

# Mikroupravljači - Programiranje mikroupravljača porodice AVR

---

**Vrhovski, Zoran**

**Authored book / Autorska knjiga**

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

*Publication year / Godina izdavanja:* **2020**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:144:263144>

*Rights / Prava:* [Attribution-NonCommercial-ShareAlike 4.0 International/Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 4.0 međunarodna](#)

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

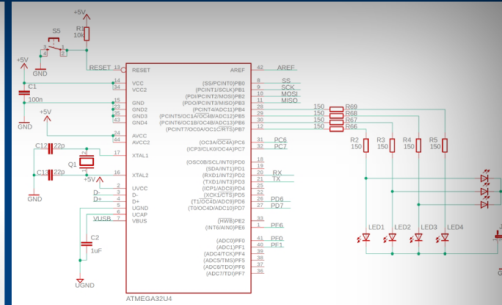
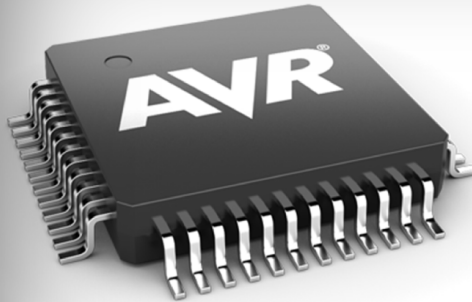


*Repository / Repozitorij:*

[Digital Repository of Bjelovar University of Applied Sciences](#)

# VELEUČILIŠTE U BJELOVARU

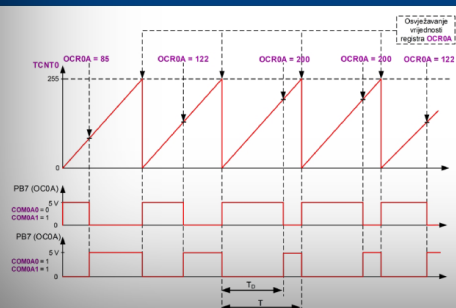
Zoran Vrhovski



```
DigitalOutput crvena(B4);  
DigitalOutput zuta(B5);  
DigitalOutput zelena(B6);  
DigitalOutput plava(B7);
```

```
DigitalInput tipkalo0(D0);  
DigitalInput tipkalo1(D1);  
DigitalInput tipkalo2(F6);  
DigitalInput tipkalo3(F7);
```

```
tipkalo0.pullup_on();  
tipkalo1.pullup_on();  
tipkalo2.pullup_on();  
tipkalo3.pullup_on();
```



## MIKROUPRAVLJAČI

### Programiranje mikroupravljača porodice AVR



Bjelovar, 2020.

VELEUČILIŠTE U BJELOVARU

# MIKROUPRAVLJAČI

Programiranje mikroupravljača porodice AVR

Prvo izdanje



Bjelovar, 2020.

dr. sc. Zoran Vrhovski

MIKROUPRAVLJAČI  
Programiranje mikroupravljača porodice AVR

Izdavač:  
Veleučilište u Bjelovaru

Za izdavača:  
doc. dr. sc. Zrinka Puharić, dr. med.

Tehnički urednik:  
dr. sc. Zoran Vrhovski

Recenzenti:  
dr. sc. Mihael Kukec  
dr. sc. Igor Petrović

Lektorica:  
Mirjana Bučar, prof.

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

ISBN 978-953-7676-34-6

©Niti jedan dio ove knjige ne smije se preslikavati ni umnožavati  
bez prethodne pismene suglasnosti autora.

dr. sc. Zoran Vrhovski

MIKROUPRAVLJAČI  
Programiranje mikroupravljača porodice AVR

Prvo izdanje



Bjelovar, 2020.



# Predgovor

*Tko želi nešto naučiti, naći će način;  
tko ne želi, naći će izliku.*  
Pablo Picasso

Udžbenik Mikroupravljači namijenjen je prvenstveno studentima preddiplomskih stručnih studija Mehatronika i Računarstvo na Veleučilištu u Bjelovaru koji u svom studijskom programu slušaju predmet koji se bavi mikroupravljačima. Svakako se ovaj rukopis može koristiti i na drugim visokim učilištima koja se u svom nastavnom programu bave mikroupravljačima.

Ovaj udžbenik opisuje razvoj programa za mikroupravljač ATmega32U4 u programskom okruženju *Atmel Studio 7*. Programi su pisani u sintaksi programskih jezika C i C++. Vježbe koje će biti opisane u ovom udžbeniku napisane su u programskom okruženju *Atmel Studio 7* i nalaze se na mrežnoj stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Uz vježbe se na mrežnoj stranici nalaze i rješenja vježbi kako biste mogli provjeriti ispravnost vlastitih programskih rješenja. U udžbeniku su opisana rješenja ukupno 58 vježbi koje su vrlo praktične i daju dobar uvod u svijet razvoja programske podrške za ugradbene računalne sustave.

Motiv pisanja ovog udžbenika jest nepostojanje sličnog rukopisa na hrvatskom jeziku. Ako budete uživali u čitanju ovog udžbenika i radu na razvojnom okruženju s mikroupravljačem ATmega32U4 barem upola kao što sam ja uživao stvarajući rukopis, moja će misija biti ispunjena. Možda se pitate zašto nisam koristio neku drugu porodicu mikroupravljača pri pisanju udžbenika. Odgovor je vrlo jednostavan. Ja sam veliki ljubitelj AVR porodice mikroupravljača i to je jedini razlog.

Zahvaljujem se recenzentima dr. sc. Mihaelu Kukecu i dr. sc. Igoru Petroviću na korisnim savjetima i sugestijama te lektorici Mirjani Bučar, prof. na strpljenju pri čitanju ovog udžbenika i usklađivanju teksta s hrvatskim standardnim jezikom.

Posebno se zahvaljujem Veleučilištu u Bjelovaru na potpori u izdavanju ove knjige. Veliku zahvalnost upućujem kolegi Goranu Benkeku koji je aktivno radio na razvoju i proizvodnji razvojnog okruženja s mikroupravljačem ATmega32U4.

Udžbenik posvećujem kćeri Sofiji i supruzi Ani. Obje ste moj neiscrpan izvor energije i hvala vam na tome.

Zoran Vrhovski





# Sadržaj

<b>1</b>	<b>Uvod</b>	<b>1</b>
<b>2</b>	<b>Razvojno okruženje <i>Atmel Studio 7</i></b>	<b>3</b>
2.1	Odabir razvojnog okruženja . . . . .	3
2.2	Stvaranje projekta u razvojnem okruženju <i>Atmel Studio 7</i> . . . . .	4
<b>3</b>	<b>Snimanje i pokretanje strojnog koda na mikroupravljaču ATmega32U4</b>	<b>11</b>
3.1	Razvojno okruženje s mikroupravljačem ATmega32U4 . . . . .	11
3.2	Programiranje mikroupravljača ATmega32U4 . . . . .	15
<b>4</b>	<b>Digitalni izlazi i ulazi</b>	<b>25</b>
4.1	Digitalni izlazi mikroupravljača ATmega32U4 . . . . .	26
4.1.1	Vježbe - digitalni izlazi mikroupravljača ATmega32U4 . . . . .	26
4.1.2	Zadaci - digitalni izlazi mikroupravljača ATmega32U4 . . . . .	42
4.2	Digitalni ulazi mikroupravljača ATmega32U4 . . . . .	44
4.2.1	Vježbe - digitalni ulazi mikroupravljača ATmega32U4 . . . . .	45
4.2.2	Zadaci - digitalni ulazi mikroupravljača ATmega32U4 . . . . .	58
<b>5</b>	<b>LCD displej</b>	<b>61</b>
5.1	Vježbe - LCD displej . . . . .	61
5.1.1	Zadaci - LCD displej . . . . .	78
<b>6</b>	<b>EEPROM memorija</b>	<b>81</b>
6.1	Vježbe - EEPROM memorija . . . . .	81
6.1.1	Zadaci - EEPROM memorija . . . . .	84
<b>7</b>	<b>Analogno-digitalna pretvorba</b>	<b>85</b>
7.1	Vježbe - analogno-digitalna pretvorba . . . . .	85
7.1.1	Zadaci - analogno-digitalna pretvorba . . . . .	99
<b>8</b>	<b>Tajmeri i brojači</b>	<b>101</b>
8.1	Vježbe - Normalan način rada tajmera . . . . .	105
8.1.1	Zadaci - Normalan način rada tajmera . . . . .	123
8.2	Vježbe - PWM način rada tajmera . . . . .	124

---

8.2.1	<i>Fast</i> PWM način rada tajmera . . . . .	124
8.2.2	<i>Phase Correct</i> PWM način rada tajmera . . . . .	127
8.2.3	Konfiguracija PWM načina rada tajmera pomoću zaglavlja " <code>timer.h</code> " . . . . .	130
8.2.4	Zadaci - PWM način rada tajmera . . . . .	147
<b>9</b>	<b>Numerički displej i posmačni registar</b>	<b>149</b>
9.1	Vježbe - numerički displej i posmačni registar . . . . .	156
9.2	Zadaci - numerički displej . . . . .	172
<b>10</b>	<b>Univerzalna asinkrona serijska komunikacija</b>	<b>175</b>
10.1	Vježbe - univerzalna asinkrona serijska komunikacija . . . . .	177
10.1.1	Zadaci - univerzalna asinkrona serijska komunikacija . . . . .	204
<b>11</b>	<b>Vanjski prekidi</b>	<b>207</b>
11.1	Vježbe - vanjski prekidi . . . . .	207
11.1.1	Zadaci - vanjski prekidi . . . . .	218
<b>12</b>	<b>Povezivanje odabranih elektroničkih modula na mikroupravljač</b>	<b>219</b>
12.1	Rotacijski enkoder . . . . .	219
12.1.1	Zadatak - rotacijski enkoder . . . . .	226
12.2	Tranzistor kao sklopka i relej . . . . .	227
12.2.1	Zadatak - tranzistor kao sklopka i relej . . . . .	230
12.3	Ultrazvučni senzor HC-SR04 . . . . .	230
12.3.1	Zadatak - ultrazvučni senzor HC-SR04 . . . . .	236
12.4	Temperaturni senzor LM35 . . . . .	236
12.4.1	Zadatak - temperaturni senzor LM35 . . . . .	238
12.5	Servomotor . . . . .	239
12.5.1	Zadatak - servomotor . . . . .	246
12.6	RGB dioda . . . . .	246
12.6.1	Zadatak - RGB dioda . . . . .	249
	<b>Bibliografija</b>	<b>250</b>

# Poglavlje 1

## Uvod

Mikroračunala su mala, relativno jeftina i pouzdana računala koja sadrže mikroprocesor kao centralnu procesorsku jedinicu, memoriju te ulazno/izlazne digitalne pinove koji omogućuju vezu s vanjskim okruženjem.

Mikroupravljači su upravljačka mikroračunala posebne namjene. Ova mikroračunala sadrže programirajive ulazno/izlazne digitalne pinove, tajmere, analogno-digitalne pretvornike, sučelja za serijsku komunikaciju i drugo. Današnji elektronički uređaji nezamislivi su bez mikroupravljača koji služi za obradu signala, upravljanje sustavima, prikupljanje informacija iz sustava i slično. Mikroupravljači se koriste u automobilima, sustavima automatizacije, kućanstvima, mjernim instrumentima, pametnim kućama, robotima, CNC strojevima, LED rasvjeti, odnosno gotovo u svim tehničkim sustavima. Jedan automobil ima na desetke mikroupravljača koji su sastavni dio sustava sigurnosti, putnog računala, navigacijskog sustava, ozvučenja, sustava zračnih jastuka i drugih.

Ovaj udžbenik opisuje razvoj programa za mikroupravljač ATmega32U4 u programskom okruženju *Atmel Studio 7*. Programi su pisani u sintaksi programskih jezika C i C++. Vježbe koje će biti opisane u ovom udžbeniku napisane su u programskom okruženju *Atmel Studio 7* i nalaze se na mrežnoj stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Uz vježbe se na mrežnoj stranici nalaze i rješenja vježbi kako biste mogli provjeriti ispravnost vlastitih programskih rješenja. U udžbeniku su opisana rješenja ukupno 58 vježbi koje su vrlo praktične i daju dobar uvod u svijet razvoja programske podrške za ugradbene računalne sustave.

U ovom uvodnom poglavlju ukratko ćemo proći kroz sva poglavlja ovog udžbenika kako bismo zainteresirali čitatelje, a to su na prvom mjestu studenti preddiplomskih stručnih studija Mehatronika i Računarstvo na Veleučilištu u Bjelovaru.

U drugom poglavlju ukratko su opisana programska okruženja koja se koriste za razvoj programa za mikroupravljač ATmega32U4. Nadalje, opisano je stvaranje projekata u programskom okruženju *Atmel Studio 7*.

Razvojno okruženje s mikroupravljačem ATmega32U4 i programiranje mikroupravljača ATmega32U4 pomoću softvera *eXtreme Burner - AVR* te pomoću programskog okruženja *Atmel Studio 7* opisano je u trećem poglavlju.

Digitalni izlazi i ulazi mikroupravljača ATmega32U4 kao veza prema vanjskom okruženju opisani su u četvrtom poglavlju. Mikroupravljač ATmega32U4 ima 26 digitalnih pinova koji imaju više namjena, a svaki pin zasebno moguće je konfigurirati kao izlaz ili kao ulaz. Ako je pin konfiguriran kao ulazni pin, na njemu je moguće uključiti pritezni otpornik, što je korisno za digitalne senzore s otvorenim kolektorom, tipkala i krajnje prekidače.

U petom poglavlju opisan je LCD displej koji se često koristi za prikazivanje varijabli sustava

u kojem se nalazi mikroupravljač. Prikazana je konfiguracija raznih tipova LCD displeja koji se mogu spojiti na mikroupravljač ATmega32U4.

EEPROM memorija koja podatke čuva i kad nema napajanja mikroupravljača opisana je u šestom poglavlju.

U sedmom poglavlju opisano je korištenje analogno-digitalne pretvorbe na mikroupravljaču ATmega32U4. Analogni senzori koji se koriste za mjerenje napona, tlaka, vlage, temperature i drugih fizikalnih veličina često se koriste u kombinaciji s mikroupravljačima koji mjerne rezultate mogu prikazivati na LCD displeju ili na neki drugi način.

Primjena tajmera i brojača opisana je u osmom poglavlju. Ovo je jedno od najvažnijih poglavlja jer je primjena tajmera i brojača široka. Opisana su tri načina rada tajmera: normalan način rada, *Fast PWM* način rada i *Phase Correct PWM* način rada. Ovo poglavlje predstavlja uvod u prekide i prekidne rutine.

U devetom poglavlju opisano je korištenje posmačnog registra i numeričkih displeja. Primjena posmačnog registra velika je, a najčešće se koristi u svrhu proširenja digitalnih izlaza. Numerički displeji koriste se za prikaz brojevanih vrijednosti.

Univerzalna asinkrona serijska komunikacija opisana je u desetom poglavlju. Uz ovo poglavlje dostupna je aplikacija kojom se testira asinkrona serijska komunikacija između računala i mikroupravljača ATmega32U4. Ovo poglavlje bitno je zbog upravljanja i nadzora sustava pomoću računala gdje je mikroupravljač posrednik u komunikaciji.

U jedanaestom poglavlju opisani su vanjski prekidi. Ovi prekidi generiraju se na temelju vanjskih događaja koji su najčešće rastući i padajući bridovi signala s digitalnih senzora.

U zadnjem, dvanaestom poglavlju opisani su elektronički moduli koji se mogu spojiti na mikroupravljač ATmega32U4. Elektronički moduli koji su opisani u ovom poglavlju jesu rotacijski enkoder, tranzistor kao sklopka i relej, ultrazvučni senzor HC-SR04, temperaturni senzor LM35, servo motor i RGB dioda. Za svaki od modula razvijen je i program kojim se testira pojedini elektronički modul.

## Poglavlje 2

# Razvojno okruženje *Atmel Studio 7*

### 2.1 Odabir razvojnog okruženja

Prvi korak u razvoju strojnog koda za mikroupravljač jest odabir razvojnog okruženja u kojem ćemo stvarati programska rješenja za zadani problem. Na tržištu postoje razna programska okruženja. Neka od njih jesu:

- BASCOM,
- *Code Vision*,
- *AVR Studio 5*,
- MPLAB X IDE i
- *Atmel Studio 7*.

Programsko okruženje BASCOM jednostavno je za korištenje zato što ima podršku za niz elektroničkih uređaja koji se mogu priključiti na mikroupravljač. Programski jezik koji za razvoj aplikacija koristi BASCOM jest *Basic*. Nedostaci ovog programskog okruženja jesu zatvorenost programskog koda, programiranje u programskom jeziku *Basic*<sup>1</sup> i nemogućnost naprednijeg razvijanja aplikacija. Programsko okruženje BASCOM preporučuje se početnicima te onima koji se mikroupravljačima bave iz hobija.

Programsko okruženje *Code Vision*, kao i BASCOM, ima podršku za niz elektroničkih uređaja. Prednost naspram programskog okruženja BASCOM jest korištenje programskog jezika C za programiranje. Najveći nedostatak programskog okruženja *Code Vision* zatvorenost je programskog koda. Programsko okruženje *Code Vision* preporučujemo početnicima koji žele programirati u ozbiljnijem programskom jeziku i onima koji kasnije žele jednostavno prijeći na naprednija razvojna okruženja.

*AVR Studio 5* preteča je razvojnog okruženja *Atmel Studio 7* koje ćemo koristiti u nastavku ovog udžbenika. *Atmel Studio 7* profesionalno je razvojno programsko okruženje za stvaranje aplikacija mikroupravljača porodice *Atmel*. Ovo programsko okruženje izrađeno je na platformi programskog okruženja *Visual Studio* koje slovi za jedno od najboljih programskih okruženja. Prema tome, *Atmel Studio 7* ima sve prednosti koje donosi programsko okruženje *Visual Studio*. Razvojno je okruženje *Atmel Studio 7* besplatno, za razliku od programskih okruženja BASCOM

---

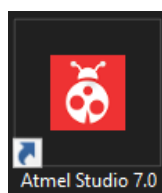
<sup>1</sup>U današnje vrijeme programski jezik *Basic* gotovo se ne koristi zbog niza nedostataka.

i *CodeVision*. MPLAB X IDE razvojno je okruženje istog proizvođača kao i razvojno okruženje *Atmel Studio 7*. Radi se o tvrtki *Microchip* koja je od 2016. godine preuzela tvrtku *Atmel*. MPLAB X IDE razvojno je okruženje koje omogućuje programiranje mikroupravljača porodice PIC, dsPIC, AVR, SAM itd. Programski jezici koji su podržani ovim razvojnim okruženjem jesu C i C++. MPLAB X IDE vrlo je napredno razvojno okruženje, no za AVR porodicu mikroupravljača ono je u beta verziji.

Od navedenih razvojnih okruženja, zbog svih prednosti, za daljnji razvoj programskih rješenja koristit ćemo *Atmel Studio 7*. Najnoviju verziju razvojnog okruženja *Atmel Studio 7* možete preuzeti na stranici <https://www.microchip.com/mplab/avr-support/atmel-studio-7>.

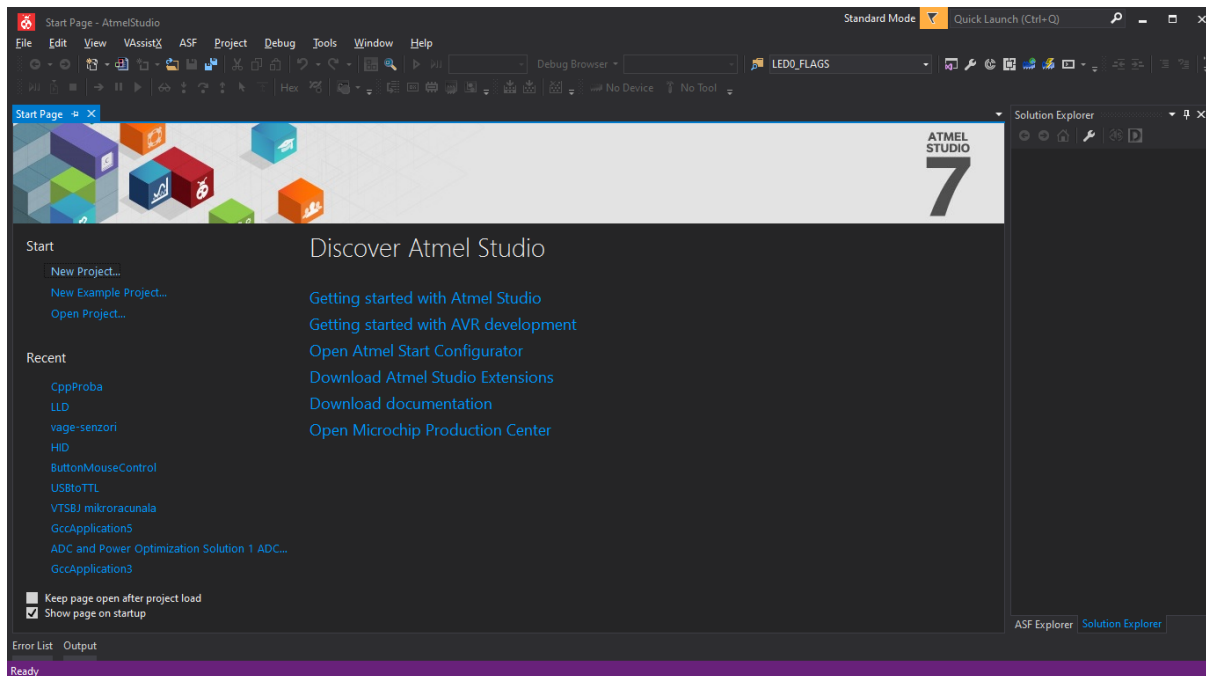
## 2.2 Stvaranje projekta u razvojnom okruženju *Atmel Studio 7*

Razvojno okruženje *Atmel Studio 7* možemo pokrenuti dvostrukim klikom na ikonu sa slike 2.1.



Slika 2.1: *Atmel Studio 7* - ikona

Nakon pokretanja razvojnog okruženja *Atmel Studio 7*, pojavit će se početna stranica prikazana na slici 2.2.



