motor. Dakle, ako razlika napona ΔU postane negativna, analogni komparator treba generirati prekid. U postavkama analognog komparatora prekid je moguće konfigurirati na sljedeće načine [2]:

- na rastući brid razlike napona ΔU (kada razlika napona ΔU prelazi iz negativne u pozitivnu vrijednost),
- na padajući brid razlike napona ΔU (kada razlika napona ΔU prelazi iz pozitivne u negativnu vrijednost),
- na rastući i na padajući brid razlike napona ΔU (kada razlika napona ΔU prelazi iz negativne u pozitivnu vrijednost te isto kada razlika napona ΔU prelazi iz pozitivne u negativnu vrijednost).

Prema pretpostavkama koje smo napravili u ovoj vježbi, analogni komparator za naše potrebe konfigurirat ćemo da poziva prekidnu rutinu `ISR(ANALOG_COMP_vect)` na padajući brid razlike napona ΔU (kada je napon U_{AIN1} veći od napona U_{AIN0}).

U inicijalizacijskoj funkciji `init()` provedene su sljedeće konfiguracije:

- inicijaliziran je LCD displej,

- inicijalizirana je AD pretvorba,
- pin PB1 je konfiguriran kao izlaz (za potrebe generiranja PWM signala na kanalu OC1A),
- sklop *Timer/Counter1* konfiguriran je u *Phase Correct* načinu rada rezolucije 10 bita,
- omogućeno je generiranje PWM signala na kanalu OC1A,
- konfiguriran je analogni komparator,
- globalno je omogućen prekid.

Konfiguracija analognog komparatora provedena je u registru `ACSR`. Bit `ACIE` omogućuje prekidni način rada analognog komparatora (postavlja se u vrijednost 1) kako bi se pozvala prekidna rutina `ISR(ANALOG_COMP_vect)`. Pomoću bitova `ACIS1` i `ACIS0` konfigurira se brid razlike napona ΔU koji će generirati prekid. Ako bit `ACIS1` ima vrijednost 1, a bit `ACIS0` vrijednost 0, padajući brid razlike napona ΔU generirat će prekid i pozvati prekidnu rutinu `ISR(ANALOG_COMP_vect)`.

Po uključenju sustava na LCD displeju se ispisuje poruka `Motor ON!`. U beskonačnoj `while` petlji mijenja se faktor vođenja PWM signala na kanalu OC1A na koji je spojen DC-DC pretvarač kojim se upravlja brzinom vrtnje istosmjernog motora ako se nije dogodio prekid izazvan analognim komparatorom. Naravno, na našem razvojnom okruženju crvena LED dioda će simulirati uključenost motora, a intenzitet crvene LED diode brzinu vrtnje motora.

Kada napon na pinu AIN1 postane veći od napona na pinu AIN0, poziva se prekidna rutina analognog komparatora `ISR(ANALOG_COMP_vect)`. Ova prekidna rutina onemogućuje generiranje PWM signala na kanali OC1A što će trenutno isključiti motor, a dodatno i faktor vođenja postavlja na 0. Na LCD displeju se ispisuje poruka `Motor OFF!`. Na kraju prekidne rutine onemogućujemo generiranje novih prekida analognog komparatora jer je sustav isključen te se varijabla `prekidAnaComp` postavlja u vrijednost `true` kako bi se onemogućilo mijenjanje faktora vođenja u beskonačnoj `while` petlji.

Prevedite datoteku `vjezba144.cpp` u strojni kod i snimite ga na mikroupravljač ATmega328P. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega328P. Mijenjajte poziciju potenciometra koji je spojen na pin ADC0. Na razvojnom okruženju 2.1 pored releja su na trnove izvučeni svi pinovi porta D. Prstom prođite preko trnova 7 (AIN1 (PD7)) i 6 (AIN0 (PD6)). Generirat ćete smetnju koja će osigurati da u nekom trenutku razlika napona ΔU postane negativna. Analogni komparator će generirati prekid koji će isključiti crvenu LED diodu (simulirani motor) te ispisati na LCD displeju poruku `Motor OFF!`.