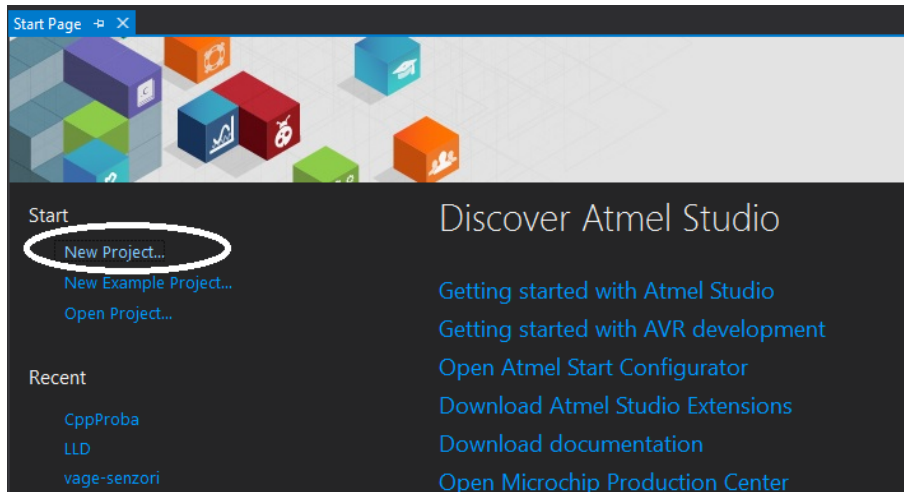
Slika 2.2: Razvojno okruženje *Atmel Studio 7* - početna stranica

Početna stranica omogućuje nam stvaranje novog projekta, otvaranje postojećeg projekta ili otvaranje nedavno otvorenog projekta. Novi projekt možemo stvoriti na jedan od dvaju načina:

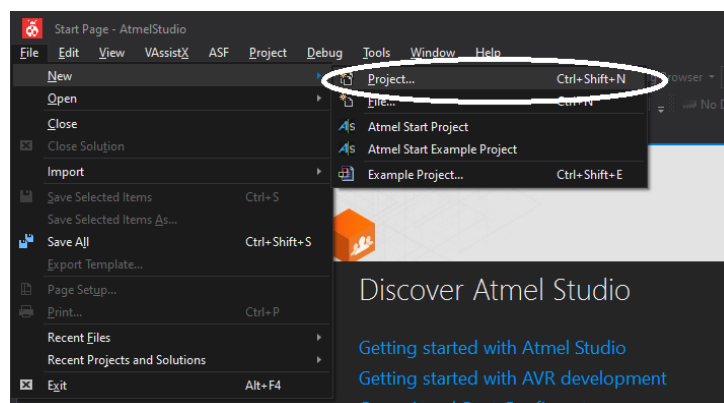
1. na početnoj stranici odaberite *New Project...* (slika 2.3) ili

2. u izborniku odaberite *File* → *New* → *Project...* (slika 2.4),

nakon čega će se otvoriti prozor prikazan na slici 2.5 koji služi za unos vrste, imena i lokacije projekta.



Slika 2.3: Razvojno okruženje *Atmel Studio 7* - stvaranje novog projekta (1)

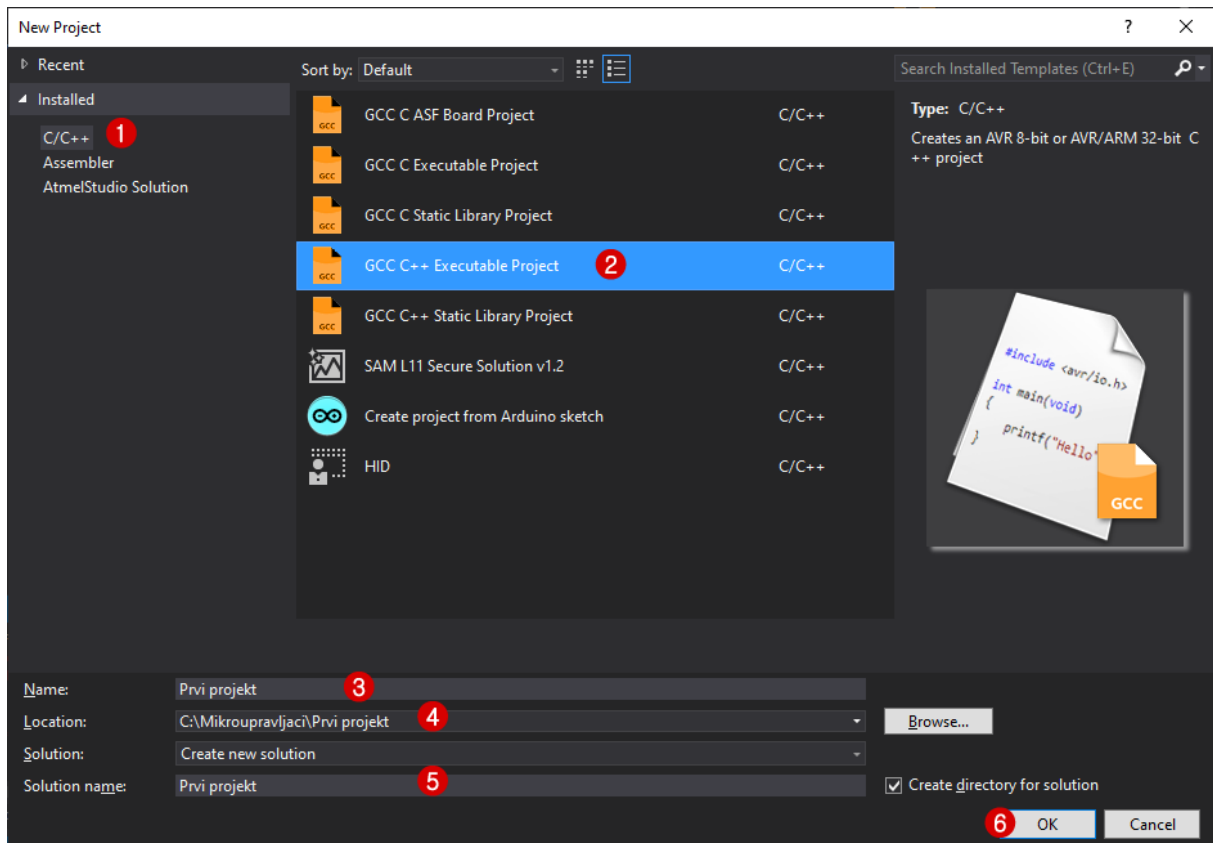


Slika 2.4: Razvojno okruženje *Atmel Studio 7* - stvaranje novog projekta (2)

U prozoru sa slike 2.5 potrebno je provesti korake označene brojevima od 1 do 6:

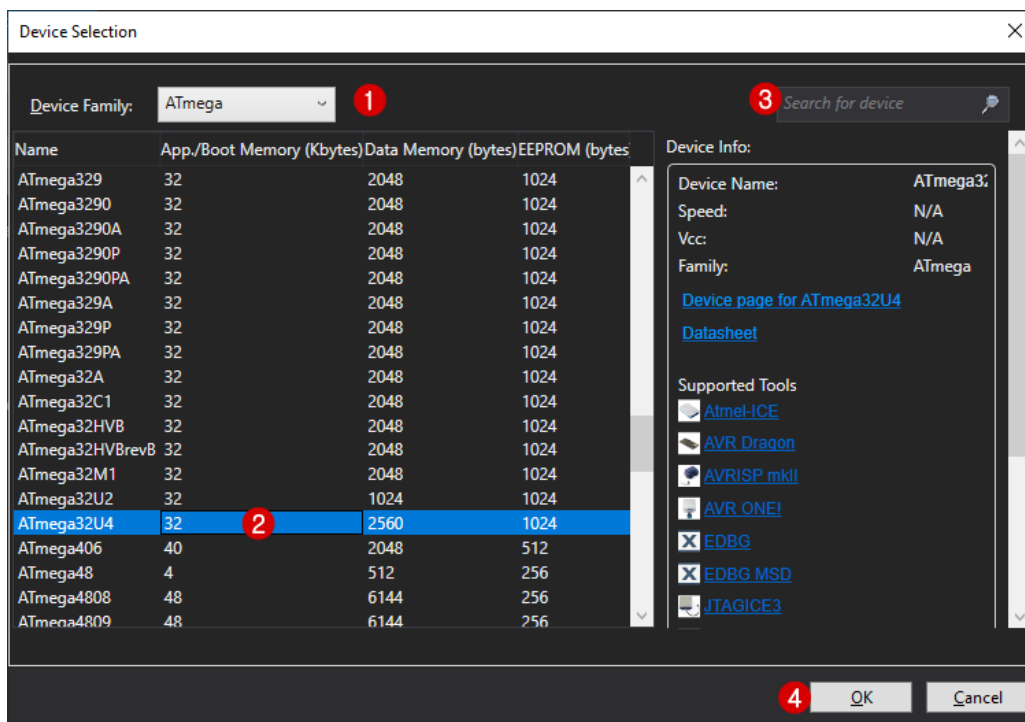
1. U izborniku *Installed Templates* odaberite *C/C++*.
2. Odaberite *GCC C++ Executable Project*.
3. Upišite ime projekta (npr. *Prvi projekt*).
4. Odaberite lokaciju projekta (npr. *C:\Mikroracunala\Prvi projekt\*).
5. Odaberite ime solucije<sup>2</sup> (npr. *Prvi projekt*).
6. Pritisnite OK za nastavak.

<sup>2</sup>U razvojnom okruženju *Atmel Studio 7* jedna solucija može imati nekoliko projekata.



Slika 2.5: Razvojno okruženje *Atmel Studio 7* - prozor za unos vrste, imena i lokacije projekta

Nakon prozora sa slike 2.5 otvara se novi prozor u kojem je moguće odabrati mikroupravljač za koji ćemo pisati program (slika 2.6).



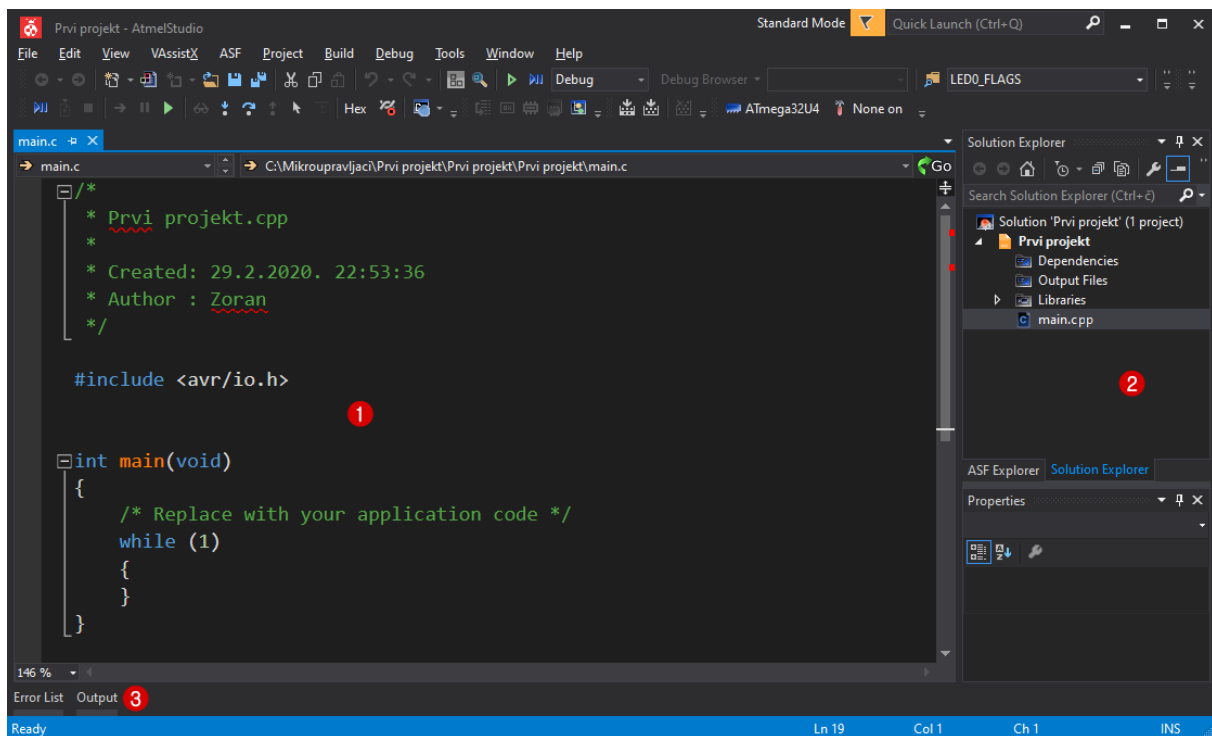
Slika 2.6: Razvojno okruženje *Atmel Studio 7* - odabir mikroupravljača



Odabir mikroupravljača prikazan je koracima od 1 do 4 na slici 2.6:

1. Odaberite porodicu mikroupravljača (npr. ATmega).
2. Odaberite mikroupravljač iz odabrane porodice (npr. ATmega32U4).
3. Korak 1 i 2 mogu se preskočiti ukoliko poznajete točno ime mikroupravljača koji ćete upisati u pretraživač.
4. Pritisnite OK za nastavak.

Nakon što ste završili prethodnu fazu stvaranja novog projekta, otvorit će se uređivač programskog koda prikazan na slici 2.7.



Slika 2.7: Razvojno okruženje *Atmel Studio 7* - uređivač programskog koda

Uređivač programskog koda sadrži sljedeće elemente označene brojevima na slici 2.7:

1. tekstualni uređivač programskog koda,
2. projektno stablo i
3. pokaznik statusnih poruka.

U tekstualni uređivač koda piše se programski kod u programskom jeziku C<sup>3</sup>. Projektno stablo služi za strukturiranje programskog koda u zaglavlja, dok pokaznik statusnih poruka ispisuje upozorenja i pogreške koje se eventualno javljaju prilikom prevođenja programskog jezika C u strojni kod.

<sup>3</sup>Jedan dio programskog koda koji ćemo koristiti pisan je u programskom jeziku C++ te je stoga ekstenzija datoteke s `main()` funkcijom \*.cpp. Programski jezik C++ u pravilu je povratno kompatibilan (engl. *backward compatibility*) s programskim jezikom C.

Prvi projekt stvoren u razvojnom okruženju *Atmel Studio 7* imat će zadaću uključivati LED diode spojene na port B (pinovi 4 do 7) pomoću tipkala spojenih na port D (pinovi 0 i 1) te port F (pinovi 6 i 7). U ovom trenutku nećemo proučavati programski kod, već samo način na koji se koristi razvojno okruženje *Atmel Studio 7* za stvaranje strojnog koda. Programski kod prikazan je u nastavku.

```

/*
 * Prvi_projekt.c
 *
 * Vrijeme kreiranja: 29.2.2020. 23:49:48
 * Autor: Zoran Vrhovski
 */
#include <avr/io.h>

int main(void) {

    // postavljanje ulaznih pinova za tipkala
    DDRD  &= ~( (1 << 0) | (1 << 1) );
    DDRF  &= ~( (1 << 6) | (1 << 7) );

    // uključenje pull up otpornika
    PORTD |= (1 << 0) | (1 << 1);
    PORTF |= (1 << 6) | (1 << 7);

    // postavljanje izlaznih pinova za LED diode
    DDRB |= (1 << 7) | (1 << 6) | (1 << 5) | (1 << 4);
    uint8_t portb = PORTB;

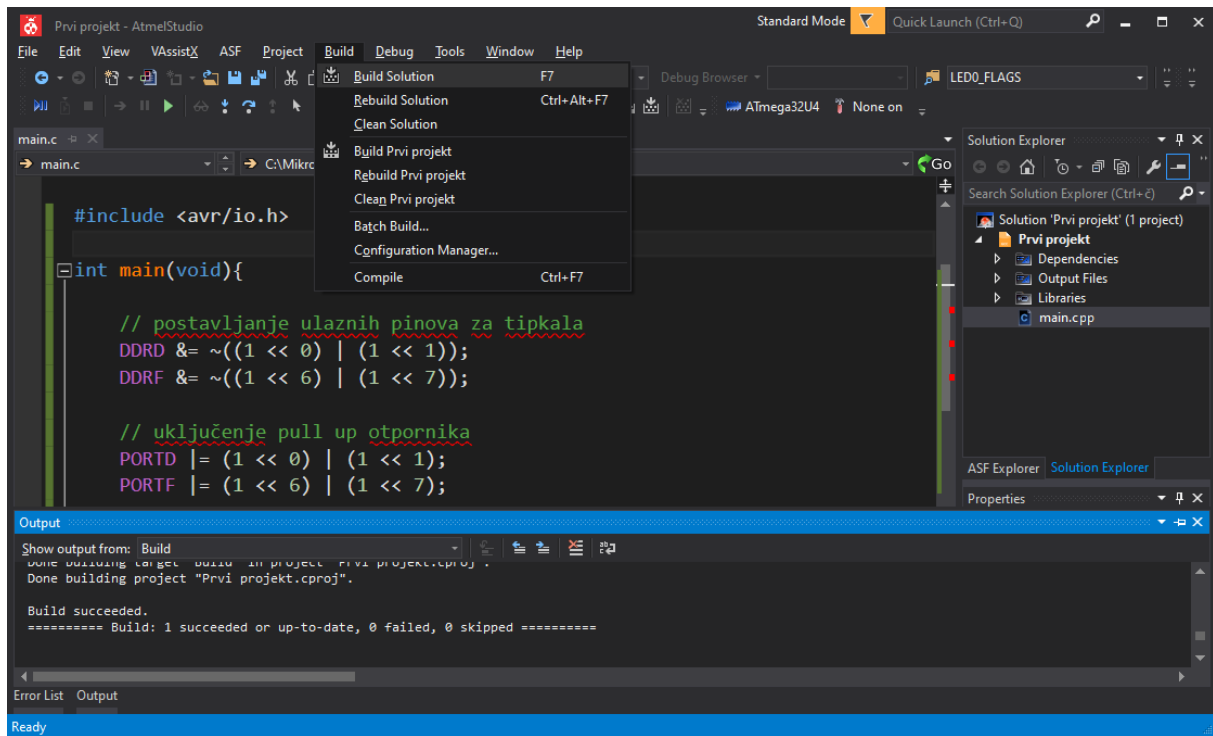
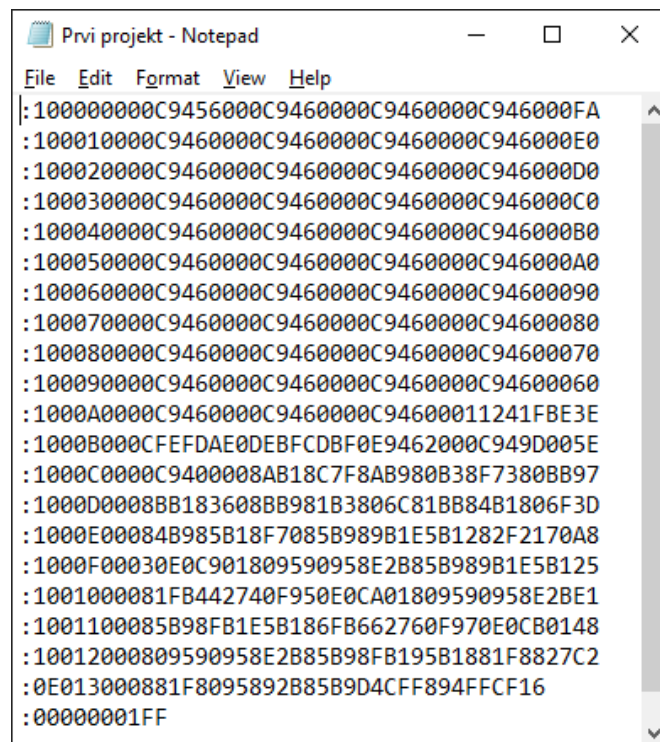
    while (1) {
        portb &= ~( (1 << 7) | (1 << 6) | (1 << 5) | (1 << 4) );
        portb |= ( (~((PIND & (1 << 0)) >> 0) & 0x01) << 4 );
        portb |= ( (~((PIND & (1 << 1)) >> 1) & 0x01) << 5 );
        portb |= ( (~((PINF & (1 << 6)) >> 6) & 0x01) << 6 );
        portb |= ( (~((PINF & (1 << 7)) >> 7) & 0x01) << 7 );
        PORTB = portb;
    }
}

```

Prethodni programski kod potrebno je prevesti u strojni kod. Prevođenje programskog koda moguće je provesti na dva načina (slika 2.8):

- pritisnite tipku *F7* ili
- u izborniku odaberite *Build* → *Build Solution*.

Programski kod uspješno je preveden ako se u pokazniku statusnih poruka (slika 2.8) pojavi poruka „Uspješno prevođenje” (engl. *Build succeeded*). Ukoliko prevođenje nije uspješno, potrebno je korigirati programski kod sukladno pravilima programskog jezika C. Rezultat prevođenja programskog koda strojni je kod prikazan na slici 2.9.

Slika 2.8: Razvojno okruženje *Atmel Studio 7* - prevođenje programskog koda u strojni jezikSlika 2.9: Strojni kod zapisan u datoteci *Prvi projekt.hex*

Datoteka sa strojnim kodom nalazi se na lokaciji `C:\Mikroracunala\Prvi projekt\Prvi projekt\Debug` i ima ekstenziju `*.hex`. Strojni kod sa slike 2.9 u heksadecimalnom je zapisu, a u mikroupravljač se snima pomoću softvera za programiranje mikroupravljača.

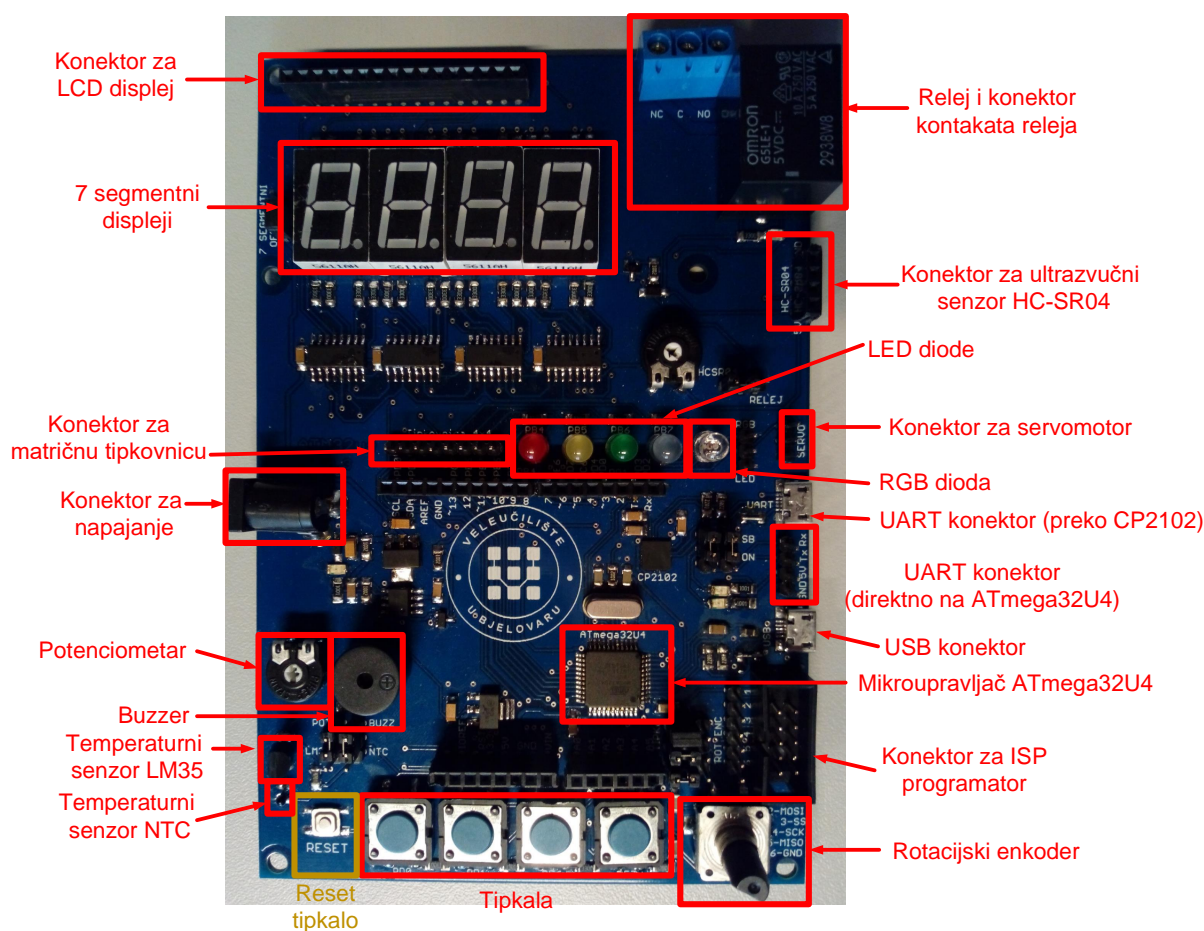


## Poglavlje 3

# Snimanje i pokretanje strojnog koda na mikroupravljaču ATmega32U4

### 3.1 Razvojno okruženje s mikroupravljačem ATmega32U4

U prethodnom poglavlju napisali smo programski kod i preveli ga u strojni kod pomoću razvojnog okruženja *Atmel Studio 7*. Strojni kod potrebno je snimiti na mikroupravljač ATmega32U4 pomoću programatora. U sklopu laboratorijskih vježbi iz predmeta Mikroracunala koristit ćemo razvojno okruženje s mikroupravljačem ATmega32U4 prikazano na slici 3.1.



Slika 3.1: Razvojno okruženje s mikroupravljačem ATmega32U4

Na slici 3.1 prikazani su osnovni dijelovi razvojnog okruženja s mikroupravljačem ATmega32U4:

- konektor za napajanje,
- mikroupravljač ATmega32U4,
- tipkala,
- LED diode i RGB dioda,
- konektori za ultrazvučni senzor HC-SR04 i servomotor,
- USB konektor,
- relej,
- 7 segmentni displej (numerički displej),
- konektori za LCD displej i za matričnu tipkovnicu,
- potenciometar,
- temperaturni senzori LM35 i NTC,
- Buzzer,
- RESET tipkalo,
- rotacijski enkoder,
- konektori za serijsku (UART) komunikaciju,
- konektor za ISP programiranje.

Ostali dijelovi razvojnog okruženja sa slike 3.1 bit će opisani kroz naredna poglavlja. Razvojno okruženje može se napajati putem USB konektora, konektora za napajanje i ISP programatora.

Mikroupravljač ATmega32U4 jedan je od često korištenih mikroupravljača opće namjene (nalazi se na razvojnom okruženju *Arduino Leonardo*). Karakteristike su ATmega32U4 mikroupravljača sljedeće [1]:

- visoke performanse,
- mala snaga,
- AVR<sup>®</sup> porodica, 8-bitni mikroupravljač,
- RISC arhitektura:
  - 135 instrukcija,
  - 32 registra opće namjene,
  - do 16 MIPS (16 milijuna instrukcija u jednoj sekundi) na 16 MHz.
- 32 kB (engl. *In-System Self-programmable Flash*) programske memorije,
- 1 kB EEPROM memorije,
- 2.5 kB SRAM memorije,

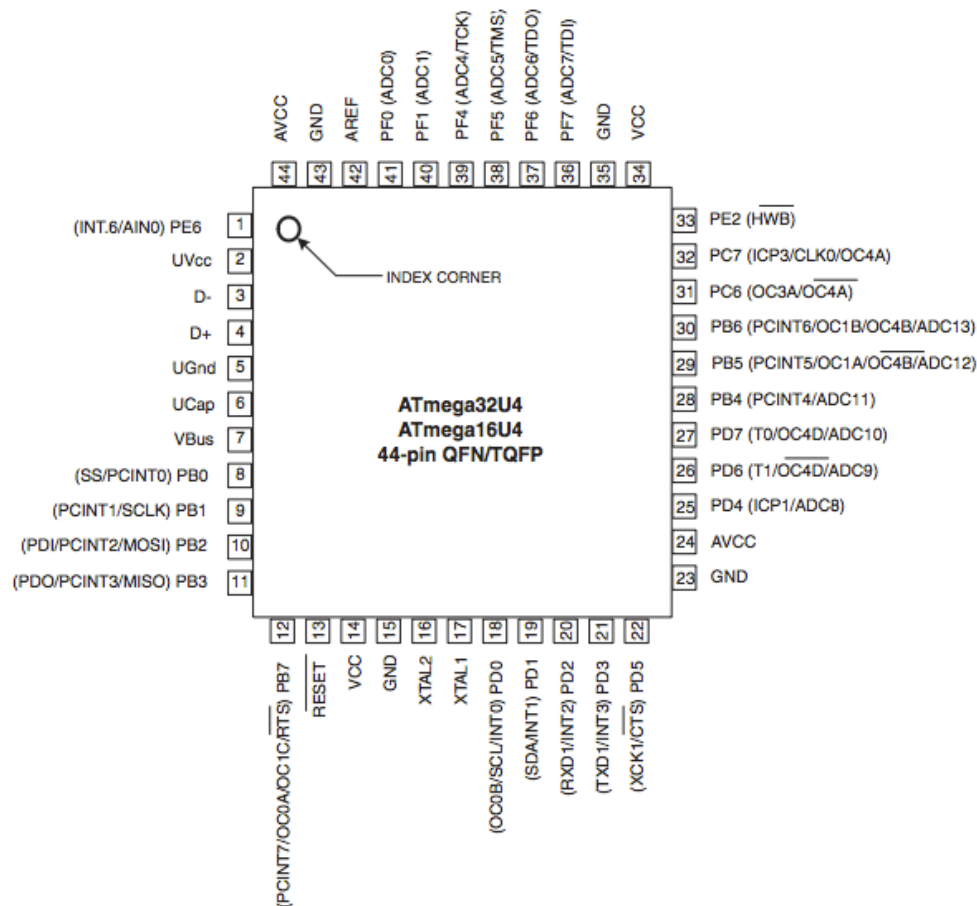
- Ciklus pisanja/brisanja: *Flash* 10 000 puta/EEPROM 100 000 puta,
- JTAG sučelje za ispravljanje pogrešaka u programu,
- jedan 8-bitni tajmer (*Timer*)/brojač (*Counter*) s djelitejima frekvencije,
- dva 16-bitna tajmera (*Timers*)/brojača (*Counters*) s djelitejima frekvencije,
- jedan 10-bitni tajmer (*High-Speed Timer*)/brojača (*High-Speed Counter*) visoke brzine,
- 12 ADC pretvornika (rezolucija 10 bitova),
- dva *Master/Slave* SPI (engl. *Serial Peripheral Interface*) sučelja za komunikaciju,
- jedno *2-wire Serial Interface* sučelje za komunikaciju (isto što i I2C komunikacija),
- programirajući USART (engl. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*),
- programirajući tajmer za nadzor ispravnog rada (engl. *Watchdog Timer*),
- *Power-on Reset*,
- programirajući detektor pada napona napajanja (engl. *Brown Out Detection*),
- unutarnji i vanjski izvor prekida (engl. *Interrupts*),
- šest *Sleep* modova rada,
- interni oscilator 8 MHz (kalibrirani),
- 26 ulaznih/izlaznih pinova koji su smješteni na pet portova,
- I/O podnožje: 44-lead TQFP Package/QFN Package,
- nazivni napon: 2.7 - 5.5 V
- radni takt: 0 - 16 MHz.

Raspored pinova na mikroupravljaču ATmega32U4 prikazan je na slici 3.2. Mikroupravljač ima 26 ulaznih/izlaznih pinova koji su smješteni na pet portova:

- **PORTB** (pinovi: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- **PORTC** (pinovi: PC6, PC7),
- **PORTD** (pinovi: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7),
- **PORTE** (pinovi: PE2, PE6) i
- **PORTF** (pinovi: PF0, PF1, PF4, PF5, PF6, PF7).

Većina je digitalnih pinova višenamjenska, što omogućuje veliku fleksibilnost pri razvijanju elektroničkog uređaja. Tipkala su spojena na port D i F, a LED diode spojene su na port B mikroupravljača i koristit će se za testiranje digitalnih ulaza i izlaza. Serijska komunikacija koristit će se za komunikaciju mikroupravljača s računalom, što nam omogućuje upravljanje i nadzor sustava putem računala. Pinovi mikroupravljača koji se koriste za serijsku komunikaciju jesu RXD1 (PD2) i TXD1 (PD3) (slika 3.2). Vanjski prekidi (engl. *External Interrupts*) okidaju se pomoću pinova INT0 (PD0), INT1 (PD1), INT2 (PD2), INT3 (PD3) i INT6 (PE6) (slika 3.2). Osoba koja razvija elektronički uređaj mora voditi računa da, ukoliko koristi serijsku

komunikaciju, neće imati dostupne vanjske prekide INT2 i INT3 jer serijska komunikacija i vanjski prekidi INT2 i INT3 dijele iste digitalne pinove (PD2, PD3). Analognih ulaza na mikroupravljaču ATmega32U4 ima ukupno 14: ADC0 (PF0), ADC1 (PF1), ADC4 (PF4), ADC5 (PF5), ADC6 (PF6), ADC7 (PF7), ADC8 (PD4), ADC9 (PD6), ADC10 (PD7), ADC11 (PB4), ADC12 (PB5) i ADC13 (PB6) (slika 3.2).



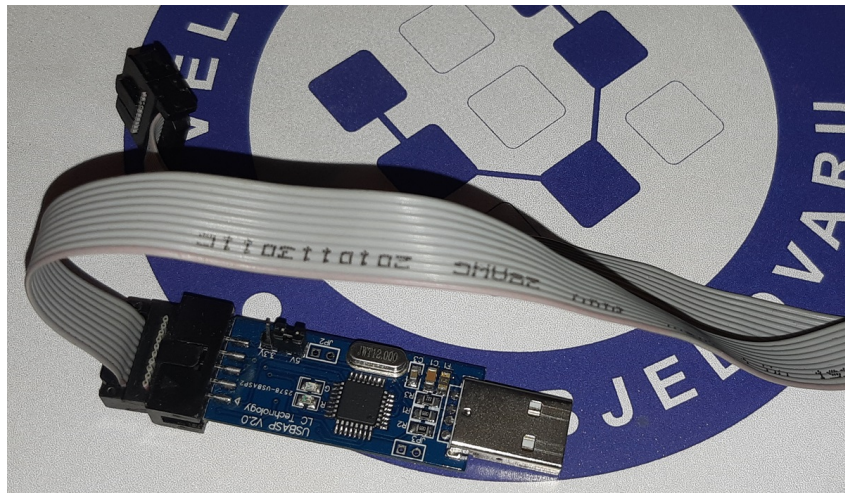
Slika 3.2: Raspored pinova na mikroupravljaču ATmega32U4 [1]

Za programiranje mikroupravljača ATmega32U4 koristit ćemo programator USBasp prikazan na slici 3.3. Prednosti ovog programatora naspram ostalih jesu kompatibilnost s USB sučeljem, niska cijena i besplatni softver za programiranje. Programator je potrebno spojiti na konektor za ISP<sup>1</sup> programator (slika 3.1). Shema spajanja programatora s mikroupravljačem ATmega32U4 prikazana je na slici 3.4. Za programiranje mikroupravljača potrebno je napajanje iznosa 5 V koje se može dovesti ili preko programatora ili preko vanjskog izvora. Programiranje se izvodi putem SPI<sup>2</sup> protokola čije se sučelje sastoji od pinova SS (PB0), SCLK (PB1), MOSI (PB2) i MISO (PB3). Pri programiranju mikroupravljača programator koristi i RESET pin. Vanjski oscilator prikazan na shemi 3.4 oznakom Q1 kvarcni je oscilator frekvencije 16 MHz.

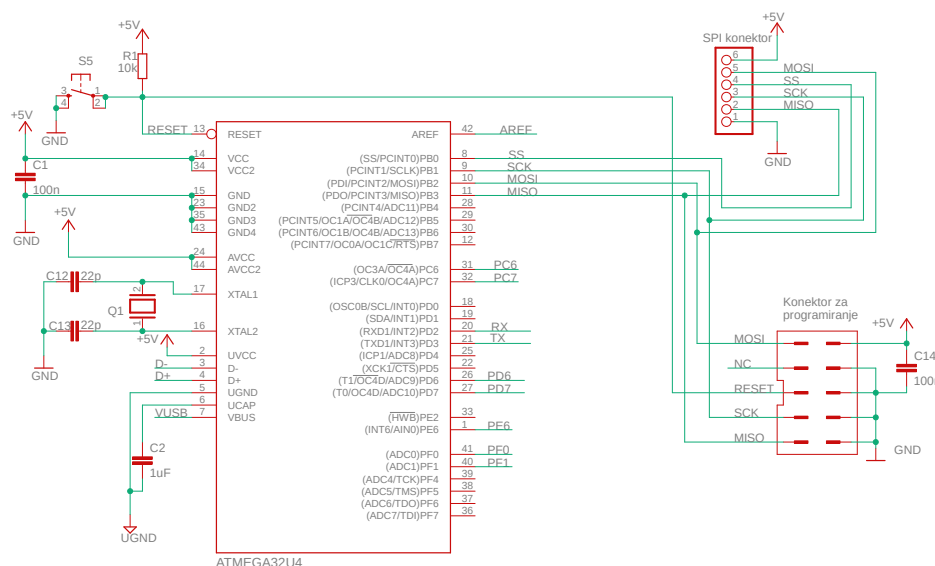
<sup>1</sup>Oznaka ISP dolazi od *In-System Programming*, što ukazuje na mogućnost programiranja mikroupravljača koji se nalazi na razvojnom okruženju. Nekada su se mikroupravljači programirali na samom programatoru, što je zahtijevalo neprestano premještanje mikroupravljača s programatora na razvojno okruženje i obratno.

<sup>2</sup>*Serial Peripheral Interface*.





Slika 3.3: Programator USBasp



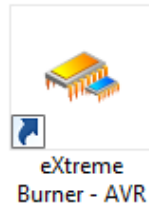
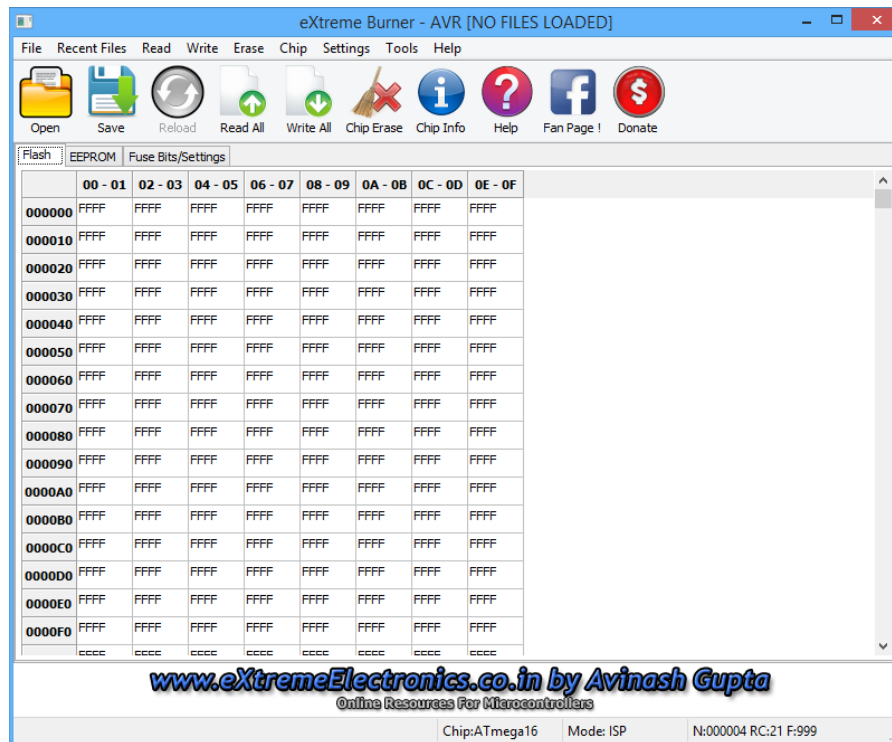
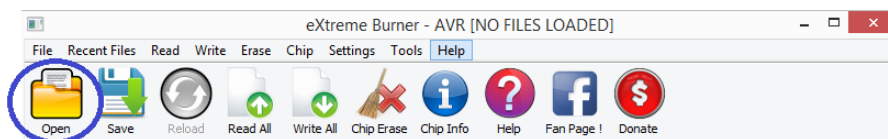
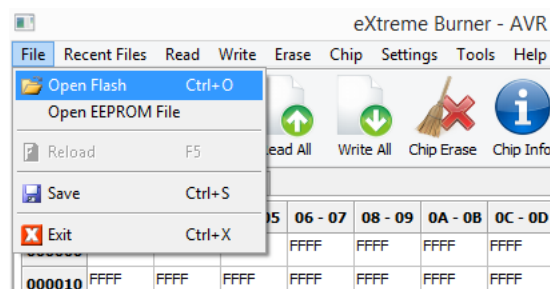
Slika 3.4: Shema spajanja programatora s mikroupravljačem ATmega32U4

## 3.2 Programiranje mikroupravljača ATmega32U4

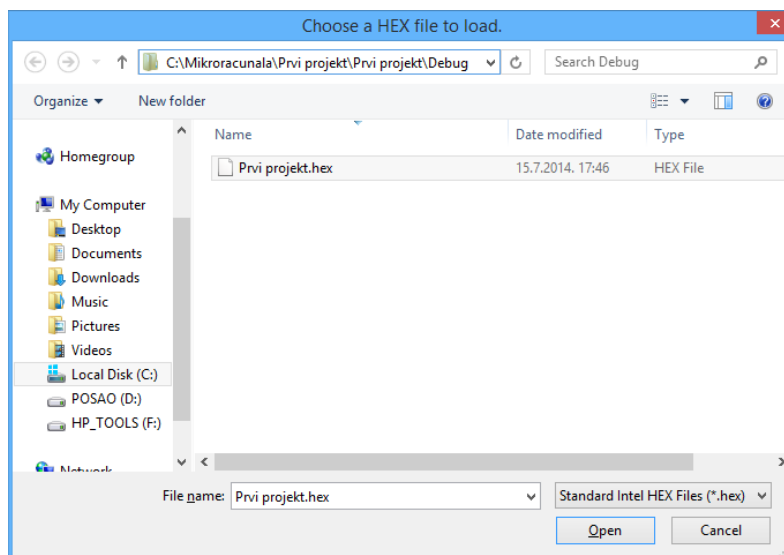
Kao programsku podršku programatoru USBasp koristit ćemo softver *eXtreme Burner – AVR*. Najnoviju verziju softvera *eXtreme Burner – AVR* možete pronaći na stranici <https://extreme-burner-avr.software.informer.com/download/>.

Softver *eXtreme Burner – AVR* možemo pokrenuti dvostrukim klikom na ikonu sa slike 3.5. Nakon pokretanja softvera *eXtreme Burner – AVR* pojavit će se prozor sa slike 3.6. Strojni kod koji smo stvorili pomoću razvojnog programskog okruženja *Atmel Studio 7* potrebno je učitati u softver *eXtreme Burner – AVR*. To možemo učiniti na dva načina:

1. na početnoj stranici softvera *eXtreme Burner – AVR* odaberite *Open* (slika 3.7) ili
2. u izborniku softvera *eXtreme Burner – AVR* odaberite *File → Open Flash* (slika 3.8).

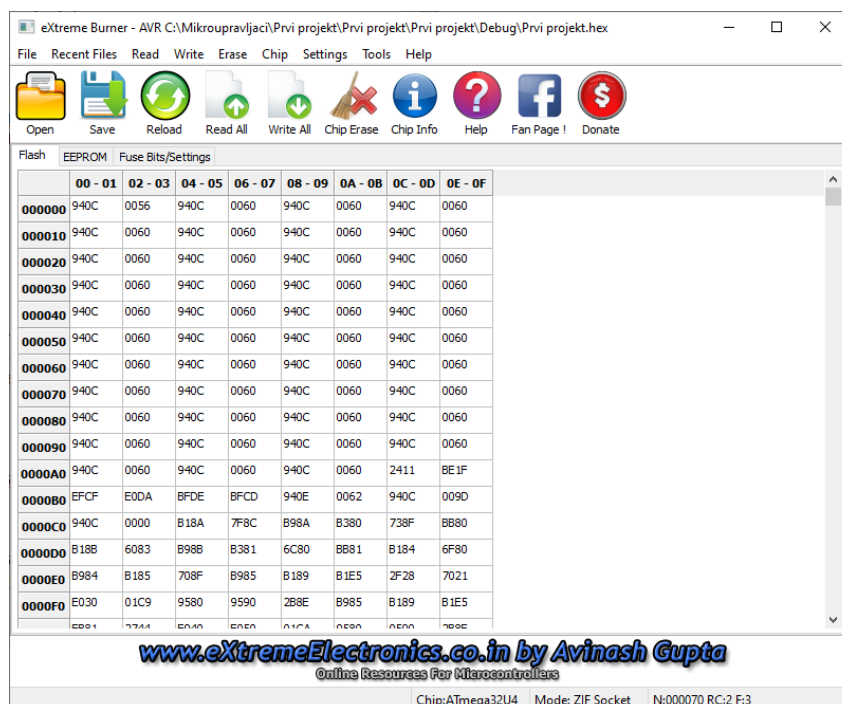
Slika 3.5: Softver *eXtreme Burner - AVR* - ikonaSlika 3.6: Softver *eXtreme Burner - AVR* - početni prozorSlika 3.7: Softver *eXtreme Burner - AVR* - učitavanje strojnog koda (1)Slika 3.8: Softver *eXtreme Burner - AVR* - učitavanje strojnog koda (2)

Prozor za učitavanje strojnog koda prikazan je na slici 3.9. Na slici 3.9 može se primijetiti da je jedina ekstenzija datoteke koja se može učitati u softver *eXtreme Burner – AVR* \*.hex. Strojni kod stvoren razvojnim programskim okruženjem *Atmel Studio 7* nalazi se na lokaciji `C:\Mikroracunala\Prvi projekt\Prvi projekt\Debug`. Na ovoj lokaciji potrebno je odabrati datoteku `Prvi projekt.hex` (slika 3.9).