Zatvorite datoteku `vjezba144.cpp` i onemogućite prevođenje ove datoteke.

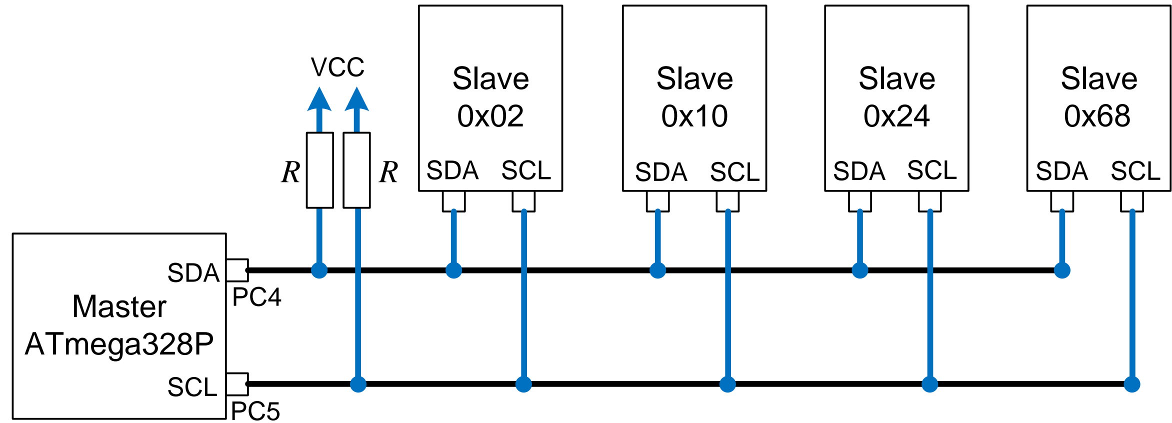
14.5 I2C komunikacija

Nedostatak UART komunikacije jest nemogućnost komunikacije između tri i više uređaja putem iste komunikacijske sabirnice. I2C komunikacija je sinkrona serijska vrsta komunikacije pomoću koje je međusobno moguće povezati do 128 uređaja na istu sabirnicu. Ova komunikacija koristi sabirnicu s dvije žice (linije):

- SCL (engl. *Serial Clock Line*) - linija takta za sinkronizaciju komunikacije,
- SDA (engl. *Serial Data Line*) - podatkovna linija.

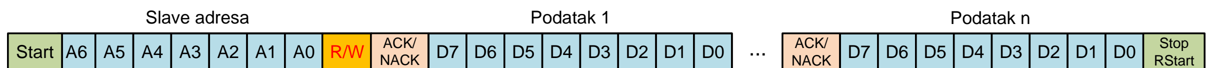
I2C komunikacija je vrsta *Mater-Slave* komunikacije. Shema povezivanja *Master* i *Slave* uređaja na I2C sabirnicu prikazana je na slici 14.1. Komunikaciju uvijek inicira *Master* uređaj

prozivajući po adresi pojedini *Slave* uređaj. *Slave* uređaj ne može inicirati komunikaciju između različitih uređaja spojenih na I2C sabirnicu. Linije SDA i SCL (I2C sabirnica) priteznim su otpornicima spojene na VCC potencijal (npr. 5 V). Ove otpornike potrebno je dodati izvana kako bi *Maste* i *Slave* uređaji mogli ispravno komunicirati. Brzina prijenosa podataka I2C komunikacijom može se konfigurirati do 400 kb/s. I2C komunikacija je vrlo fleksibilna. Novi *Slave* uređaj dovoljno je samo povezati na SDA i SCL linije I2C sabirnice.



Slika 14.1: Shema povezivanja *Master* i *Slave* uređaja na I2C sabirnicu

S obzirom da na istoj sabirnici imamo više *Slave* uređaja, a komunikacija se odvija uvijek između *Master* uređaja i jednog *Slave* uređaja, *Slave* uređaji imaju jedinstvene 7-bitne adrese. Pojedinačni podaci koji se razmjenjuju I2C komunikacijom su 8-bitni. Podatkovni okvir I2C komunikacije prikazan je na slici 14.2.



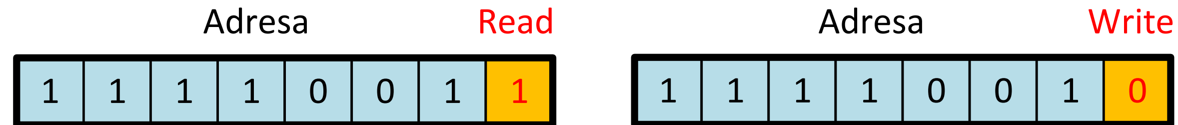
Slika 14.2: Podatkovni okvir I2C komunikacije

Kada uređaji na I2C sabirnici ne komuniciraju, linije SDA i SCL su u visokom stanju. Ovo stanje se zove stanje mirovanja (engl. *idle*). Komunikaciju koju inicira *Master* uređaj uvijek započinje *Start* bitom (na liniji SCL se počinje generirati takt, a na liniji SDA se pojavljuje padajući brid). Nakon toga *Master* uređaj šalje adresu *Slave* uređaja ($A_6A_5A_4A_3A_2A_1A_0$) s kojim želi komunicirati i jedan bit (R/W) koji određuje da li *Master* uređaj želi nešto zapisati na *Slave* uređaj ili želi nešto pročitati s njega. I2C adrese za čitanje sa *Slave* uređaja i pisanje na *Slave* uređaj prikazane su na slici 14.3. Za adresu $A_6A_5A_4A_3A_2A_1A_0$ i R/W bit vrijedi:

- ako *Master* uređaj na I2C sabirnicu pošalje 7-bitnu adresu (primjer adrese na slici 14.3 je 0x79) i nakon toga R/W bit čija je vrijednost 1, tada želi pročitati sadržaj sa *Slave* uređaja s adresom 0x79 = 0b1111001 (slika 14.3 lijevo),

- ako *Master* uređaj na I2C sabirnicu pošalje 7-bitnu adresu (primjer adrese na slici 14.3 je 0x79) i nakon toga R/W bit čija je vrijednost 0, tada želi zapisati sadržaj na *Slave* uređaja s adresom 0x79 = 0b1111001 (slika 14.3 desno).

Nakon što je *Master* uređaj poslao R/W bit, ukoliko postoji *Slave* uređaj s adresom $A_6A_5A_4A_3A_2A_1A_0$, on će poslati potvrdu (ACK bit (engl. *acknowledge*)) da je prepoznao svoju adresu koja je bila na I2C sabirnici. Ostali *Slave* uređaji čekaju da ih se u nekom trenutku prozove s njihovom adresom.



Slika 14.3: Prikaz I2C adrese za čitanje sa *Slave* uređaja i pisanje na *Slave* uređaj

Ako je R/W bit bio u visokom stanju (vrijednost 1), *Master* uređaj čita podatke sa *Slave* uređaja. Nakon što *Slave* uređaj pošalje potvrdu adrese ACK, započinje slanje podataka *Master* uređaju. Broj 8-bitnih podataka koji će *Slave* uređaj poslati *Master* uređaju je proizvoljan. *Master* uređaj će za svaki primljeni 8-bitni podatak *Slave* uređaju poslati potvrdu ACK da je primio podatak. Iznimka od ovog pravila je kada *Master* uređaj želi primiti zadnji 8-bitni podatak, tada *Slave* uređaju šalje NACK bit (engl. *no-acknowledge*). Nakon zadnjeg pročitanog 8-bitnog podatka, *Master* uređaj šalje *Stop* bit kako bi ukazao da je I2C sabirnica slobodna ili bit za ponovljeni start *RStart* kako bi se nastavila nova razmjena podataka.