Slika 3.9: Softver *eXtreme Burner – AVR* - prozor za učitavanje strojnog koda

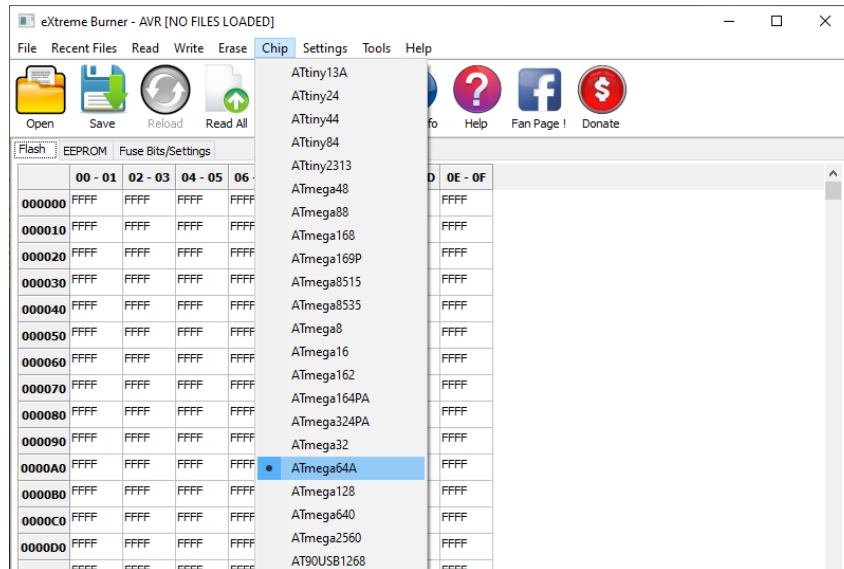
Strojni kod učitani u softver *eXtreme Burner – AVR* prikazan je na slici 3.10 i istovjetan je strojnom kodu sa slike 2.9. Strojni kod nalazi se na kartici *Flash* u softveru *eXtreme Burner – AVR*. Osim kartice *Flash*, softver *eXtreme Burner – AVR* ima kartice EEPROM<sup>3</sup> i *Fuse Bits/Settings*.



Slika 3.10: Strojni kod učitani u softver *eXtreme Burner – AVR*

<sup>3</sup>Memorija koja čuva svoj sadržaj nakon gubitka napajanja.

Nakon učitavanja strojnog koda u softver *eXtreme Burner - AVR* potrebno je odabrati na koji ćemo mikroupravljač snimiti strojni kod. Za odabir mikroupravljača potrebno je u izborniku *Chip* odabrati jedan od ponuđenih mikroupravljača porodice AVR. Mikroupravljač koji moramo odabrati jest ATmega32U4, no, kako se vidi na slici 3.11, on ne postoji na listi ponuđenih mikroupravljača.



Slika 3.11: Softver *eXtreme Burner - AVR* - odabir AVR mikroupravljača

Ovaj problem možemo riješiti dodavanjem novog mikroupravljača na listu. Dodavanje novog mikroupravljača može se provesti na sljedeći način:

1. pomoću uređivača teksta (npr. Notepad++) potrebno je otvoriti XML datoteku **Chips** koja se nalazi na lokaciji `C:\Program Files\eXtreme Burner - AVR\Data4`,
2. u XML datoteku **Chips** dodati XML programski kod koji je crveno zaokružen na slici 3.12 te
3. snimiti promjene u XML datoteci **Chips** i zatvoriti datoteku.

Ključna svojstva u XML programskom kodu za mikroupravljač ATmega32U4 jesu:

- `<NAME>ATmega32U4</NAME>` - ime dodanog mikroupravljača,
- `<FLASH>32768</FLASH>` - količina programske memorije (32 kB = 32768 B),
- `<EEPROM>1024</EEPROM>` - količina EEPROM memorije (1 kB = 1024 B),
- `<SIG>0x0087951E</SIG>` - jedinstveni potpis (engl. *Signature*) mikroupravljača ATmega32U4 koji se može pronaći u njegovoj tehničkoj dokumentaciji:
  1. 0x1E označava proizvođača mikroupravljača
  2. 0x95 označava 32 kB *Flash* memorije
  3. 0x87 označava mikroupravljač ATmega32U4
- `<PAGE>64</PAGE>` - programska memorija podijeljena je na stranice (engl. PAGE) od kojih svaka ima 64 memorijskih riječi (1 memorijska riječ = 16 bitova = 2 B).

<sup>4</sup>Lokacija XML datoteke **Chips** može biti drugačija ukoliko ste softver *eXtreme Burner - AVR* instalirali na nekom drugom mjestu.

Ostala svojstva XML koda označavaju koliko bajtova memorije zauzimaju *Fuse* bitovi, postoje li *Lock* bitovi i postoji li polje za kalibraciju unutarnjeg oscilatora.

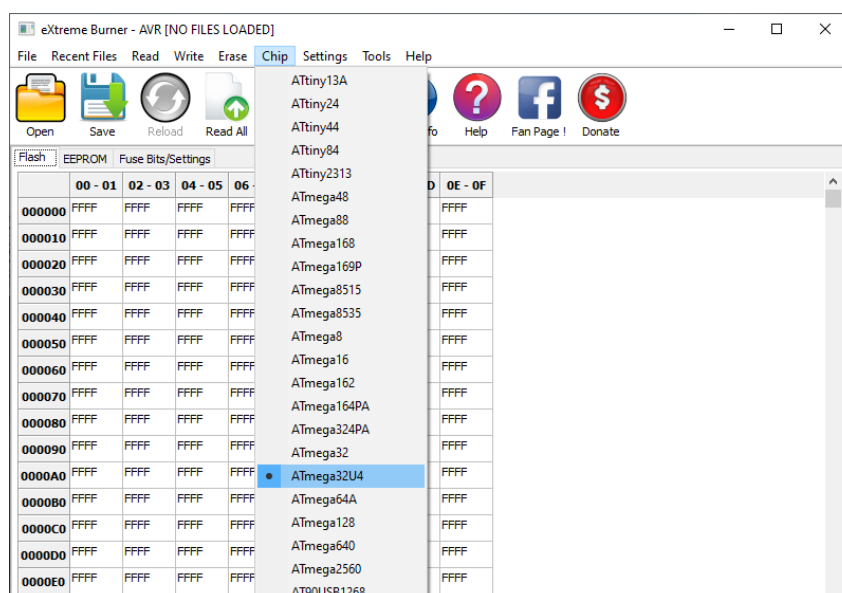
```

249 <CHIP>
250 <NAME>ATmega32U4</NAME>
251 <FLASH>32768</FLASH>
252 <EEPROM>1024</EEPROM>
253 <SIG>0x0087951E</SIG>
254 <PAGE>64</PAGE>
255 <LFUSE layout="1">YES</LFUSE>
256 <HFUSE layout="2">YES</HFUSE>
257 <EFUSE layout="3">YES</EFUSE>
258 <EFUSE>YES</EFUSE>
259 <LOCK>YES</LOCK>
260 <CALIB>YES</CALIB>
261 <PLACEMENT>.\Images\Placements\ZIF_DIP_40.bmp</PLACEMENT>
262 </CHIP>
263
264 <CHIP>
265 <NAME>ATmega64A</NAME>
266 <FLASH>65536</FLASH>
267 <EEPROM>2048</EEPROM>
268 <SIG>0x0002961E</SIG>
269 <PAGE>256</PAGE>
270 <LFUSE layout="1">YES</LFUSE>
271 <HFUSE layout="2">YES</HFUSE>
272 <EFUSE layout="3">YES</EFUSE>
273 <LOCK>YES</LOCK>
274 <CALIB>YES</CALIB>
275 <PLACEMENT>.\Images\Placements\ZIF_DIP_40.bmp</PLACEMENT>
276 </CHIP>

```

Slika 3.12: Sadržaj XML datoteke Chips

Nakon provedene procedure dodavanja novoga mikroupravljača na listu, mikroupravljač ATmega32U4 pojavit će se na popisu mikroupravljača (slika 3.13) te ga je potrebno odabrati.



Slika 3.13: Softver *eXtreme Burner - AVR* - odabir AVR mikroupravljača ATmega32U4

Ako prvi put programiramo mikroupravljač ili mu se mijenjaju osnovne postavke, potrebno je namjestiti tzv. *Fuse* bitove. Ti bitovi određuju rad mikroupravljača. U softveru *eXtreme Burner - AVR* odaberite karticu *Fuse Bits/Settings* prikazanu na slici 3.14.



Slika 3.14: Softver *eXtreme Burner - AVR* - podešavanje *Fuse* bitova

*Fuse* bitovima moguće je mijenjati frekvenciju rada mikroupravljača, izvor radnog takta koji može biti unutarnji i vanjski, način programiranja i drugo. Tvorničke su postavke mikroupravljača ATmega32U4 sljedeće:

- izvor radnog takta vanjski je oscilator frekvencije veće od 8 MHz i sa *Start-up* vremenom  $258 \text{ CK} + 65 \text{ ms}$ ,
- radni takt interno se dijeli s 8,
- omogućeno je programiranje putem SPI sučelja,
- omogućeno je ispravljanje pogrešaka na mikroupravljaču (engl. *JTAG Interface Enabled*),
- *Brown-out* razina podešena je na 2.6 V,
- *Boot Flash* veličine je 2048 memorijskih riječi (4 kB),
- omogućen je *Hardware Boot*.

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

- niži *Fuse* bitovi (engl. *Low Fuse*) jesu 0x5E,
- viši *Fuse* bitovi (engl. *High Fuse*) jesu 0x99,
- prošireni *Fuse* bitovi (engl. *Extended Fuse*) jesu 0xF3.

Za proračun *Fuse* bitova preporučuje se kalkulator *Fuse* bitova koji se nalazi na stranici [www.engbedded.com/fusecalc/](http://www.engbedded.com/fusecalc/). Za potrebe vježbi u nastavku ovog udžbenika podesit ćemo sljedeće vrijednosti *Fuse* bitova:

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

Navedeni *Fuse* bitovi (slika 3.14) odgovaraju sljedećim postavkama mikroupravljača ATmega32U4:

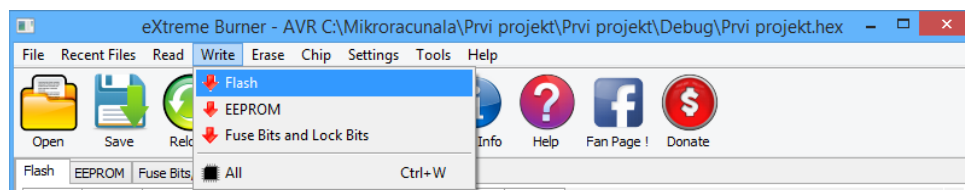
- izvor radnog takta vanjski je oscilator frekvencije veće od 8 MHz i sa *Start-up* vremenom 16k CK + 65 ms,
- omogućeno je programiranje putem SPI sučelja,
- *Brown-out* razina podešena je na 2.6 V,
- *Boot Flash* veličine je 2048 memorijskih riječi (4 kB),

Nakon što proračunamo *Fuse* bitove potrebno ih je snimiti na mikroupravljač. To ćemo učiniti tako da na kartici *Fuse Bits/Settings* slijedimo korake od 1 do 7 (slika 3.14):

1. u polje *Low Fuse* upišite FF,
2. u polje *High Fuse* upišite D9
3. u polje *Extended Fuse* upišite FB,
4. kvačicom označite *Write* u polju *Low Fuse*,
5. kvačicom označite *Write* u polju *High Fuse*,
6. kvačicom označite *Write* u polju *Extended Fuse*,
7. odaberite *Write*.

Osim snimanja *Fuse* bitova, oni se mogu i pročitati iz mikroupravljača ukoliko na kartici *Fuse Bits/Settings* odaberete dugme *Read All*.

U konačnici, nakon namještanja postavki mikroupravljača, potrebno je strojni kod snimiti u programsku memoriju mikroupravljača tako da u softveru *eXtreme Burner – AVR* odaberete *Write* → *Flash*. Postupak je prikazan na slici 3.15.

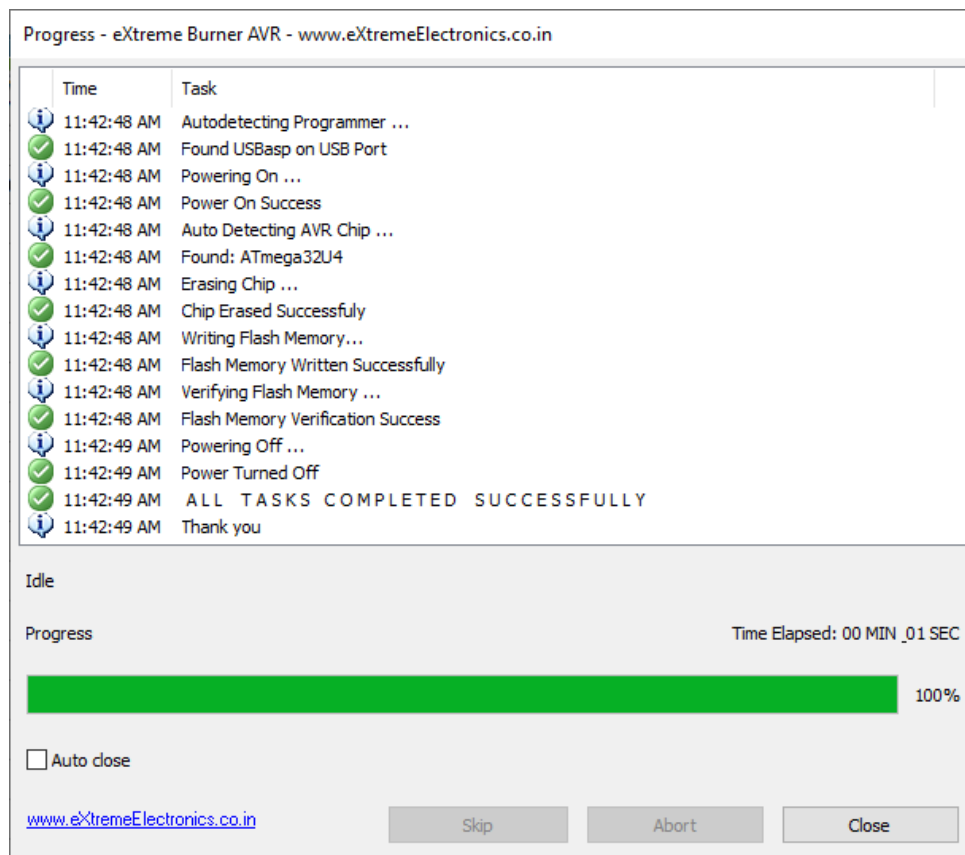


Slika 3.15: Softver *eXtreme Burner – AVR* - snimanje strojnog koda u programsku memoriju mikroupravljača ATmega32U4

Ukoliko je programiranje mikroupravljača uspješno završilo, pojavit će se prozor na slici 3.18. Na slici 3.18 možemo vidjeti da proces programiranja mikroupravljača ima nekoliko koraka:

1. detektiranje mikroupravljača,
2. brisanje programske memorije mikroupravljača,

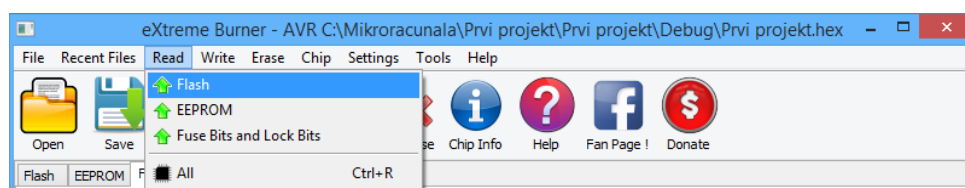
3. snimanje strojnog koda u programsku memoriju mikroupravljača,
4. verifikacija strojnog koda u mikroupravljaču.



Slika 3.16: Softver *eXtreme Burner – AVR* - proces programiranja mikroupravljača

Nakon programiranja mikroupravljača potrebno je provjeriti ispravnost rada programskog koda koji smo napisali na razvojnom okruženju sa slike 3.1. Ako se razvojno okruženje ne ponaša u skladu s našim naumom, potrebno je ispraviti greške u programskom kodu, ponovno prevesti programski kod u strojni kod te snimiti novi strojni kod na mikroupravljač. Ovaj se ciklus ponavlja dokle god ne postignemo ispravan rad razvojnog okruženja u skladu s našim naumom.

Strojni kod iz mikroupravljača moguće je pročitati tako da u softveru *eXtreme Burner – AVR* odaberete *Read* → *Flash* (slika 3.17). Strojni kod nije čitljiv i na temelju njega nećete znati što i kako radi mikroupravljač. Strojni se kod ne može pretvoriti u programski kod koji je čitljiv programeru mikroupravljača.



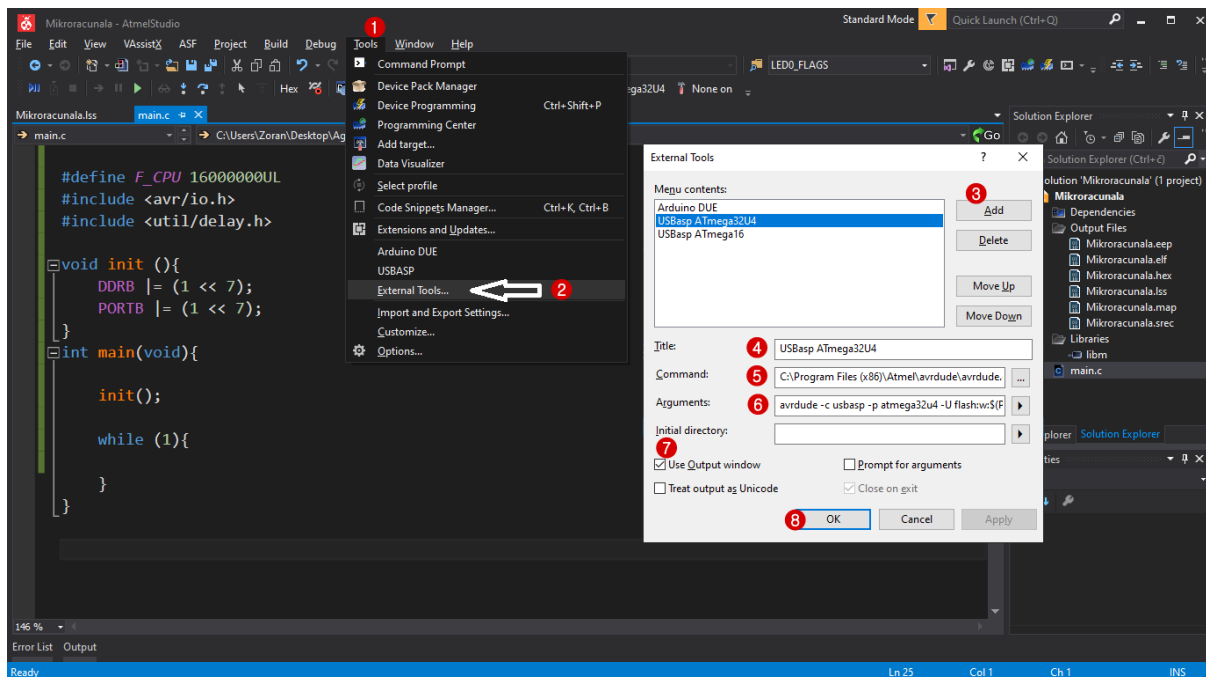
Slika 3.17: Softver *eXtreme Burner – AVR* - čitanje programske memorije mikroupravljača

Princip programiranja mikroupravljača isti je i za ostale mikroupravljače iz porodice *AVR* koji su na popisu mikroupravljača u softveru *eXtreme Burner – AVR* te onih koji se mogu dodati



na popis, a podržavaju ISP programiranje. Prilikom učitavanja strojnog koda u softver *eXtreme Burner – AVR* pazite da nijedna datoteka na putanji prema \*.hex datoteci nema dijakritičkih znakova. U suprotnom softver *eXtreme Burner – AVR* neće učitati strojni kod.

Programiranje mikroupravljača porodice *AVR* moguće je i neposredno iz razvojnog okruženja *Atmel Studio 7*. U tu svrhu potrebno je imati programator AVRISP mkII proizvođača *Microchip*. Taj programator puno je skuplji od programatora USBasp, ali zato omogućuje razvoj aplikacije u jednom softveru i brže ispravljanje grešaka. No, postoji rješenje u kojem je moguće dodati vanjski uređaj za programiranje u razvojno okruženje *Atmel Studio 7*. Na slici 3.18 je prikazan postupak dodavanja USBasp programatora za mikroupravljač ATmega32U4 kao vanjskog uređaja u razvojno okruženje *Atmel Studio 7*.



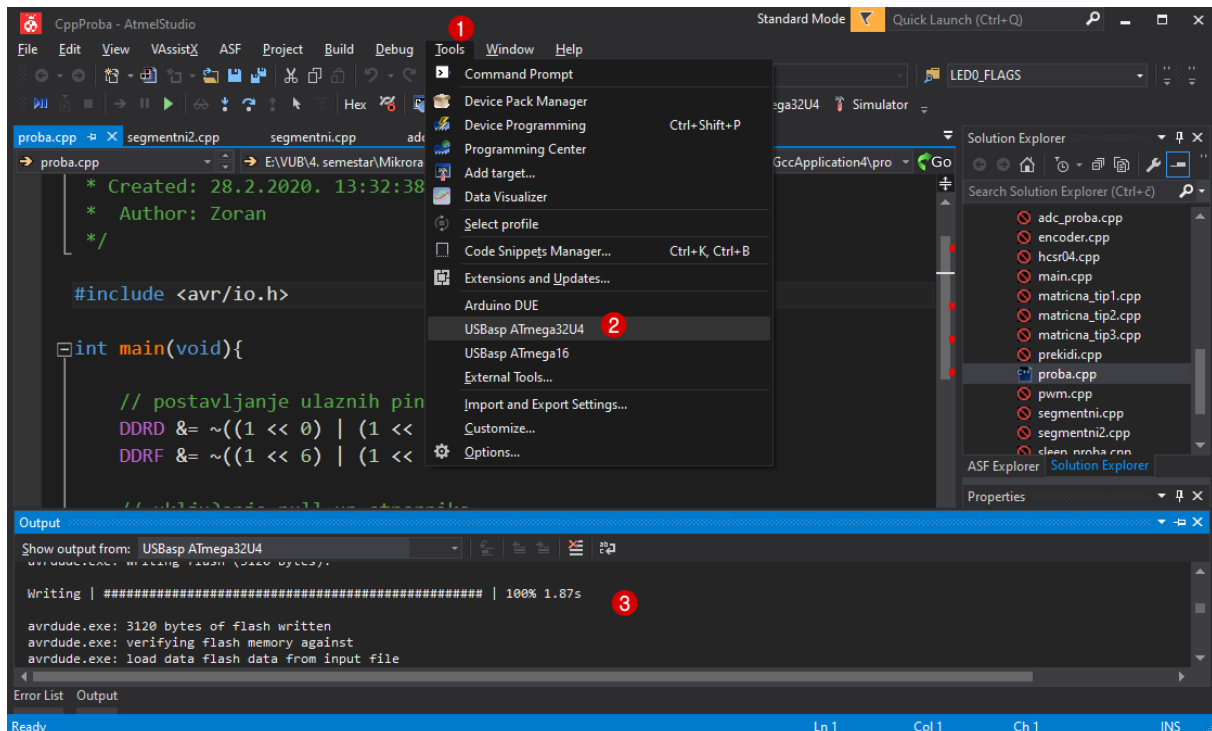
Slika 3.18: Programiranje mikroupravljača iz razvojnog okruženja *Atmel Studio 7* - parametriranje programatora USBasp

Postupak dodavanja USBasp programatora za mikroupravljač ATmega32U4 može se provesti u osam koraka:

1. u razvojnom okruženju *Atmel Studio 7* odaberite izbornik **Tools**,
2. u izborniku **Tools** odaberite **External Tools...**,
3. u novootvorenom prozoru pritisnite gumb **Add**,
4. u polju **Title** upišite: USBasp ATmega32U4,
5. u polje **Command** upišite: C:\Program Files (x86)\Atmel\avrdude\avrdude.exe<sup>5</sup>
6. u polje **Arguments** upišite:  
avrdude -c usbasp -p atmega32u4 -U flash:w:\$(ProjectDir)Debug\\$(TargetName).hex:i
7. označite opciju **Use Output window**,
8. završite parametriranje pritiskom na gumb **OK**.