Ako je R/W bit bio u niskom stanju (vrijednost 0), *Master* uređaj zapisuje podatke na *Slave* uređaj. Nakon što *Slave* uređaj pošalje potvrdu adrese ACK, *Master* uređaj šalje *Slave* uređaju proizvoljan broj 8-bitnih podataka. Nakon svakog uspješno primljenog podatka, *Slave* uređaja šalje *Master* uređaju potvrdu ACK da je primio podatak. Nakon zadnjeg zapisanog 8-bitnog podatka, *Master* uređaj šalje *Stop* bit kako bi ukazao da je I2C sabirnica slobodna ili bit za ponovljeni start *RStart* kako bi se nastavila nova razmjena podataka.

Mikroupravljač ATmega328P ima hardversku podršku za I2C komunikaciju. U AVR mikroupravljačima ovo se sučelje naziva serijsko sučelje s 2 žice (engl. *2-wire Serial Interface - TWI*). Podatkovna linija SDA nalazi se na digitalnom pinu PC4, a linija za takt SCL nalazi se na digitalnom pinu PC5 mikroupravljača ATmega328P. Dakle, alternativna namjena digitalnih pinova PC4 i PC5 jest komunikacija I2C protokolom. TWI sučelje mikroupravljača ATmega328P kompatibilno je s I2C sučeljom. Na razvojnom okruženju 2.1 izvučeni su trnovi s oznakama SDA i SCL na koje možete povezati razne I2C uređaje.

Registar kojim se konfigurira I2C komunikacija je `TWCR`, dok se za slanje i primanje podataka koristi registar `TWDR`. Kako bi se olakšala I2C komunikacija, autor je izradio zaglavlje `"i2c.h"` u kojem se nalaze funkcije koje se koriste za I2C komunikaciju. U nastavku je dan popis i opis funkcija za I2C komunikaciju:

- `void i2cInit()` - funkcija kojom se određuje frekvencija I2C komunikacije.
- `uint8_t i2cStart(uint8_t address, uint8_t RW)` - funkcija koja služi za iniciranje I2C komunikacije slanjem *Start* bita na SDA liniju i pokretanjem takta na SCL liniji. Ova funkcija prima dva argumenta:
  - `address` - adresa *Slave* uređaja s kojime je potrebno ostvariti I2C komunikaciju,
  - `RW` - određuje da li će *Master* uređaj čitati podatke sa *Slave* uređaja ili zapisivati podatke na njega. Ako je vrijednost argumenta 1 ili konstanta `I2C_READ`, tada *Master* uređaj čita podatke sa *Slave* uređaja, a ako je vrijednost argumenta 0 ili konstanta `I2C_WRITE`, tada *Master* uređaj zapisuje podatke na *Slave* uređaj. Povratna vrijednost funkcije je 0 ako je uspješno pokrenuta I2C komunikacija ili 1 ako nije uspješno pokrenuta.
- `uint8_t i2cWrite(uint8_t data)` - funkcija kojom *Master* uređaj šalje 8-bitni podatak na *Slave* uređaj. Ova funkcija prima argument `data` koji je 8-bitni podatak. Povratna vrijednost funkcije je 0 ako je uspješno pokrenuta I2C komunikacija ili 1 ako nije uspješno pokrenuta.

- `uint8_t i2cSReadACK()` - funkcija kojom *Master* uređaj čita 8-bitni podatak sa *Slave* uređaja. Pročitani podatak ova funkcija vraća kao povratnu vrijednost. Ova se funkcija poziva dok god *Master* uređaj ne šalje zahtjev za zadnjim 8-bitnim podatkom.