<sup>5</sup>Potrebno je skinuti *Avrdude* sa stranice <https://www.nongnu.org/avrdude/>.

Nakon provedenog postupka, prevedeni programski kod za mikroupravljač ATmega32U4 možete snimiti na razvojno okruženje sa slike 3.1 odabirom **Tools** -> **USBasp ATmega32U4** (slika 3.19 - koraci 1 i 2). Uspješnost programiranja možete pratiti u **Output** prozoru (slika 3.19 - korak 3).



Slika 3.19: Programiranje mikroupravljača iz razvojnog okruženja *Atmel Studio 7* pomoću programatora USBasp

*Fuse* bitove i dalje je potrebno snimiti pomoću softvera *eXtreme Burner – AVR*, no kada ih jednom podesite daljnje programiranje možete provoditi pomoću razvojnog okruženja *Atmel Studio 7*, što će Vam omogućiti rad u samo jednom softveru.

Strojni kod koji se nalazi u programskoj memoriji mikroupravljača izvodi se onog trenutka kada na mikroupravljač dovedemo napajanje i ako na pin RESET dovedemo visoko stanje (5 V). Ako mikroupravljač izgubi napajanje ili na pin RESET dovedemo nisko stanje (0 V), mikroupravljač se zaustavlja, a ponovnim se pokretanjem mikroupravljača strojni kod izvodi ispočetka.

Najčešća greška koju rade početnici u programiranju mikroupravljača jest izostavljanje visoke razine na pinu RESET mikroupravljača pa iz tog razloga mikroupravljač neće raditi. Na slici 3.4 možemo vidjeti da je visoko stanje na pin RESET dovedeno preko priteznog (engl. *pull-up*) otpornika iznosa otpora 10 k $\Omega$ , što znači da je dovoljno na mikroupravljač dovesti samo napajanje kako bi on izvodio strojni kod.

## Poglavlje 4

# Digitalni izlazi i ulazi

Mikroupravljač ATmega32U4 ima 26 digitalnih pinova koji se mogu konfigurirati kao izlazni pinovi ili kao ulazni pinovi. Smjer djelovanja digitalnog pina može se mijenjati tijekom rada mikroupravljača. Na primjer, digitalni pin može biti izlazni pin te tijekom rada mikroupravljača postati ulazni pin i obratno. Digitalni pinovi mikroupravljača ATmega32U4 raspoređeni su u pet grupa koje nazivamo portovima. Mikroupravljač ATmega32U4 ima sljedeće portove s oznakama digitalnih pinova:

- **PORTB** (oznake digitalnih pinova: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- **PORTC** (oznake digitalnih pinova: PC6, PC7),
- **PORTD** (oznake digitalnih pinova: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7),
- **PORTE** (oznake digitalnih pinova: PE2, PE6),
- **PORTF** (oznake digitalnih pinova: PF0, PF1, PF4, PF5, PF6, PF7).

Svaki digitalni pin ima višestruku namjenu, pa se npr. pinovi PD2 i PD3 koriste za serijsku komunikaciju. Način rada digitalnog pina na nekom portu određuju registri za konfiguraciju:

- **DDRx**, ( $x = B, C, D, E, F$ ), (engl. *Data Direction Register*) - registar smjera podataka,
- **PORTx**, ( $x = B, C, D, E, F$ ) - podatkovni registar  $i$
- **PINx**, ( $x = B, C, D, E, F$ ) - registar ulaznih pinova.

Svi navedeni registri širine su 8 bitova. Pozicija bita  $i$  ( $i = 0, 1, \dots, 7$ ) u registru određuje konfiguraciju pina na poziciji  $i$ . Ako mijenjamo, primjerice, vrijednost bita u registru **DDRD** na poziciji bita  $i = 5$ , tada se promijenjena vrijednost bita odnosi na konfiguraciju pina PD5.

Vrijednost bita na poziciji  $i$  ( $i = 0, 1, \dots, 7$ ) u registru **DDRx** određuje hoće li pin na poziciji  $i$  biti ulazni ili izlazni prema pravilima:

- ako je bit na poziciji  $i$  u registru **DDRx** jednak 0, tada će pin na poziciji  $i$  biti konfiguriran kao ulazni pin,
- ako je bit na poziciji  $i$  u registru **DDRx** jednak 1, tada će pin na poziciji  $i$  biti konfiguriran kao izlazni pin.

Na primjer, ako je **DDRB** =  $0xF0 = 0b11110000$ <sup>1</sup>, tada su gornja četiri pina na portu B izlazni pinovi, a donja četiri pina na portu B ulazni pinovi (tablica 4.1).

---

<sup>1</sup>0x u programskom jeziku C/C++ označava heksadecimalni zapis broja, a 0b binarni zapis broja.

Tablica 4.1: Konfiguracija pinova na portu B sa sadržajem registra DDRB = 0xF0 = 0b1111000

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

## 4.1 Digitalni izlazi mikroupravljača ATmega32U4

Ako je bit na poziciji  $i$  u registru DDRx jednak 1, tada će pin na poziciji  $i$  biti konfiguriran kao izlazni pin. Pin koji je konfiguriran kao izlaz može biti u dvama stanjima:

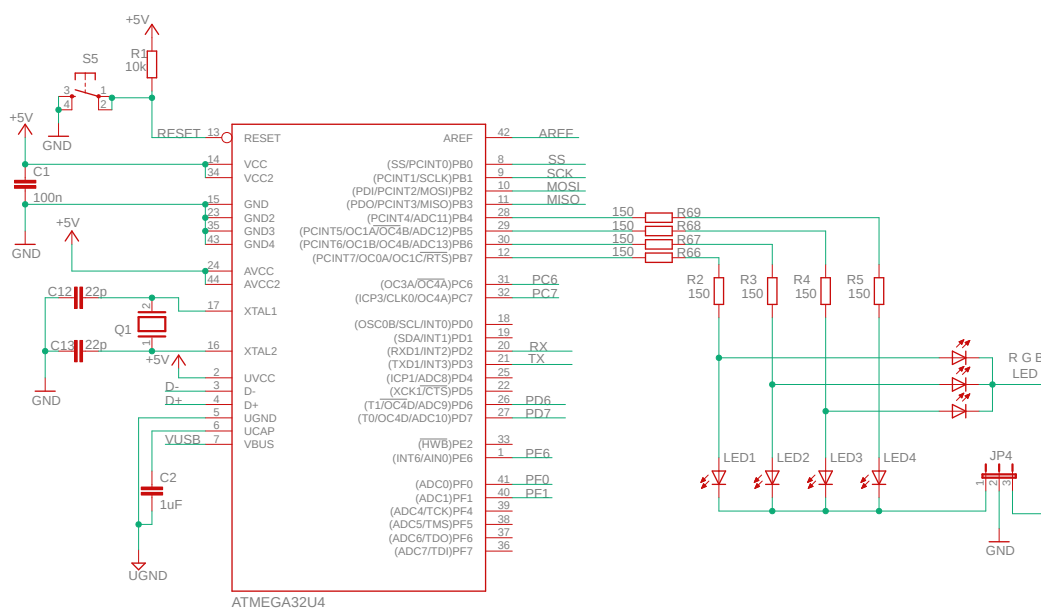
- nisko stanje ili stanje logičke nule - stanje koje odgovara naponu 0 V,
- visoko stanje ili stanje logičke jedinice - stanje koje odgovara naponu 5 V.

Nisko ili visoko stanje na izlaznom pinu određujemo registrom PORTx prema pravilima:

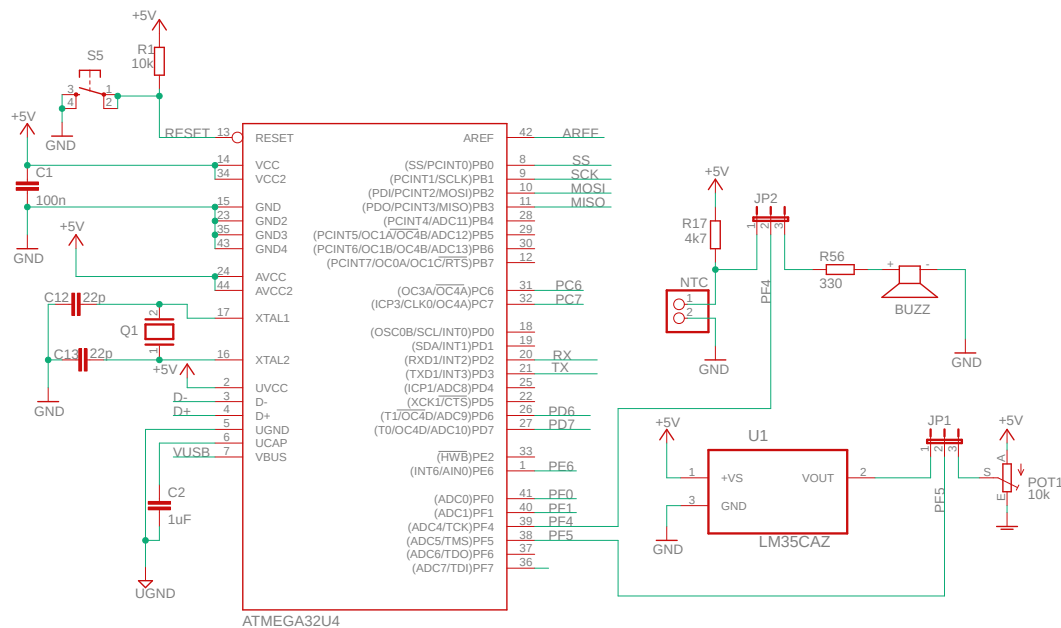
- ako je bit na poziciji  $i$  u registru PORTx jednak 0, tada će izlazni pin na poziciji  $i$  biti u niskom stanju (stanju logičke nule),
- ako je bit na poziciji  $i$  u registru PORTx jednak 1, tada će izlazni pin na poziciji  $i$  biti u visokom stanju (stanju logičke jedinice).

### 4.1.1 Vježbe - digitalni izlazi mikroupravljača ATmega32U4

Digitalne izlaze mikroupravljača ATmega32U4 testirat ćemo pomoću četiriju LED dioda i jedne zujalice. Sheme spajanja LED dioda i zujalice na digitalne izlaze mikroupravljača ATmega32U4 prikazane su na slikama 4.1 i 4.2.



Slika 4.1: Shema spajanja LED dioda na mikroupravljač ATmega32U4

Slika 4.2: Shema spajanja zujalice (engl. *buzzer*) na mikroupravljač ATmega32U4

LED diode i zujalicu moguće je spojiti na bilo koji digitalni pin. Sheme sa slika 4.1 i 4.2 usklađene su s razvojnim okruženjem prikazanim na slici 3.1. Pri spajanju LED dioda i zujalice na digitalne izlaze korišteni su otpornici serijski spojeni s LED diodama i zujalicom. Razlog tome jest strujna zaštita digitalnog pina. Struja digitalnog pina ne smije biti veća od 40 mA prema [1]. Vrijednost otpora otpornika iznosi 300  $\Omega$  za LED diode i 330  $\Omega$  za zujalicu.

Na digitalne izlaze možemo spojiti bipolarne i unipolarne tranzistore, releje s maksimalnom upravljačkom strujom od 40 mA, optičke sprežnike (engl. *optocoupler*), dijke, trijke i ostale digitalne aktuatorne, uz uvjet da se ne premaši maksimalna struja digitalnog pina.

Za detalje oko konfiguracije digitalnih izlaza pogledajte tablicu 10-1 u literaturi [1] na stranici 69. Programsko razvojno okruženje *Atmel Studio 7* ima definirana imena registara DDRx i PORTx (x = B, C, D, E, F) te se konfiguracija pinova svodi na dodjeljivanje vrijednosti u definirana imena registara. Na primjer, ako želimo da svi pinovi porta D budu izlazni pinovi u programskom razvojnom okruženju *Atmel Studio 7*, napisat ćemo `DDRD = 0xFF`; u heksadecimalnom zapisu ili `DDRD = 0b11111111`; u binarnom zapisu. Općenito vrijedi da su imena registara koja se koriste u literaturi [1] jednaka imenima definiranim u programskom razvojnom okruženju *Atmel Studio 7*.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku `Digital Output.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati *Vaše Ime i Prezime* ne koristeći pritom diakritičke znakove. Na primjer, ako je *Vaše ime* Iвица Ivić, datoteka koju ćete stvoriti zvat će se *Ivica Ivic*. Datoteku `Digital Output.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atstln` u datoteci `\\Digital Output\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju *Digitalni izlazi mikroupravljača ATmega32U4*. Vježbe ćemo pisati u datoteke s ekstenzijom `*.cpp`. Budući da će svaka datoteka u koju ćemo pisati vježbe sadržavati funkciju `main()`<sup>2</sup>, za svaku vježbu potrebno je napraviti sljedeće korake (slika 4.3):

1. u projektnom stablu odaberite datoteku s ekstenzijom `*.cpp` (npr. `vjezba411.cpp`) i na njoj pritisnite desni gumb miša te odaberite `Properties`,

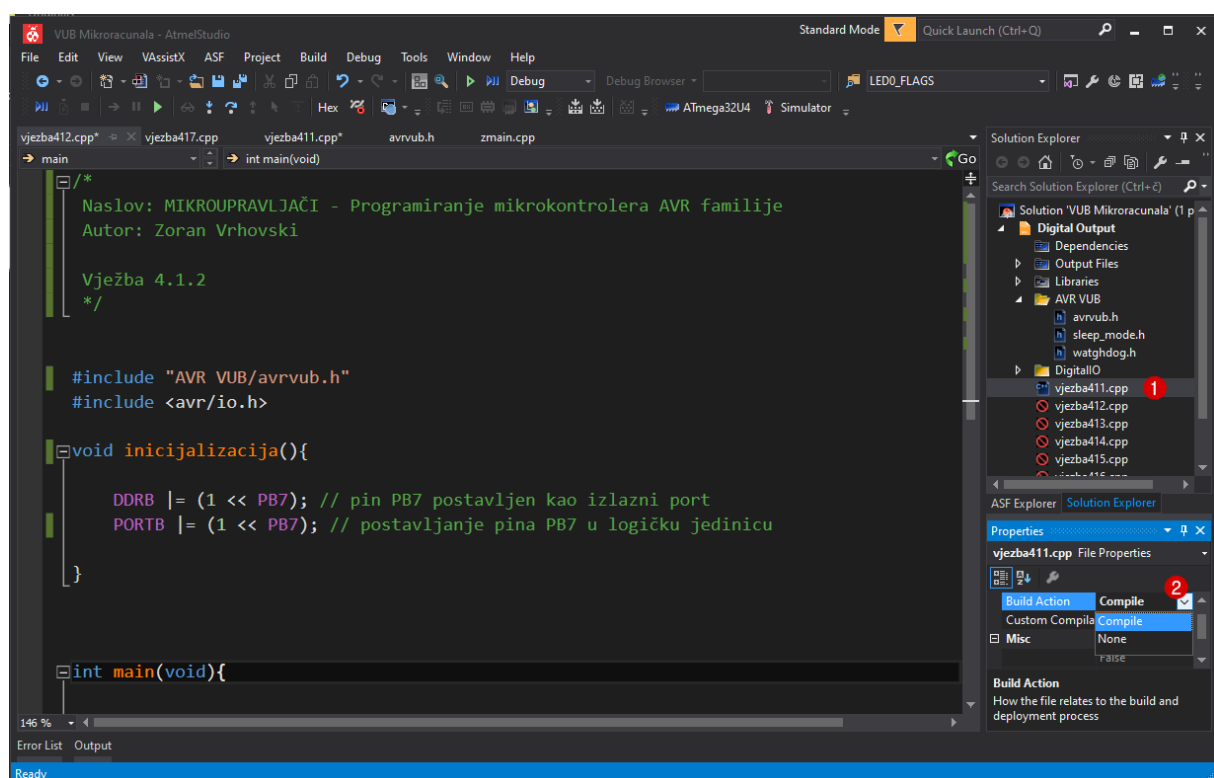
<sup>2</sup>U programskom jeziku C/C++ poznato je da program može imati samo jednu funkciju `main()`.

- 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*.

Na ovaj način osigurali ste da se samo jedna vježba prevodi u strojni kod. Ukoliko je na više vježbi u polju *Build Action* odabrana opcija *Compile*, prevoditelj će javiti grešku da u programskom kodu postoji višestruka definicija funkcije `main()`, što je prema pravilima programskog jezika C/C++ nedopustivo.

Ovaj postupak objašnjen je samo na ovom mjestu i dalje se više neće spominjati te će se pri prelasku s vježbe na vježbu samo napomenuti da se omogući ili onemogući prevođenje vježbe.

U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.



Slika 4.3: Odabir datoteke koja će se prevoditi u strojni kod



### Vježba 4.1.1

Napravite program koji će uključiti plavu LED diodu na razvojnom okruženju s mikroupravljačem ATmega32U4. Shema spajanja plave LED diode na digitalni izlaz PB7 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

U projektnom stablu otvorite datoteku `vjezba411.cpp`. Omogućite samo prevođenje datoteke `vjezba411.cpp`. Početni sadržaj datoteke `vjezba411.cpp` prikazan je programskim kodom 4.1. Objasnimo sada nepoznate linije programskog koda 4.1:

- `#include "AVR VUB/avrvub.h"` - zaglavlje `avrvub.h` s definiranim makronaredbama za

konfiguraciju mikroupravljača. Zaglavlje `avrvub.h` napisano je od strane korisnika te se u programski kod uključuje naredbom `#include ""`,

- `#include <avr/io.h>` - zaglavlje `io.h` s definiranim makronaredbama i funkcijama za manipulaciju s digitalnim ulazima i izlazima. Ovo zaglavlje u obzir uzima mikroupravljač za koji je stvoren projekt u programskom razvojnom okruženju *Atmel Studio 7*. Zaglavlje `io.h` razvijeno je u tvrtki *Microchip* te se u programski kod uključuje naredbom `#include <>`.
- `void inicijalizacija()` - inicijalizacijska funkcija koju programeri mikroupravljača često koriste za definiranje i objedinjavanje početnih postavki mikroupravljača. Naziv inicijalizacijske funkcije proizvoljan je.

Programski kod 4.1: Početni sadržaj datoteke `vjezba411.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {

}

int main(void) {

    inicijalizacija();

    return 0;
}
```

Naš je zadatak napisati tijelo funkcije `inicijalizacija()` kako bi se uključila plava LED dioda spojena na digitalni pin PB7. LED dioda elektronička je naprava kojoj je potreban dovoljan napon kako bi kroz sebe provela struju i pri tome emitirala svjetlost. Iz tog razloga pin na koji je spojena dioda mora se konfigurirati kao izlazni pin. Prema tome, u registar `DDRB` potrebno je na mjestu bita 7 upisati 1, a na sva ostala mjesta 0. Vrijednost konstante koju je potrebno upisati u registar `DDRB` jest `0b1000000` binarno ili `0x80` heksadecimalno. Češće ćemo koristiti heksadecimalni zapis, no čitatelju prepuštam da sam odluči koji mu je pristup prihvatljiviji. Kada je digitalni pin PB7 definiran kao izlaz, u registar `PORTB` na mjestu bita 7 potrebno je upisati 1 kako bi se taj pin postavio u visoko stanje te kako bi se na taj način uključila plava LED dioda. Prema tome, sadržaj registra `PORTB` bit će `0x80`.

U tijelo funkcije `inicijalizacija()` upišite gore navedene konstante.

Programski kod 4.2: Funkcija inicijalizacije mikroupravljača - prvi način

```
void inicijalizacija() {

    DDRB = 0b10000000;
    PORTB = 0x80;

}
```

Tijelo funkcije `inicijalizacija()` mora odgovarati programskom kodu 4.2. Sada je datoteku `vjezba411.cpp` potrebno prevesti u strojni kod sukladno uputama iz poglavlja *Razvojno okruženje Atmel Studio 7*. Strojni kod pomoću softvera *eXtreme Burner - AVR* snimite na mikroupravljač ATmega32U4 prema uputama iz poglavlja *Snimanje i pokretanje strojnog koda na mikroupravljaču ATmega32U4*. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Inicijalizacija 4.2 samo je jedan od načina inicijalizacije mikroupravljača i koristi se ako su poznate funkcije ostalih pinova ili ako se ostali pinovi ne koriste. U praksi se češće javlja

problem u kojem je potrebno ciljani pin postaviti kao izlazni, a da se konfiguracija ostalih pinova ne mijenja. U tu svrhu potrebno je koristiti bitovne operatore i operator posmaka (programski kod 4.3).

Programski kod 4.3: Funkcija inicijalizacije mikroupravljača - drugi način

```
void inicijalizacija() {
    DDRB |= (1 << PB7);
    PORTB |= (1 << PB7);
}
```

U programskom kodu 4.3 korištena je definirana konstanta `PB7` i njezina vrijednost iznosi 7, odnosno jednaka je poziciji pina na portu B. Konstante su definirane i za ostale pinove na ostalim portovima i dostupne su u datoteci `vjezba411.cpp` putem uključenih zaglavlja `io.h` i `avrub.h`. U programskom okruženju *Atmel Studio 7* iznad konstante `PB7` pritisnite desni gumb miša te odaberite `Goto Implementation`. Otvorit će se zaglavlje u kojem su definirane konstante za sve portove. Odvojite vremena i pogledajte ostali sadržaj otvorenog zaglavlja. Opcija `Goto Implementation` vrlo je korisna i preporučuje se njezino često korištenje za sve definirane konstante i makronaredbe. Definirane konstante i makronaredbe prepoznat ćete po ljubičastoj boji.

U tablici 4.2 prikazani su bitovni operator i operator posmaka korišteni za konfiguriranje izlaznog pina. Broj 1 posmiče se ulijevo za 7 mjesta naredbom `1 << PB7`. Na taj smo način broj 1 pozicionirali upravo ispod bita broj 7. Pretpostavimo da su stanja bitova registra `DDRB` nepoznata i ta stanja označit ćemo s  $x$ . Ako želimo da pin `PB7` bude izlazni pin, tada na mjestu bita broj 7 u registru `DDRB` moramo postaviti 1. To ćemo ostvariti tako da koristimo `ILI` operator. Stanje registra `DDRB` podvrgnemo bitovnom `ILI` operatoru s konstantom koja je dobivena naredbom `1 << PB7` te rezultat spremimo u registar `DDRB`. Na taj smo način samo ciljani sedmi bit postavili u 1, a ostali bitovi ostali su nepromijenjeni.

Tablica 4.2: Bitovni operator i 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
<code>1 &lt;&lt; PB7</code>	1	0	0	0	0	0	0	0
Stanje registra <code>DDRB</code>	$x$	$x$	$x$	$x$	$x$	$x$	$x$	$x$
<code>DDRB  = (1 &lt;&lt; PB7)</code>	1	$x$	$x$	$x$	$x$	$x$	$x$	$x$

Tijelo funkcije `inicijalizacija()` promijenite tako da odgovara programskom kodu 4.3. Prevedite datoteku `vjezba411.cpp` u strojni kod i snimite ga na mikroupravljač `ATmega32U4`. Testirajte program na razvojnom okruženju s mikroupravljačem `ATmega32U4`.

Programski kod 4.4: Funkcija inicijalizacije mikroupravljača - treći način

```
void inicijalizacija() {
    output_port(DDRB, PB7);
    set_port(PORTB, PB7, 1);
}
```

Treći je način inicijalizacije najjednostavniji. Autor udžbenika napisao je makronaredbe koje su intuitivne i jednostavne za korištenje. Makronaredbe se nalaze u zaglavlju `avrub.h`, a u programskom kodu 4.4 koristimo sljedeće:



- `output_port(DDRx, pin)` - makronaredba koja kao argumente prima registar DDRx i poziciju pina kojeg želimo postaviti kao izlazni pin,
- `set_port(PORTx, pin, stanje)` - makronaredba koja kao argumente prima registar PORTx, poziciju pina koji želimo postaviti u visoko ili nisko stanje te željeno stanje pina (0 - nisko stanje, 1 - visoko stanje).

Prednost ovog načina inicijalizacije ogleda se u tome što nije potrebno voditi brigu o konfiguraciji ostalih pinova jer makronaredbe same vode brigu o tome. Čitatelju se prepušta na volju odabir načina konfiguracije. Preporučujem drugi način konfiguracije jer daje jasnu sliku o tome što radite.

Tijelo funkcije `inicijalizacija()` promijenite tako da odgovara programskom kodu 4.4. Prevedite datoteku `vjezba411.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba411.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.2

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 tri puta uključiti i isključiti plavu LED diodu u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja plave LED diode na digitalni izlaz PB7 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

U projektnom stablu otvorite datoteku `vjezba412.cpp`. Omogućite samo prevođenje datoteke `vjezba412.cpp`. Početni sadržaj datoteke `vjezba412.cpp` prikazan je programskim kodom 4.5.

Programski kod 4.5: Početni sadržaj datoteke `vjezba412.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {
    DDRB |= (1 << PB7); // pin PB7 izlazni pin
    PORTB |= (1 << PB7); // pin PB7 početno u visokom stanju
}

int main(void) {
    inicijalizacija(); // inicijalizacija mikroupravljača
    return 0;
}
```

U vježbi je zadano da se plava LED dioda mora uključiti i isključiti ukupno tri puta i to u razmaku od jedne sekunde. Postavlja se pitanje kako je uopće moguće da se LED dioda uključuje i isključuje u razmaku od jedne sekunde. U tu svrhu potrebno je koristiti funkcije za kašnjenje. U programski kod 4.5 upišite naredbu `#include <util/delay.h>` kako bi omogućili korištenje funkcija za kašnjenje. U zaglavlju `delay.h` nalaze se sljedeće korisne funkcije:

- `_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  $\mu$ s.

Navedene funkcije doslovno zaustave rad mikroupravljača na zadano vrijeme, što u principu i nije dobro jer se u tom vremenu onemogućuje obrada podataka s vanjskih senzora. Za sada je ovo jedino rješenje koje će nam omogućiti uključivanje i isključivanje plave LED diode u razmaku od jedne sekunde. Funkcije `_delay_ms(double)` i `_delay_us(double)` poželjno je izbjegavati u ozbiljnijim programima ili ih eventualno koristiti na ispravan način.

Kako bi funkcije `_delay_ms(double)` i `_delay_us(double)` mogle ostvariti vremensko kašnjenje, potrebno je napraviti ispravnu konfiguraciju *Fuse* bitova, kao što smo to pokazali u prošlom poglavlju. U programskom okruženju *Atmel Studio 7* frekvencija se definira konstantom `F_CPU`. Važno je znati da se konstantom `F_CPU` ne odabire frekvencija mikroupravljača, već se svim bibliotekama u razvojnom okruženju *Atmel Studio 7* daje do znanja da frekvencija mikroupravljača iznosi `F_CPU`. Frekvencija vanjskog oscilatora spojenog na mikroupravljač iznosi 16 MHz pa je i konstantu `F_CPU` potrebno namjestiti na 16 MHz.

U zaglavlju `avr/vub.h` nalazi se naredba `#define F_CPU 16000000u1` kojom se prevoditelju ukazuje da frekvencija rada mikroupravljača iznosi 16 MHz. Konstantu `F_CPU` možete definirati bilo gdje u programskom kodu, ali svakako prije korištenja funkcija za kašnjenje. Ako ne definirate konstantu `F_CPU`, prevoditelj pretpostavlja da frekvencija rada mikroupravljača iznosi 1 MHz. Usklađenost frekvencija na hardverskoj i softverskoj razini lako je testirati na razvojnom okruženju s mikroupravljačem ATmega32U4.

Sada kada poznamo funkcije kašnjenja, u programski kod 4.5 u funkciju `main()` ispod poziva funkcije `inicijalizacija()`; unesite sljedeći niz naredbi koje će omogućiti uključivanje i isključivanje plave LED diode u razmaku od jedne sekunde ukupno tri puta:

- `_delay_ms(1000)`; - funkcija koja omogućuje kašnjenje od 1000 ms, što je jedna sekunda. U inicijalizacijskoj je funkciji postavljeno da je plava LED dioda u početku rada mikroupravljača uključena. Prema tome, prvo je potrebno sačekati jednu sekundu, a zatim ugasiti plavu LED diodu.
- `set_port(PORTB,PB7,0)`; - makronaredba koja isključuje plavu LED diodu.
- `_delay_us(1000000)`; - funkcija koja omogućuje kašnjenje od 1000000  $\mu$ s, što je jedna sekunda. Nakon što je plava LED dioda bila isključena jednu sekundu, sada ju je potrebno uključiti.
- `set_port(PORTB,PB7,1)`; - makronaredba koja uključuje plavu LED diodu.
- `_delay_ms(1000)`; - funkcija koja omogućuje kašnjenje od 1000 ms, što je jedna sekunda.
- `PORTB &= 0x7F`; - naredba kojom isključujemo plavu LED diodu pomoću bitovnog I operatora i maske `0x7F = 0b01111111`.
- `_delay_us(1000000)`; - funkcija koja omogućuje kašnjenje od 1000000  $\mu$ s, što je jedna sekunda.
- `set_port(PORTB,PB7,1)`; - makronaredba koja uključuje plavu LED diodu.
- `_delay_ms(1000)`; - funkcija koja omogućuje kašnjenje od 1000 ms, što je jedna sekunda.
- `PORTB &= ~(1 << PB7)`; - naredba kojom isključujemo plavu LED diodu pomoću bitovnog I operatora, operatora posmaka i operatora komplementa.

Prevedite datoteku `vjezba412.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Zašto plava LED dioda ne izmjenjuje svoje stanje nakon naredbe `PORTB &= ~(1 << PB7)`;

Zatvorite datoteku `vjezba412.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.3

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti beskonačno uključivanje i isključivanje plave LED diode u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja plave LED diode na digitalni izlaz PB7 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

U projektnom stablu otvorite datoteku `vjezba413.cpp`. Omogućite samo prevođenje datoteke `vjezba413.cpp`. Početni sadržaj datoteke `vjezba413.cpp` prikazan je programskim kodom 4.6.

Programski kod 4.6: Početni sadržaj datoteke `vjezba413.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {
    DDRB |= (1 << PB7); // pin PB7 izlazni pin
    PORTB |= (1 << PB7); // pin PB7 početno u visokom stanju
}

int main(void) {
    inicijalizacija(); // inicijalizacija mikroupravljača

    return 0;
}
```

U vježbi je zadano da se plava LED dioda mora neprestano uključivati i isključivati u razmaku od jedne sekunde. Uvjet zadatka bit će zadovoljen dok mikroupravljač ima napajanje i dok je RESET pin u visokom stanju. Kako bismo ostvarili beskonačnu izmjenu visokog i niskog stanja na plavoj LED diodi, potrebno je koristiti beskonačnu petlju. U programskom jeziku C na raspolaganju imamo tri vrste petlji. Mi ćemo koristiti petlju `while`. U programskom kodu 4.6 u funkciju `main()` ispod poziva funkcije `inicijalizacija()`; napišite programski kod 4.7.

Programski kod 4.7: Beskonačna `while` petlja

```
while(1) {
    // ovdje pišemo blok naredbi koje izvodi mikroupravljač
    // sve dok ima napajanje i dok je {RESET} pin u visokom stanju
}
```

Petlja `while` u programskom kodu 4.7 kao uvjet izvođenja ima broj različit od nule, što je istinit uvjet izvođenja, pa je prema tome ova petlja beskonačna. Unutar bloka naredbi beskonačne `while` petlje sada je potrebno upisati niz naredbi koje će osigurati beskonačnu izmjenu stanja plave diode svaku sekundu. U blok naredbi beskonačne `while` upišite sljedeće dvije naredbe:

- `_delay_ms(1000)`; - funkcija koja omogućuje kašnjenje od 1000 ms, što je jedna sekunda,
- `toggle_port(PORTB,PB7)`; - makronaredba koja mijenja stanje izlaznog pina, a kao argumente prima registar PORTx u kojem treba napraviti promjenu stanja pina i poziciju pina kojem je potrebno promijeniti stanje. Ako je stanje izlaznog pina PB7 bilo visoko, nakon izvođenja makronaredbe `toggle_port(PORTB,PB7)`; bit će nisko i obratno.

U svakom se prolazu kroz blok naredbi beskonačne `while` petlje prethodne dvije naredbe

ponovno izvode. U `while` petlji čeka se jednu sekundu i nakon toga se stanje mijenja, što će dovesti do beskonačne izmjene stanja plave LED diode.

Prevedite datoteku `vjezba413.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Pokušajte ubrzati izmjenu stanja plave LED diode.

Zatvorite datoteku `vjezba413.cpp` i onemogućite prevođenje ove datoteke.



#### Vježba 4.1.4

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 100 ms i to redoslijedom crvena → žuta → zelena → plava → crvena → ... . Za izmjenu stanja LED diode koristite makronaredbu `toggle_port`. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

U projektnom stablu otvorite datoteku `vjezba414.cpp`. Omogućite samo prevođenje datoteke `vjezba414.cpp`. Početni sadržaj datoteke `vjezba414.cpp` prikazan je programskim kodom 4.8.

Programski kod 4.8: Početni sadržaj datoteke `vjezba414.cpp`

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

void inicijalizacija() {
    // PB7, PB6, PB5 i PB4 izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    PORTB |= (1 << PB4); // postavljanje PB4 u visoko stanje
}

int main(void) {
    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja
    }
    return 0;
}
```

U vježbi je potrebno ostvariti „trčanje” LED dioda na način da trči uključena LED dioda. Princip je sljedeći:

- Uključite crvenu LED diodu, a zatim isključite plavu LED diodu. Pričekajte 100 ms.
- Uključite žutu LED diodu, a zatim isključite crvenu LED diodu. Pričekajte 100 ms.
- Uključite zelenu LED diodu, a zatim isključite žutu LED diodu. Pričekajte 100 ms.
- Uključite plavu LED diodu, a zatim isključite zelenu LED diodu. Pričekajte 100 ms.
- Ponovite prethodna četiri koraka.

Obratite pažnju na tijelo funkcije `inicijalizacija()` u programskom kodu 4.8. Digitalni pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi tako što je u registar `DDRB` na mjesto bitova 7, 6, 5 i 4 upisan broj 1. U početnom je stanju crvena LED dioda uključena (`PORTB |= (1 << PB4);`). U programskom kodu 4.8 u blok naredbi beskonačne `while` petlje upišite sljedeće naredbe:

- `_delay_ms(100);` - kašnjenje od 100 ms.
- `toggle_port(PORTB,PB4);` - promijeni stanje crvene LED diode. Prethodno je bila uključena, a nakon ove naredbe bit će isključena.
- `toggle_port(PORTB,PB5);` - promijeni stanje žute LED diode. Prethodno je bila isključena, a nakon ove naredbe bit će uključena.
- `_delay_ms(100);` - kašnjenje od 100 ms.
- `toggle_port(PORTB,PB5);` - promijeni stanje žute LED diode. Prethodno je bila uključena, a nakon ove naredbe bit će isključena.
- `toggle_port(PORTB,PB6);` - promijeni stanje zelene LED diode. Prethodno je bila isključena, a nakon ove naredbe bit će uključena.
- Pokušajte samostalno postići kašnjenje programa od 100 ms, uključiti zelenu LED diodu, a zatim isključiti žutu LED diodu na temelju prethodnih koraka.
- `_delay_ms(100);` - kašnjenje od 100 ms.
- `toggle_port(PORTB,PB7);` - promijeni stanje plave LED diode. Prethodno je bila uključena, a nakon ove naredbe bit će isključena.
- Pokušajte samostalno promijeniti stanje crvene LED diode.

Prethodni blok naredbi može se napisati na razne načine. Pokušajte umjesto makronaredbe `toggle_port` koristiti makronaredbu `set_port`.

Prevedite datoteku `vjezba414.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba414.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.5

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 100 ms i to redosljedom crvena → žuta → zelena → plava → crvena → ... . Za izmjenu stanja LED diode koristite `for` petlju. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

U projektnom stablu otvorite datoteku `vjezba415.cpp`. Omogućite samo prevođenje datoteke `vjezba415.cpp`. Početni sadržaj datoteke `vjezba415.cpp` prikazan je programskim kodom 4.9.

Programski kod 4.9: Početni sadržaj datoteke vjezba415.cpp

```

#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {
}

int main(void) {

    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) { // beskonačna petlja

        for (int i = 4; i <= 7; i++) {

            _delay_ms(100);

            PORTB &= ~(1 << i);

            if (i != 7) {
                PORTB |= (1 << (i+1));
            }
            else {
                PORTB |= (1 << PB4);
            }
        }
    }
    return 0;
}

```

U vježbi je ponovno potrebno ostvariti „trčanje” LED dioda, ali pomoću `for` petlje. Ovo je teža izvedba vježbe pa je stoga rješenje prikazano u programskom kodu 4.9. U tijelu funkcije `inicijalizacija()` u programskom kodu 4.9 konfigurirajte digitalne pinove PB7, PB6, PB5 i PB4 kao izlazne tako da u registar `DDRB` na mjesto bitova 7, 6, 5 i 4 upišete 1. Konfiguraciju izlaznih pinova ostvarite pomoću makronaredbe `output_port`. Početno stanje crvene LED diode postavite u visoko stanje pomoću makronaredbe `set_port`.

Pogledajmo sada blok naredbi `while` petlje u programskom kodu 4.9. U bloku naredbi `while` petlje nalazi se `for` petlja koja se izvodi ukupno četiri puta. Brojač `for` petlje `i` poprima vrijednosti 4, 5, 6 i 7. Primijetite da su to pozicije digitalnih pinova PB4, PB5, PB6 i PB7. Unutar `for` petlje nalazi se kašnjenje od 100 ms. Naredbom `PORTB &= ~(1 << i);` isključuje se prethodno uključena LED dioda. Uvjetnim `if` blokom provjeravamo je li brojač `for` petlje `i` različit od 7. Ako je `i` različit od 7, uključujemo sljedeću LED diodu naredbom `PORTB |= (1 << (i+1));`. Onog trenutka kada brojač `for` petlje `i` poprimi vrijednost 7 uključuje se crvena LED dioda naredbom `PORTB |= (1 << PB4);` kao početno stanje za sljedeća četiri prolaza kroz `for` petlju.

Prevedite datoteku `vjezba415.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba415.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.6

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti 50 ciklusa „trčanja” svih LED dioda svakih 100 ms i to redoslijedom crvena → žuta → zelena → plava → crvena → ... . Nakon što 50 ciklusa trčanja LED dioda završi, trajno uključite zujalicu

spojenu na pin PF4. Sheme spajanja LED dioda i zujalice na digitalne izlaze mikroupravljača ATmega32U4 prikazane su na slikama 4.1 i 4.2. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

U projektnom stablu otvorite datoteku `vjezba416.cpp`. Omogućite samo prevođenje datoteke `vjezba416.cpp`. Početni sadržaj datoteke `vjezba416.cpp` prikazan je programskim kodom 4.10.

Programski kod 4.10: Početni sadržaj datoteke `vjezba416.cpp`

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

void inicijalizacija() {
    // PB7, PB6, PB5 i PB4 izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    PORTB |= (1 << PB4); // postavljanje PB7 u visoko stanje

    DDRF |= (1 << PF4); //pin PF4 za Buzzer
}

int main(void) {

    inicijalizacija(); // inicijalizacija mikroupravljača

    for (int i = 0; i < 50; i++) { // petlja koja se izvodi 50 puta
    }
    // ovdje ugasiti crvenu LED diodu

    // ovdje uključiti Buzzer

    return 0;
}
```

U vježbi je ponovno potrebno ostvariti „trčanje” LED dioda, ali konačan broj puta. U programskom kodu 4.10 u blok naredbi `for` petlje kopirajte blok naredbi `while` petlje iz datoteke `vjezba414.cpp`. U praksi se često ponavljaju dijelovi programskog koda pa je uobičajeno kopirati gotovi programski kod u trenutni projekt koji radite.

Prevedite datoteku `vjezba416.cpp` u strojni kod. Prilikom prevođenja prevoditelj je u pokazniku statusnih poruka javio grešku koja upućuje na izostanak deklaracije funkcije `_delay_ms`. Uključite zaglavlje u kojem je deklarirana funkcija `_delay_ms`.

Nakon što `for` petlja završi ciklus od 50 koraka, potrebno je isključiti crvenu LED diodu koja je ostala uključena. Ispod bloka naredbi `for` petlje isključite crvenu LED diodu makronaredbom `set_port`. Zatim, potrebno je trajno uključiti zujalicu spojena na pin PF4, što možete napraviti makronaredbom `set_port(PORTF, PF4, 1);`.

Prevedite datoteku `vjezba416.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba416.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.7

Napravite program kojim će razvojno okruženje s mikroupravljačem ATmega32U4 reproducirati zvučnu signalizaciju pomoću zujalice i funkcije `BUZZ(double trajanje, int frekvencija)`. Shema spajanja zujalice na digitalni izlaz PF4 mikroupravljača ATmega32U4 prikazana je na

slici 4.2.

U projektnom stablu otvorite datoteku `vjezba417.cpp`. Omogućite samo prevođenje datoteke `vjezba417.cpp`. Početni sadržaj datoteke `vjezba417.cpp` prikazan je programskim kodom 4.11.

Programski kod 4.11: Početni sadržaj datoteke `vjezba417.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>

int main(void){

    while (1) { // beskonačna petlja

        //super mario sound - isječak
        BUZZ(0.1, 660); _delay_ms(150);
        BUZZ(0.1, 660); _delay_ms(300);
        BUZZ(0.1, 660); _delay_ms(300);
        BUZZ(0.1, 510); _delay_ms(100);
        BUZZ(0.1, 660); _delay_ms(300);
        BUZZ(0.1, 770); _delay_ms(550);
        BUZZ(0.1, 770); _delay_ms(575);
        BUZZ(0.1, 510); _delay_ms(450);
        BUZZ(0.1, 380); _delay_ms(400);
        BUZZ(0.1, 320); _delay_ms(500);
        BUZZ(0.1, 440); _delay_ms(300);
        BUZZ(0.08, 480); _delay_ms(330);
        BUZZ(0.1, 450); _delay_ms(150);
        BUZZ(0.1, 430); _delay_ms(300);
        BUZZ(0.1, 380); _delay_ms(200);
        BUZZ(0.08, 660); _delay_ms(200);
        BUZZ(0.1, 760); _delay_ms(150);
        BUZZ(0.1, 860); _delay_ms(300);
        BUZZ(0.08, 700); _delay_ms(150);
        BUZZ(0.05, 760); _delay_ms(350);
        BUZZ(0.08, 660); _delay_ms(300);
        BUZZ(0.08, 520); _delay_ms(150);
        BUZZ(0.08, 580); _delay_ms(150);
        BUZZ(0.08, 480); _delay_ms(1000);
    }
    return 0;
}
```

Zvučna signalizacija često je potrebna u praksi kako bi ukazala na neki događaj. U ovoj vježbi koristit ćemo funkciju `BUZZ(double trajanje, int frekvencija)` koja prima dva argumenta:

1. `trajanje` - realni broj dvostruke preciznosti koji predstavlja trajanje zvučnog signala u sekundama,
2. `frekvencija` - cijeli broj koji predstavlja frekvenciju zvučnog signala u Hz.

Zujalica je spojena na digitalni izlaz PF4. Na shemi sa slike 4.2 primijetite da je zujalica na mikroupravljač spojena preko kratkospojnika (engl. *jumper*) JP2 između trnova 2 i 3. Ako na razvojnom okruženju s mikroupravljačem ATmega32U4 nema kratkospojnika JP2, zujalica neće raditi. Zujalicu možemo spojiti na bilo koji digitalni pin.