- `uint8_t i2cSReadNACK()` - funkcija kojom *Master* uređaj čita 8-bitni podatak sa *Slave* uređaja. Pročitani podatak ova funkcija vraća kao povratnu vrijednost. Ova se funkcija poziva kada *Master* uređaj želi pročitati zadnji 8-bitni podatak sa *Slave* uređaja.

- `void i2cStop()` - funkcija koja šalje *Stop* bit na I2C sabirnicu.

U nastavku ćemo pokazati primjere korištenja prethodno opisanih funkcija:

- `i2cStart(0x68, I2C_WRITE)` - pokretanje I2C komunikacije sa *Slave* uređajem na adresi 0x68. *Master* uređaj šalje zahtjev za pisanjem podataka na *Slave* uređaj.

- `i2cStart(0x48, I2C_READ)` - pokretanje I2C komunikacije sa *Slave* uređajem na adresi 0x48. *Master* uređaj šalje zahtjev za čitanjem podataka sa *Slave* uređaja.

- `i2cWrite(120)` - *Master* uređaj šalje vrijednost 120 na *Slave* uređaj.

- `uint8_t d1 = i2cSReadACK()` -*Master* uređaj čita 8-bitni podatak sa *Slave* uređaja. Pročitani podatak sprema se u varijablu `d1`. Nakon poziva ove funkcije, *Master* uređaj će primiti barem jedan podatak sa *Slave* uređaja.

- `uint8_t d2 = i2cSReadNACK()` -*Master* uređaj čita 8-bitni podatak sa *Slave* uređaja. Pročitani podatak sprema se u varijablu `d2`. Nakon poziva ove funkcije, *Master* uređaj neće više čitati podatke sa *Slave* uređaja.

Vježba 14.5

Napravite program kojim ćete mjeriti temperaturu pomoću digitalnog temperaturnog senzora DS1621. Senzor je potrebno spojiti na trnove SDA i SCL razvojnog okruženja sa slike 2.1. Shema povezivanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega328P prikazana je na slici 6.1.

U projektnom stablu otvorite datoteku `vjezba145.cpp`. Omogućite prevođenje datoteke `vjezba145.cpp`, a prevođenje ostalih datoteka s funkcijom `main()` onemogućite. Početni sadržaj datoteke `vjezba145.cpp` prikazan je programskim kodom 14.6.

Programski kod 14.6: Početni sadržaj datoteke `vjezba145.cpp`

```
#include "AVR/avrlib.h"
#include "LCD/lcd.h"
#include "I2C/i2c.h"

#define DS1621_ADDRESS 0x68 // DS1621 - Slave adresa
#define DS1621_START_CNV 0xEE // instrukcija za pokretanje mjerenja temperature
#define DS1621_RD_TEMP 0xAA // instrukcija za ocitanje temperature

void init() {
    lcdInit(); // inicijalizacija LCD displeja
    i2cInit(); // inicijalizacija I2C komunikacije
}

int main(void) {
```

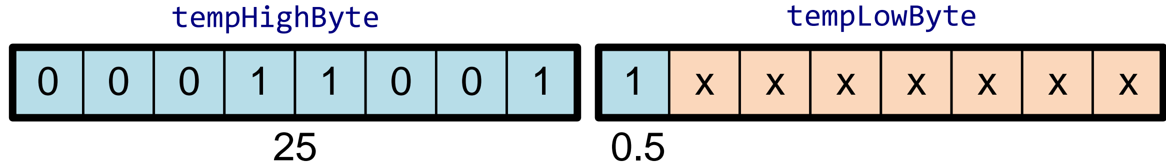
```
    // dva bajta za ocitanje temperature
    uint8_t tempLowByte, tempHighByte;
    // temperatura realan broj
    float T;
    init(); // inicijalizacija mikroupravljaca

    // pokretanje mjerenja temperature
    i2cStart(DS1621_ADDRESS, I2C_WRITE);
    i2cWrite(DS1621_START_CNV);
    i2cStop();

    while (1) {
        // pokretanje ocitanja temperature
        i2cStart(DS1621_ADDRESS, I2C_WRITE);
        i2cWrite(DS1621_RD_TEMP);
        i2cStart(DS1621_ADDRESS, I2C_READ);
        tempHighByte = i2cSReadACK();
        tempLowByte = i2cSReadNACK();
        i2cStop();
        // izracun temperature
        T = (float) tempHighByte;
        if (tempLowByte & 0x80) {
            T += 0.5;
        }
        // ispis temperature na LCD
        lcdClrScr();
        lcdprintf("T = %.1f", T);
        _delay_ms(1000);
    }
}
```