Definicija funkcije `BUZZ()` nalazi se u datoteci `AVR_VUB/avrvub.h`. Proučite definiciju ove funkcije. U datoteci `AVR_VUB/avrvub.h` povrh deklaracije funkcije `BUZZ` nalazi se dio



programskog koda koji omogućuje definiranje digitalnog pina na koji je spojena zujalica. Definiranje digitalnog pina prikazano je programskim kodom 4.12.

Programski kod 4.12: Definiranje digitalnog pina PF4 na koji je spojena zujalica

```
#define BUZZER_PORT PORTF
#define BUZZER_DDR DDRF
#define BUZZER_PIN PF4
```

Pojasnimo definiranje digitalnog pina na koji je spojena zujalica kako je prikazano u programskom kodu 4.12:

- `#define BUZZER_PORT PORTF` - podatkovni registar koji uključuje i isključuje digitalni pin PF4,
- `#define BUZZER_DDR DDRF` - registar smjera podataka koji definira digitalni pin PF4 kao izlazni pin,
- `#define BUZZER_PIN PF4` - pozicija digitalnog pina PF4.

Ako bi zujalica bila spojena na digitalni pin PD7, programski kod 4.12 trebalo bi promijeniti na način prikazan u programskom kodu 4.13.

Programski kod 4.13: Definiranje digitalnog pina PD7 na koji je spojena zujalica

```
#define BUZZER_PORT PORTD
#define BUZZER_DDR DDRD
#define BUZZER_PIN PD7
```

Prevedite datoteku `vjezba417.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Sada pokušajte napraviti vlastitu zvučnu signalizaciju. Ponovno prevedite datoteku `vjezba417.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba417.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.1.8

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 250 ms i to redosljedom plava → zelena → žuta → crvena → plava → ... . Implementaciju rješenja potrebno je izvesti pomoću biblioteke `DigitalIO.h` i klase `DigitalOutput`. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

U projektnom stablu otvorite datoteku `vjezba418.cpp`. Omogućite samo prevođenje datoteke `vjezba418.cpp`. Početni sadržaj datoteke `vjezba418.cpp` prikazan je programskim kodom 4.14.

Programski kod 4.14: Početni sadržaj datoteke `vjezba418.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "DigitalIO/DigitalIO.h"
#include <avr/io.h>
#include <util/delay.h>
```

```

int main(void) {

    DigitalOutput plava(B7); // stvori objekt plava za diodu na pinu PB7

    while (1) {

        plava.toggle(); // promjena stanja plave diode
        _delay_ms(250); // čekaj 250 ms
        // nastaviti dalje
    }
    return 0;
}

```

Autor je za manipulaciju digitalnim izlazima i ulazima izradio biblioteku `DigitalIO.h` u kojoj se nalaze klase `DigitalOutput` i `DigitalInput`. Biblioteka `DigitalIO.h` u programskom je kodu 4.14 uključena pomoću naredbe `#include "DigitalIO/DigitalIO.h"`. Ovaj pristup konfiguracije i manipulacije digitalnim izlazima i ulazima sličan je implementaciji u *Arduino* razvojnom okruženju s kojim ste se možda već susreli.

U ovom zadatku koristi se klasa `DigitalOutput` pa će ona biti detaljnije opisana. U klasi `DigitalOutput` definirana su tri preopterećena konstruktora. Dva konstruktora bitna su za stvaranje objekata koji će se koristiti kao digitalni izlazi. Stvaranje objekata koji su digitalni izlazi moguće je na dva načina:

- `DigitalOutput plava(B7);` - objekt se inicijalizira konstantom  $Xi$  ( $X = B, C, D, E, F; i = 0, 1, 2, 3, 4, 5, 6, 7$ ) gdje  $X$  označava port, a  $i$  pin na portu. Npr. za pin 7 na portu B koristit ćemo konstantu `B7`, a za pin 2 na portu D koristit ćemo konstantu `D2`.
- `DigitalOutput crvena(PB, PB4);` - objekt se inicijalizira konstantama  $X$  i  $Xi$  ( $X = PB, PC, PD, PE, PF; i = 0, 1, 2, 3, 4, 5, 6, 7$ ), gdje  $X$  označava port, a  $Xi$  pin na portu. Npr. za pin 4 na portu B koristit ćemo konstante (`PB, PB4`), a za pin 2 na portu D koristit ćemo konstante (`PD, PD2`).

Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7. Instanciranje objekata za digitalne izlaze na koje su spojene LED diode prikazano je programskim kodom 4.15. Taj kod potrebno je napisati na samom početku `main()` funkcije.

Programski kod 4.15: Instanciranje objekata za digitalne izlaze na koje su spojene LED diode

```

DigitalOutput plava(B7); // stvori objekt plava za diodu na pinu PB7
DigitalOutput zelena(B6); // stvori objekt zelena za diodu na pinu PB6
DigitalOutput zuta(B5); // stvori objekt zuta za diodu na pinu PB5
DigitalOutput crvena(PB, PB4); // stvori objekt crvena za diodu na pinu PB4

```

Primijetite kako u ovom pristupu nema inicijalizacijske funkcije `inicijalizacija()`. Razlog tome jest to što se objekti trebaju tretirati kao i varijable jer egzistiraju samo u bloku u kojem su definirani. Dakle, objekti `plava`, `zelena`, `zuta`, `crvena` dohvatljivi su samo u `main()` funkciji.

Klasa `DigitalOutput` ima brojne metode (funkcije klase) koje se mogu pozvati nad objektima:

- `on()` - metoda uključuje izlazni digitalni pin objekta,
- `off()` - metoda isključuje izlazni digitalni pin objekta,
- `set()` - metoda uključuje izlazni digitalni pin objekta,

- `reset()` - metoda isključuje izlazni digitalni pin objekta,
- `toggle()` - metoda mijenja stanje izlaznog digitalnog pina objekta,
- `disable()` - metoda onemogućuje korištenje izlaznog digitalnog pina objekta,
- `enable()` - metoda omogućuje korištenje izlaznog digitalnog pina objekta,
- `state()` - metoda vraća `bool` stanje izlaznog digitalnog pina objekta.

Metode se nad objektima pozivaju operatorom `.` pa se na primjer plava LED dioda može uključiti naredbom `plava.on()`; . Da bismo ostvarili funkcionalnost traženu u zadatku, potrebno je u beskonačnu petlju upisati sljedeći niz naredaba (koristit će se brojne metode iako je rješenje moguće samo metodom `toggle()`):

- `plava.toggle()`; - promijeni stanje plave LED diode (početno stanje bilo je 0, a nakon navedene naredbe bit će 1),
- `_delay_ms(250)`; - pričekaj 250 ms,
- `plava.toggle()`; - promijeni stanje plave LED diode (prethodno stanje bilo je 1, a nakon navedene naredbe bit će 1),
- `zelena.on()`; - uključi zelenu LED diodu,
- `_delay_ms(250)`; - pričekaj 250 ms,
- `zelena.off()`; - isključi zelenu LED diodu,
- `zuta.set()`; - uključi žutu LED diodu,
- `_delay_ms(250)`; - pričekaj 250 ms,
- `zuta.reset()`; - isključi žutu LED diodu,
- `crvena.on()`; - uključi crvenu LED diodu,
- `_delay_ms(250)`; - pričekaj 250 ms,
- `crvena.off()`; // isključi crvenu diodu.

Prevedite datoteku `vjezba418.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba418.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 4.1.2 Zadaci - digitalni izlazi mikroupravljača ATmega32U4

#### Zadatak 4.1.1

Napravite program koji će uključiti zelenu LED diodu na razvojnom okruženju s mikroupravljačem ATmega32U4. Shema spajanja zelene LED diode na digitalni izlaz PB6 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

---

#### Zadatak 4.1.2

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 tri puta uključiti i isključiti zelenu LED diodu u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja zelene LED diode na digitalni izlaz PB6 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

---

#### Zadatak 4.1.3

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti beskonačno uključivanje i isključivanje zelene LED diode svakih 500 ms. Shema spajanja zelene LED diode na digitalni izlaz PB6 mikroupravljača ATmega32U4 prikazana je na slici 4.1.

---

#### Zadatak 4.1.4

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 350 ms i to redoslijedom plava → zelena → žuta → crvena → plava → ... . Za izmjenu stanja LED diode koristite makronaredbu `toggle_port`. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

---

#### Zadatak 4.1.5

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 350 ms i to redoslijedom plava → zelena → žuta → crvena → plava → ... . Za izmjenu stanja LED diode koristite `for` petlju. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

---

### Zadatak 4.1.6

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti 100 ciklusa „trčanja” svih LED dioda svakih 200 ms i to redoslijedom plava → zelena → žuta → crvena → plava → ... Nakon što 100 ciklusa trčanja LED dioda završi, trajno uključite zujalicu spojenu na pin PF4. Sheme spajanja LED dioda i zujalice na digitalne izlaze mikroupravljača ATmega32U4 prikazane su na slikama 4.1 i 4.2. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

---

### Zadatak 4.1.7

Napravite program kojim će razvojno okruženje s mikroupravljačem ATmega32U4 reproducirati zvučnu signalizaciju pomoću zujalice i funkcije `BUZZ(double trajanje, int frekvencija)`. Zujalica mora reproducirati zvuk frekvencije 550 Hz u trajanju od jedne sekunde, a zatim zvuk frekvencije 780 Hz u trajanju od 500 ms. Navedenu je zvučnu signalizaciju potrebno neprestano ponavljati. Shema spajanja zujalice na digitalni izlaz PF4 mikroupravljača ATmega32U4 prikazana je na slici 4.2.

---

### Zadatak 4.1.8

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 300 ms i to redoslijedom crvena → žuta → zelena → plava → crvena → ... . Implementaciju rješenja potrebno je izvesti pomoću biblioteke `DigitalIO.h` i klase `DigitalOutput`. Shema spajanja LED dioda na digitalne izlaze mikroupravljača ATmega32U4 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7.

---

## 4.2 Digitalni ulazi mikroupravljača ATmega32U4

Ako je bit na poziciji  $i$  u registru DDRx jednak 0, tada će pin na poziciji  $i$  biti konfiguriran kao ulazni pin. Pin koji je konfiguriran kao ulaz može biti u dvama stanjima:

- nisko stanje ili stanje logičke nule - stanje koje odgovara naponu 0 V,
- visoko stanje ili stanje logičke jedinice - stanje koje odgovara naponu 5 V.

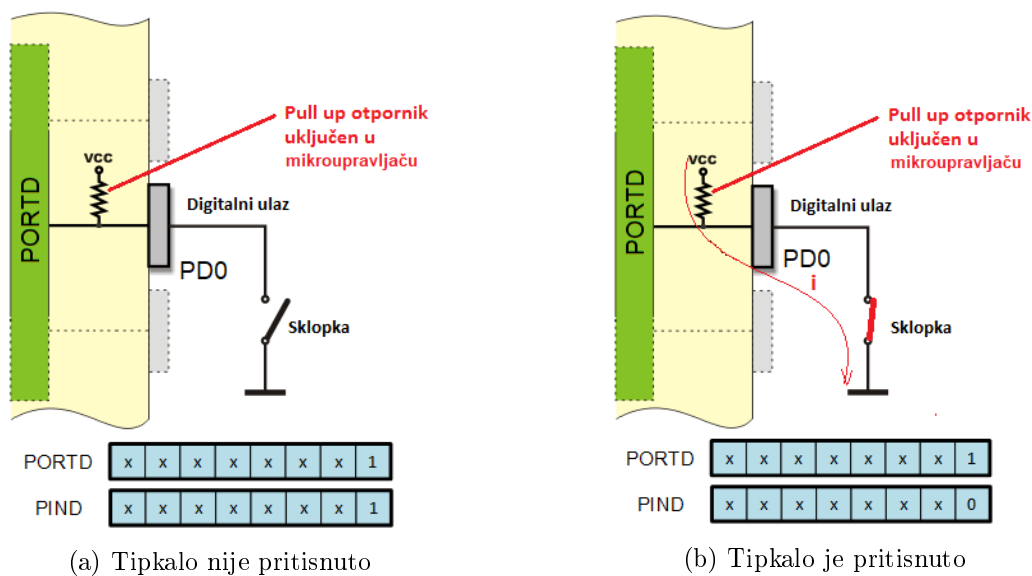
Nisko ili visoko stanje na ulaznom pinu možemo pročitati u registru PINx prema pravilima:

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

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

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

Koja je svrha uključivanja priteznog otpornika prikazat ćemo pomoću slike 4.4<sup>3</sup>.



Slika 4.4: Tipkalo spojeno na ulazni pin PDI0 mikroupravljača ATmega32U4

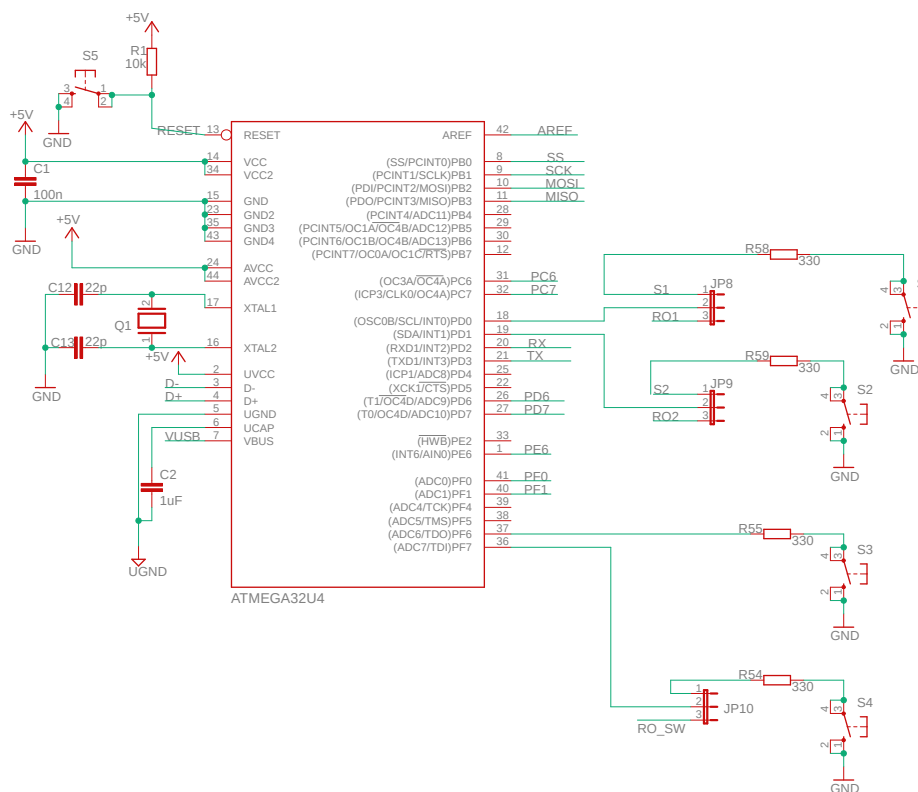
Pretpostavimo da pritezni otpornik nije uključen. Koliki je potencijal na digitalnom ulazu PDI0 ako tipkalo nije pritisnuto (slika 4.4a)? To ne znamo jer je digitalni ulaz u stanju visoke impedancije te u registru PIND nećemo dobiti ispravno očitavanje. Bilo kakav manji poremećaj može promijeniti stanje bita 0 u registru PIND. Iz tog se razloga pin „pritegne” na potencijal 5

<sup>3</sup>Izvor: <https://www.mikroe.com/ebooks/pic-microcontrollers-programming-in-assembly/io-ports>

V. Kada tipkalo nije pritisnuto (slika 4.4a), stanje pina je visoko, odnosno 5 V, a stanje bita 0 u registru **PIND** bit će 1. Kada je tipkalo pritisnuto (slika 4.4b), stanje pina je nisko, odnosno 0 V, a stanje bita 0 u registru **PIND** bit će 0. Struja će u slučaju pritisnutog tipkala teći iz mikroupravljača prema masi (slika 4.4b).

Pritezni otpornici uključuju se ako se na mikroupravljač spajaju tipkala i senzori s otvorenim kolektorom. Postoje i senzori koji aktivno na svom izlazu mogu dati i visoko i nisko stanje<sup>4</sup>. Ako na mikroupravljač spajamo takav senzor, pritezni otpornik nije potrebno uključivati<sup>5</sup>. Pritezni otpornik može se na mikroupravljač spojiti izvana, a njegova je vrijednost najčešće 10 kΩ.

#### 4.2.1 Vježbe - digitalni ulazi mikroupravljača ATmega32U4



Slika 4.5: Shema spajanja tipkala na mikroupravljač ATmega32U4

Digitalne ulaze mikroupravljača ATmega32U4 testirat ćemo pomoću četiriju tipkala spojenih na pinove PD0, PD1, PF6 i PF7. Shema spajanja tipkala na digitalne ulaze mikroupravljača ATmega32U4 prikazana je na slici 4.5. LED diode koje ćemo uz tipkala također koristiti u ovoj vježbi prikazane su na slici 4.1.

Za detalje o konfiguraciji digitalnih ulaza pogledajte tablicu 10-1 u literaturi [1] na stranici 69. Programsko razvojno okruženje *Atmel Studio 7* ima definirana imena registara DDRx, PORTx i PINx (x = B, C, D, E, F). Konfiguracija pinova svodi se na dodjeljivanje vrijednosti u definirana imena registara DDRx i PORTx. Stanja ulaznih pinova čitaju se u registru PINx. Na primjer, ako želimo da svi pinovi porta D budu ulazni pinovi u programskom razvojnom okruženju *Atmel Studio 7*, napisat ćemo **DDRD = 0x00**; u heksadecimalnom zapisu

<sup>4</sup>Senzori s tzv. *push-pull* izlazom.

<sup>5</sup>Ništa se neće dogoditi ako uključite pritezni otpornik.

ili `DDRD = 0b00000000`; u binarnom zapisu. Ukoliko na portu D želimo uključiti pritezne otpornike na svim pinovima u programskom razvojnom okruženju *Atmel Studio 7*, napisat ćemo `PORTD = 0xFF`; ili `PORTD = 0b11111111`; . Pretpostavimo da su na ulaznim pinovima PD0, PD1, PD2 i PD3 visoka stanja, a na pinovima PD4, PD5, PD6 i PD7 niska stanja. Tada bi u programskom razvojnom okruženju *Atmel Studio 7* vrijednost registra `PIND` bila `0x0F = 0b00001111`.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku `Digital Input.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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se *Ivica Ivic*. Datoteku `Digital Input.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atsln` u datoteci `\\Digital Input\vjezbe`. U otvorenom projektu nalaze se sve vježbe koje ćemo obraditi u poglavlju *Digitalni ulazi mikroupravljača ATmega32U4*. 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 4.2.1

Napravite program koji će na razvojnom okruženju s mikroupravljačem `ATmega32U4` uključiti sve LED diode ako je pritisnuto tipkalo spojeno na pin `PD0`. Sheme spajanja tipkala i LED dioda na mikroupravljač `ATmega32U4` prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba421.cpp`. Omogućite samo prevođenje datoteke `vjezba421.cpp`. Početni sadržaj datoteke `vjezba421.cpp` prikazan je programskim kodom 4.16.

Programski kod 4.16: Početni sadržaj datoteke `vjezba421.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {

    output_port(DDRB,PB7); // PB7 postavljen kao izlazni pin
    output_port(DDRB,PB6); // PB6 postavljen kao izlazni pin
    output_port(DDRB,PB5); // PB5 postavljen kao izlazni pin
    output_port(DDRB,PB4); // PB4 postavljen kao izlazni pin

    input_port(DDRD,PD0); // PD0 postavljen kao ulazni pin
    set_port(PORTD,PD0,1); // uključenje priteznog otpornika na PD0
}

int main(void){

    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) {
        if((PIND & 0x01) == 0x00) { // ako je pin PD0 u logičkoj nuli
            PORTB |= 0xF0; // uključi sve led diode
        }
        else {
            PORTB &= ~0xF0; // inače ih isključi
        }
    }
    return 0;
}
```



```

}

```

U tijelu funkcije `inicijalizacija()` nalazi se makronaredba `input_port(DDRx, pin)` koju do sada nismo koristili. Ova makronaredba služi za konfiguraciju ulaznog pina. Kao argumente prima registar `DDRx` i poziciju pina koji želimo postaviti kao ulazni pin.

U bloku naredbi `while` petlje nalazi se `if` uvjetni blok koji uključuje ili isključuje LED diode u ovisnosti o stanju bita 0 registra `PIN`. U tablici 4.3 prikazan je slučaj u kojem je pritisnuto tipkalo spojeno na pin `PD0`. Kada je tipkalo pritisnuto, na pinu `PD0` nisko je stanje, a vrijednost bita 0 u registru `PIND` jest 0. Maska `0x01` koja se koristi za ispitivanje stanja bita 0 formira se tako da se na poziciju bita 0 postavi 1, a na poziciju ostalih bitova postavi se 0 (tablica 4.3). Stanje pina `PD0` u programskom razvojnom okruženju *Atmel Studio 7* ispituje se naredbom `if((PIND & 0x01)== 0x00)`. Rezultat bitovne operacije `PIND & 0x01` bit će jednak `0x00` ako je tipkalo pritisnuto (tablica 4.3). Ako tipkalo nije pritisnuto, rezultat bitovne operacije `PIND & 0x01` bit će jednak `0x01` (tablica 4.4).

Tablica 4.3: Ispitivanje stanja pina `PD0` - slučaj u kojem je pritisnuto tipkalo

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
<b>PIND</b>	x	x	x	x	x	x	x	0
<b>0x01</b>	0	0	0	0	0	0	0	1
<b>PIND &amp; 0x01</b>	0	0	0	0	0	0	0	0

Tablica 4.4: Ispitivanje stanja pina `PD0` - slučaj u kojem nije pritisnuto tipkalo

Pozicija bita	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
<b>PIND</b>	x	x	x	x	x	x	x	1
<b>0x01</b>	0	0	0	0	0	0	0	1
<b>PIND &amp; 0x01</b>	0	0	0	0	0	0	0	1

Općenito, stanje ulaznog pina na poziciji  $i$  na portu  $D$  ispituje se naredbom `if((PIND & maska)== 0x00)`. Varijabla `maska` formira se tako da se na poziciju bita  $i$  postavi 1, a na poziciju ostalih bitova 0. Isti je princip i za port  $B$ ,  $C$ ,  $E$  ili  $F$ , gdje je registar `PIND` potrebno zamijeniti registrom `PINB`, `PINC`, `PINE` ili `PINF`.

Prevedite datoteku `vjezba421.cpp` u strojni kod i snimite ga na mikroupravljač `ATmega32U4`. Testirajte program na razvojnom okruženju s mikroupravljačem `ATmega32U4`.

Mikroupravljač se može inicijalizirati i funkcijom `inicijalizacija()` prikazanom programskim kodom 4.17.