Ova vježba prikazuje primjer I2C komunikacije sa senzorom DS1621. DS1621 je digitalni temperaturni senzor s funkcijom termostata. Temperaturni senzor DS1621 ima mjerni opseg od -55°C do 125 °C s preciznošću od 0.5 °C. Opremljen je sa sučeljem za I2C komunikaciju. Više informacija o ovom senzoru pronađite na Internetu.

U inicijalizacijskoj funkciji `init()` provedene su sljedeće konfiguracije:

- inicijaliziran je LCD displej,
- inicijalizirana je I2C komunikacija.

Temperaturni senzor DS1621 temperaturu sprema u 9-bitnom formatu u dva bajta prikazanom na slici 14.4. Kada se senzoru DS1621 pošalje zahtjev za očitanjem temperature, senzor najprije šalje viši bajt, a nakon toga šalje niži bajt. U višem bajtu je cjelobrojni iznos temperature, a u nižem bajtu se nalazi bit koji ima vrijednost polovine °C (na poziciji najvišeg bita). Za potrebe čitanja dva bajta deklarirali smo dvije varijable tipa `uint8_t`: `tempLowByte` i `tempHighByte`.



Slika 14.4: 9-bitni format temperature senzora DS1621 (25.5 °C)

Nakon inicijalizacije mikroupravljača potrebno je pokrenuti mjerenje temperature na senzoru DS1621. Mjerenje se temperature na senzoru DS1621 može i isključiti što omogućuje uštedu

energije. U nastavku ćemo opisati niz pozvanih funkcija kojima se pokreće mjerenje temperature:

- `i2cStart(DS1621_ADDRESS, I2C_WRITE);` - mikroupravljač ATmega328P šalje *Start* bit na I2C sabirnicu. Nakon toga šalje adresu senzora DS1621 koja se nalazi u konstanti `DS1621_ADDRESS`. S obzirom da se na senzor DS1621 šalje instrukcija za pokretanje mjerenja temperature, kao drugi argument šaljemo konstantu `I2C_WRITE` što će mikroupravljaču ATmega328P omogućiti pisanje na senzor DS1621.

- `i2cWrite(DS1621_START_CNV);` - mikroupravljač ATmega328P šalje na senzor DS1621 konstantu `DS1621_START_CNV`. Ova konstanta je instrukcija temeljem koje senzor DS1621 pokreće mjerenje temperature.

- `i2cStop();` - mikroupravljač ATmega328P šalje *Stop* bit na I2C sabirnicu.

U beskonačnoj `while` petlji provodi se očitanje temperature sa senzora DS1621 na sljedeći način:

- `i2cStart(DS1621_ADDRESS, I2C_WRITE);` - mikroupravljač ATmega328P šalje *Start* bit na I2C sabirnicu. Nakon toga šalje adresu senzora DS1621 koja se nalazi u konstanti `DS1621_ADDRESS`. S obzirom da se na senzor DS1621 šalje instrukcija za očitanje temperature, kao drugi argument šaljemo konstantu `I2C_WRITE` što će mikroupravljaču ATmega328P omogućiti pisanje na senzor DS1621.

- `i2cWrite(DS1621_RD_TEMP);` - mikroupravljač ATmega328P šalje na senzor DS1621 konstantu `DS1621_RD_TEMP`. Ova konstanta predstavlja instrukciju senzoru DS1621 kojom će senzor za sljedeće čitanje podataka s njega pripremiti viši i niži bajt temperature. Neka druga instrukcija bi omogućila čitanje drugih podataka sa senzora DS1621.

- `i2cStart(DS1621_ADDRESS, I2C_READ);` - mikroupravljač ATmega328P šalje ponovljeni *Start* bit na I2C sabirnicu. Nakon toga šalje adresu senzora DS1621 koja se nalazi u konstanti `DS1621_ADDRESS`. S obzirom da će mikroupravljač ATmega328P sada čitati podatke sa senzora DS1621, kao drugi argument šaljemo konstantu `I2C_READ`.

- `tempHighByte = i2cSReadACK();` - mikroupravljač prima viši bajt sa senzora DS1621 i javlja da je primio podatak (ACK). Ovime sugerira da će preuzeti barem još jedan podatak.

- `tempHighByte = i2cSReadNACK();` - mikroupravljač prima niži bajt sa senzora DS1621 bez slanja potvrde što znači da je to zadnji podatak koji prima u ovom prijenosu podataka I2C sabirnicom.

- `i2cStop();` - mikroupravljač ATmega328P šalje *Stop* bit na I2C sabirnicu.

U varijablu `T` pohranjuje se cjelobrojna vrijednost temperature koja je implicitno pretvorena u realni broj. Uvjetovanim grananjem `if (tempLowByte & 0x80){}` se ispituje da li je najviši bit u nižem bajtu jednak 1. Ako je, tada je varijabli `T` potrebno dodati 0.5, inače nije potrebno. Izračunata varijabla `T` ispisuje se na LCD displej. Očitanje temperature provodi se jednom u sekundi.

Ovu vježbu možete prevesti, ali njeno testiranje ovisit će da li imate senzor DS1621. Cilj ove vježbe bio je pokazati kako se koristi I2C komunikacija za čitanje podataka sa senzora koji je *Slave* uređaj. Na tržištu danas postoji velik broj senzora koji rade na I2C komunikaciji, stoga ovu vježbu možete lako prilagoditi svom senzoru. Navedene funkcije koje smo koristili za I2C komunikaciju nalaze se u zaglavlju `"i2c.h"` koje se nalazi u mapi `I2C`. U programski kod ovo zaglavlje uključuje se naredbom `#include "I2C/i2c.h"`.

Zatvorite datoteku `vjezba145.cpp` i onemogućite prevođenje ove datoteke. Zatvorite

programsko razvojno okruženje *Microchip Studio*.

Bibliografija

[1] Z. Vrhovski, *MIKROUPRAVLJAČI - Programiranje mikroupravljača porodice AVR.* https://vub.hr/izdavastvo/knjiga/mikroupravljaci: Veleučilište u Bjelovaru, Bjelovar, 2020.

[2] Atmel, *ATmega328P - 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash.* Microchip, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega3 2015.

[3] XIAMAN OCULAR, www.elmicro.com/files/lcd/gdm1602a_datasheet.pdf, *Specification of LCD Module GDM1602A*, 2005.

[4] G. Gridling and B. Weiss, *Introduction to Microcontrollers.* Vienna University of Technology, Institute of Computer Engineering, Vienna, 2007.

[5] TEXAS INSTRUMENTS, http://www.ti.com/lit/ds/symlink/lm35.pdf, *LM35 Precision Centigrade Temperature Sensors*, 2013.

[6] Z. Vrhovski, *AUTOMATSKO UPRAVLJANJE - analiza i sinteza linearnih kontinuiranih sustava.* https://vub.hr/izdavastvo/knjiga/automatsko-upravljanje-analiza-i-sinteza-linearnih-kontinuiranih- Veleučilište u Bjelovaru, Bjelovar, 2013.