Programski kod 4.17: Funkcija inicijalizacije mikroupravljača - drugi način

```

void inicijalizacija() {
    DDRB |= 0xF0;
    DDRD &= 0xFE;
    PORTD |= 0x01;
}

```

Ovaj način inicijalizacije zahtijeva kreiranje maski kojima će se uključiti, odnosno isključiti bitovi u pojedinim registrima. Na primjer, naredbom `DDRB |= 0xF0`; gornja četiri bita u registru `DDRD` bit će postavljena u 1 (izlazni pinovi), a donja četiri bita istog registra neće biti

promijenjena. Promijenite funkciju `inicijalizacija()` u programskom razvojnom okruženju *Atmel Studio 7* sukladno programskom kodu 4.17. Prevedite datoteku `vjezba421.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Treći način inicijalizacije mikroupravljača jest pomoću bitovnih operatora (programski kod 4.18). Ovo je najsloženiji način inicijalizacije, ali daje jasnu sliku o konfiguraciji pinova. Promijenite funkciju `inicijalizacija()` u programskom razvojnom okruženju *Atmel Studio 7* sukladno programskom kodu 4.18. Prevedite datoteku `vjezba421.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Programski kod 4.18: Funkcija inicijalizacije mikroupravljača - treći način

```
void inicijalizacija() {
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    DDRD &= ~(1 << PD0);

    PORTD |= (1 << PD0);
}
```

Izbrišite naredbu `PORTD |= (1 << PD0);` u funkciji `inicijalizacija()`. Prevedite datoteku `vjezba421.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Pokušajte sada jednim prstom dotaknuti nožice tipkala koje je spojeno na pin PD0, a drugim prstom dotaknuti trn konektora za serijsku komunikaciju na kojem piše 5 V. Primijetite da se LED diode neprestano uključuju i isključuju iako niste pritisnuli tipkalo spojeno na pin PD0. Onog trenutka kada smo u funkciji `inicijalizacija()` izbrisali naredbu `PORTD |= (1 << PD0);`, isključili smo pritezni otpornik na pinu PD0. Kada tipkalo nije pritisnuto, pin PD0 nalazi se u stanju visoke impedancije te svaki vanjski poremećaj utječe na promjenu stanja bita 0 u registru `PIND`.

Zatvorite datoteku `vjezba421.cpp` i onemogućite prevođenje ove datoteke.



## Vježba 4.2.2

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i plava LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba422.cpp`. Omogućite samo prevođenje datoteke `vjezba422.cpp`. Početni sadržaj datoteke `vjezba422.cpp` prikazan je programskim kodom 4.19.

Programski kod 4.19: Početni sadržaj datoteke `vjezba422.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {
    // PB7, PB6, PB5, i PB4 postavljeni kao izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    // PD1 i PD0 postavljeni kao ulazni pinovi
```

```

    DDRD  &= ~( (1 << PD1) | (1 << PD0));
    // PF7 i PF6 postavljeni kao ulazni pinovi
    DDRF  &= ~( (1 << PF7) | (1 << PF6));
    // pritezni otpornici uključeni na pinovima PD1 i PD0
    PORTD |= (1 << PD1) | (1 << PD0);
    // pritezni otpornici uključeni na pinovima PF7 i PF6
    PORTF |= (1 << PF7) | (1 << PF6);
}

int main(void) {
    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) {
        if((PIND & (1 << PD0)) == 0x00) { // ako je pin PD0 u logičkoj nuli
            PORTB |= (1 << PB4); // uključiti crvenu LED diodu
        }
        else {
            PORTB &= ~(1 << PB4); // inače je isključi
        }
        // nastavite za ostala tipkala
    }
    return 0;
}

```

U vježbi je potrebno uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PD0, žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PD1, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PF6 i plavu LED diodu ako je pritisnuto tipkalo spojeno na pin PF7.

U programskom kodu 4.19 prikazano je tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PD0, PD1, PF6 i PF7 konfigurirani kao ulazni. Uključeni su pritezni otpornici za pinove PD0, PD1, PF6 i PF7. Pri pokretanju strojnog koda u mikroupravljaču svi DDRx registri inicijalno su u stanju 0x00, pa smo stoga mogli izostaviti dio tijela funkcije `inicijalizacija()` koji konfigurira ulazne pinove. Preporučuje se da unatoč tome ne izostavljate konfiguraciju ulaznih pinova zbog preglednosti programskog koda i jasnog uvida u namjenu digitalnog pina.

U `while` petlji prikazanoj u programskom kodu 4.19 nalazi se niz naredbi koje će uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PD0. Proširite niz naredbi `while` petlje tako da uključite žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PD1, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PF6 i plavu LED diodu ako je pritisnuto tipkalo spojeno na pin PF7. Maska za naredbu `if((PIND & maska)== 0x00)` formira se na sljedeći način:

- `maska = (1 << PD0)` - ako želite provjeriti stanje pina PD0 (`(1 << PD0) = 0x01` na poziciji bita 0 ima vrijednost 1, ostali bitovi imaju vrijednost 0),
- `maska = (1 << PD1)` - ako želite provjeriti stanje pina PD1 (`(1 << PD1) = 0x02` na poziciji bita 1 ima vrijednost 1, ostali bitovi imaju vrijednost 0).

Maska za naredbu `if((PINF & maska)== 0x00)` formira se na sljedeći način:

- `maska = (1 << PF6)` - ako želite provjeriti stanje pina PF6 (`(1 << PF6) = 0x40` na poziciji bita 6 ima vrijednost 1, ostali bitovi imaju vrijednost 0),
- `maska = (1 << PF7)` - ako želite provjeriti stanje pina PF7 (`(1 << PF7) = 0x80` na poziciji bita 7 ima vrijednost 1, ostali bitovi imaju vrijednost 0).

Prevedite datoteku `vjezba422.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba422.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.2.3

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i plava LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Za provjeru stanja ulaznog pina koristite makronaredbu `get_pin`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba423.cpp`. Omogućite samo prevođenje datoteke `vjezba423.cpp`. Početni sadržaj datoteke `vjezba423.cpp` prikazan je programskim kodom 4.20.

Programski kod 4.20: Početni sadržaj datoteke `vjezba423.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {

    output_port(DDRB, PB7); // PB7 postavljen kao izlazni pin
    output_port(DDRB, PB6); // PB6 postavljen kao izlazni pin
    output_port(DDRB, PB5); // PB5 postavljen kao izlazni pin
    output_port(DDRB, PB4); // PB4 postavljen kao izlazni pin

    input_port(DDRD, PD0); // PD0 postavljen kao ulazni pin
    input_port(DDRD, PD1); // PD1 postavljen kao ulazni pin
    input_port(DDRF, PF6); // PF6 postavljen kao ulazni pin
    input_port(DDRF, PF7); // PF7 postavljen kao ulazni pin

    set_port(PORTD, PD0, 1); // uključivanje priteznog otpornika na PD0
    set_port(PORTD, PD1, 1); // uključivanje priteznog otpornika na PD1
    set_port(PORTF, PF6, 1); // uključivanje priteznog otpornika na PF6
    set_port(PORTF, PF7, 1); // uključivanje priteznog otpornika na PF7
}

int main(void) {

    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) {

        if(get_pin(PIND, PD0) == 0) { // ako je pin PD0 u logičkoj nuli
            set_port(PORTB, PB4, 1); // uključi crvenu LED diodu
        }
        else {
            set_port(PORTB, PB4, 0); // inače je isključi
        }
        // nastavite za ostala tipkala
    }
    return 0;
}
```

U ovoj vježbi cilj je isti kao i u prethodnoj vježbi. U programskom kodu 4.20 prikazano je

tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PD0, PD1, PF6 i PF7 konfigurirani kao ulazni. Uključeni su pritezni otpornici za pinove PD0, PD1, PF6 i PF7. Za navedene konfiguracije korištene su makronaredbe `output_port`, `input_port` i `set_port` iz zaglavlja `avrvub.h`.

U `while` petlji prikazanoj u programskom kodu 4.20 nalazi se niz naredbi koje će uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PD0. Za provjeru stanja ulaznog pina korištena je makronaredba `get_pin`. Ova makronaredba kao argumente prima registar PINx i poziciju pina za koji želite ispitati stanje, a kao rezultat vraća stanje ulaznog pina. Na primjer, ako je na ulaznom pinu PD0 visoko stanje, makronaredba `get_pin(PIND,PD0)` vratit će vrijednost 1. Ako je na ulaznom pinu PD0 nisko stanje, makronaredba `get_pin(PIND,PD0)` vratit će vrijednost 0. Kada je tipkalo pritisnuto, stanje je pina nisko. Prema tome, logički uvjet `(get_pin(PIND,PD0)== 0)` bit će istinit ako je tipkalo pritisnuto, a lažan ako tipkalo nije pritisnuto.

Proširite niz naredbi `while` petlje tako da uključite žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PD1, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PF6 i plavu LED diodu ako je pritisnuto tipkalo spojeno na pin PF7 korištenjem makronaredbe `get_pin`.

Prevedite datoteku `vjezba423.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba423.cpp` i onemogućite prevođenje ove datoteke.



#### Vježba 4.2.4

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i plava LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Za provjeru stanja ulaznog pina koristite funkciju `filtered_pin_state`. Funkcija `filtered_pin_state` može filtrirati smetnje koje se javljaju zbog istitravanja tipkala. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba424.cpp`. Omogućite samo prevođenje datoteke `vjezba424.cpp`. Početni sadržaj datoteke `vjezba424.cpp` prikazan je programskim kodom 4.21.

Programski kod 4.21: Početni sadržaj datoteke `vjezba424.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {

    // PB7,PB6,PB5,i PB4 postavljeni kao izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    // PD1 i PD0 postavljeni kao ulazni pinovi
    DDRD &= ~((1 << PD1) | (1 << PD0));
    // PF7 i PF6 postavljeni kao ulazni pinovi
    DDRF &= ~((1 << PF7) | (1 << PF6));
    // pritezni otpornici uključeni na pinovima PD1 i PD0
    PORTD |= (1 << PD1) | (1 << PD0);
    // pritezni otpornici uključeni na pinovima PF7 i PF6
    PORTF |= (1 << PF7) | (1 << PF6);
}
```

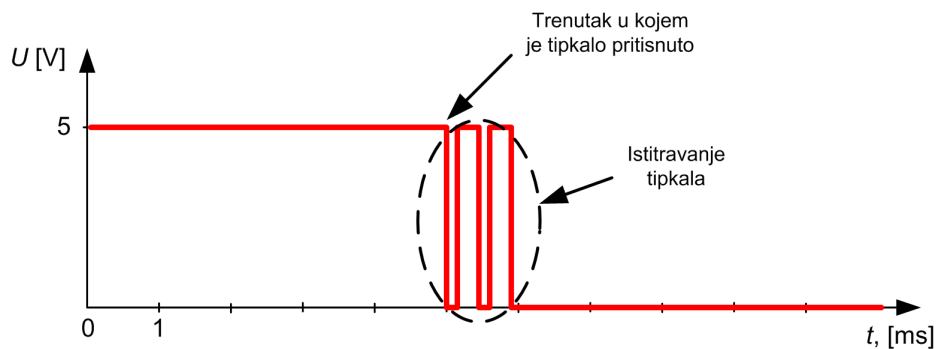
```

int main(void) {
    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) {
        // ako je filtrirani pin PD0 u logičkoj nuli
        if(filtered_pin_state(PD,PD0,false,30) == false) {
            set_port(PORTB, PB4, 1); // uključi crvenu LED diodu
        }
        else {
            set_port(PORTB, PB4, 0); // inače je isključi
        }
        // nastavite za ostala tipkala
    }
    return 0;
}

```

U trenutku kada pritisnemo istrošeno tipkalo na razvojnom okruženju s mikroupravljačem ATmega32U4, brid signala tipkala neće odmah pasti na nisku razinu, već dolazi do istitravanja (slika 4.6).



Slika 4.6: Istitravanje koje se javlja kod istrošenog tipkala

Istitravanje može biti nezgodno ako npr. moramo brojati koliko je puta tipkalo pritisnuto. Pogledajmo signal tipkala na slici 4.6. Ukoliko bi za provjeru stanja tipkala koristili makronaredbu `get_pin`, brojač koji broji koliko je puta pritisnuto tipkalo uvećao bi se za tri. Razlog tome jest uzastopni prijelaz iz visokog u nisko stanje zbog mehaničkih karakteristika tipkala.

Ovaj problem može se riješiti uvođenjem vremenskog okvira unutar kojeg ćemo pratiti koliko je vremena tipkalo bilo pritisnuto. U zaglavlju `avr/vub.h` nalazi se funkcija `filtered_pin_state` koja rješava problem istitravanja tipkala.

Funkcija `filtered_pin_state` prima četiri argumenta:

- oznaku porta ulaznog pina čije stanje želite ispitati (**PB**, **PC**, **PD**, **PE**, **PF**),
- poziciju ulaznog pina čije stanje želite ispitati (npr. **PD1**),
- stanje koje želite ispitati na poziciji ulaznog pina (npr. ako želite ispitati je li na poziciji pina nisko stanje, ovdje ćete upisati 0),
- vremenska konstanta u milisekundama predstavlja vremenski okvir unutar kojeg će se svake milisekunde ispitivati koje je trenutno stanje ulaznog pina.

Pretpostavimo da smo koristili funkciju `filtered_pin_state(PF,PF6,0,30)`. Ova će funkcija vratiti vrijednost 0 ako je ulazni pin PF6 na portu F (**PF**) 90 % vremena definiranog vremenskom

konstantom (četvrti argument 30 ms) bio u niskom stanju. Na ovaj način kratkotrajni će poremećaji biti filtrirani zbog činjenice da ulazni pin PF6 90 % vremena definiranog vremenskom konstantom biti neće u niskom stanju. Vremenska konstanta podešava se iskustveno, a preporučena joj je vrijednost 25 ms.

Definicija funkcije `filtered_pin_state` nalazi se u datoteci `avrvub.cpp`. Znatizeljni čitatelj detaljno će proučiti način rada ove funkcije.

U programskom kodu 4.21 prikazano je tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PD0, PD1, PF6 i PF7 konfigurirani kao ulazni. Uključeni su pritezni otpornici za pinove PD0, PD1, PF6 i PF7.

U `while` petlji prikazanoj u programskom kodu 4.21 nalazi se niz naredbi koje će uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PD0. Za provjeru stanja ulaznog pina korištena je funkcija `filtered_pin_state`. Proširite niz naredbi `while` petlje tako da uključite žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PD1, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PF6 i plavu LED diodu ako je pritisnuto tipkalo spojeno na pin PF7 korištenjem funkcije `filtered_pin_state`. Vremensku konstantu funkcije `filtered_pin_state` postavite na 25 ms.

Prevedite datoteku `vjezba424.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Sada promijenite vremensku konstantu tako da za pin PD0 ona iznosi 30 ms, za pin PD1 100 ms, za pin PF6 500 ms, a za pin PF7 1000 ms. Ponovno prevedite datoteku `vjezba424.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Komentirajte rezultat!

Zatvorite datoteku `vjezba424.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 4.2.5

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 promijeniti stanje crvene LED diode na padajući brid signala tipkala spojenog na pin PD0, žute LED diode na rastući brid signala tipkala spojenog na pin PD1, zelene LED diode na padajući brid signala tipkala spojenog na pin PF6 i plave LED diode na rastući brid signala tipkala spojenog na pin PF7. Za detekciju padajućeg brida koristite funkciju `isFalling_edge`, a za detekciju rastućeg brida funkciju `isRising_edge`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba425.cpp`. Omogućite samo prevođenje datoteke `vjezba425.cpp`. Početni sadržaj datoteke `vjezba425.cpp` prikazan je programskim kodom 4.22.

Programski kod 4.22: Početni sadržaj datoteke `vjezba425.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {
    // napravite konfiguraciju
}

int main(void) {

    inicijalizacija(); // inicijalizacija mikroupravljača

    while (1) {
```

```

// ako je padajući brid na pinu PD0
if(isFalling_edge(PD, PD0) == true) {
    toggle_port(PORTB, PB4);    // promijeni stanje crvene LED diode
}

// ako je rastući brid na pinu PD1
if(isRising_edge(PD, PD1) == true) {
    toggle_port(PORTB, PB5);    // promijeni stanje žute LED diode
}
// nastavite za ostala tipkala
}
return 0;
}

```

Često se u praktičnoj primjeni koriste gumbi na dodir (engl. *touch button*). Oni služe za uključenje nekog dijela sustava. Do uključanja sustava mora doći kada dodirnemo gumb i nakon toga sustav mora nastaviti raditi. Dosad nismo imali takav slučaj jer su LED diode bile uključene samo ako su tipkala konstantno bila pritisnuta.

U programskom kodu 4.22 prikazano je tijelo funkcije `inicijalizacija()`. Napravite konfiguraciju svih pinova koji će se koristiti u ovoj vježbi. Način konfiguracije proizvoljan je.

U `while` petlji prikazanoj u programskom kodu 4.22 nalazi se niz naredbi koje će promijeniti stanje crvene LED diode ako se na pinu PD0 pojavi padajući brid signala te promijeniti stanje žute LED diode ako se na pinu PD1 pojavi rastući brid signala. Padajući brid signala na pinu PD0 javit će se u trenutku kada pritisnemo tipkalo. Tada će stanje iz visokog prijeći u nisko. Rastući brid signala na pinu PD1 javit će se u trenutku kada otpustimo tipkalo. Tada će stanje iz niskog prijeći u visoko. Za detekciju padajućeg brida koristimo funkciju `isFalling_edge`, a za detekciju rastućeg brida funkciju `isRising_edge`. Obje funkcije vraćaju `bool` vrijednost na sljedeći način:

- funkcija `isFalling_edge` vratit će `true` ako se na pinu pojavio padajući brid, a `false` ako se brid nije pojavio,
- funkcija `isRising_edge` vratiti će `true` ako se na pinu pojavio rastući brid, a `false` ako se brid nije pojavio.

Funkcije `isFalling_edge` i `isRising_edge` primaju dva argumenta:

- oznaku porta ulaznog pina čiji brid signala želite ispitati (`PB`, `PC`, `PD`, `PE`, `PF`),
- poziciju ulaznog pina čiji brid signala želite ispitati (npr. `PD1`).

Definicije funkcija `isFalling_edge` i `isRising_edge` nalaze se u datoteci `avrvub.cpp`. Znatiželjni čitatelj detaljno će proučiti način rada ovih funkcija, kao i deklariranu varijablu (polje podataka) u kojoj se čuvaju stare vrijednosti ulaznog signala kako bi se mogao detektirati brid signala.

Proširite niz naredbi `while` petlje tako da se stanje zelene LED diode promijeni na padajući brid signala tipkala spojenog na pin PF6, a stanje plave LED diode na rastući brid signala tipkala spojenog na pin PF7.

Prevedite datoteku `vjezba425.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba425.cpp` i onemogućite prevođenje ove datoteke.





### Vježba 4.2.6

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, uključiti žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PD1 (filtrirati istitravanje tipkala), promijeniti stanje zelene LED diode na padajući brid signala tipkala spojenog na pin PF6 te promijeniti stanje plave LED diode na rastući brid signala tipkala spojenog na pin PF7. Implementaciju rješenja potrebno je izvesti pomoću biblioteke `DigitalIO.h` i klasa `DigitalOutput` i `DigitalInput`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

U projektnom stablu otvorite datoteku `vjezba426.cpp`. Omogućite samo prevođenje datoteke `vjezba426.cpp`. Početni sadržaj datoteke `vjezba426.cpp` prikazan je programskim kodom 4.23.

Programski kod 4.23: Početni sadržaj datoteke `vjezba426.cpp`

```
#include "AVR_VUB/avrub.h"
#include "DigitalIO/DigitalIO.h"
#include <avr/io.h>
#include <util/delay.h>

int main(void) {

    // instance objekata digitalnih izlaza
    DigitalOutput plava(B7); // stvori objekt plava za diodu na pinu PB7
    DigitalOutput zelena(B6); // stvori objekt zelena za diodu na pinu PB6
    DigitalOutput zuta(B5); // stvori objekt zuta za diodu na pinu PB5
    DigitalOutput crvena(B4); // stvori objekt crvena za diodu na pinu PB4

    // instance objekata digitalnih ulaza
    DigitalInput tipkalo1(D0); // stvori objekt za tipkalo na pinu PD0
    // instancirati ostale digitalne ulaze

    tipkalo1.pullup_on(); //uključiti pull up za pin PD0
    //uključiti ostale pull up otpornike

    while (1) {

        // napraviti funkcionalnost zadanu zadatkom
    }
    return 0;
}
```

Autor je za manipulaciju digitalnim izlazima i ulazima izradio biblioteku `DigitalIO.h` u kojoj se nalaze klase `DigitalOutput` i `DigitalInput`. Biblioteka `DigitalIO.h` u programskom je kodu 4.23 uključena pomoću naredbe `#include "DigitalIO/DigitalIO.h"`. Ovaj pristup konfiguracije i manipulacije digitalnim izlazima i ulazima sličan je implementaciji u *Arduino* razvojnom okruženju s kojim ste se možda već susreli.

U ovom zadatku koriste se klase `DigitalOutput` i `DigitalInput`. Klasa `DigitalOutput` detaljno je opisana u vježbi 4.1.8. U nastavku ćemo opisati klasu `DigitalInput`. U klasi `DigitalInput` definirana su tri preopterećena konstruktora. Dva konstruktora bitna su za stvaranje objekata koji će se koristiti kao digitalni ulazi. Stvaranje objekata koji su digitalni ulazi moguće je na dva načina:

- `DigitalInput tipkalo1(D0)`; - objekt se inicijalizira konstantom  $Xi$  ( $X = B, C, D, E, F$ ;  $i = 0,1,2,3,4,5,6,7$ ) gdje  $X$  označava port, a  $i$  pin na portu. Npr. za pin 0 na portu D

koristit ćemo konstantu `D0`, a za pin 3 na portu B koristit ćemo konstantu `B3`.

- `DigitalInput` `tipkalo1(PD, PD0)`; - objekt se inicijalizira konstantama  $X$  i  $X_i$  ( $X = PB, PC, PD, PE, PF$ ;  $i = 0,1,2,3,4,5,6,7$ ) gdje  $X$  označava port, a  $X_i$  pin na portu. Npr. za pin 6 na portu F koristit ćemo konstante (`PF, PF6`), a za pin 1 na portu D koristit ćemo konstante (`PD, PD1`).

Tipkala su redom spojena na pinove `PD0, PD1, PF6` i `PF7`. Instanciranje objekata za digitalne ulaze na koje su spojena tipkala prikazano je programskim kodom 4.24. Taj kod potrebno je napisati u `main()` funkciji nakon dijela programskog koda kojim su instancirane LED diode.

Programski kod 4.24: Instanciranje objekata za digitalne ulaze na koje su spojena tipkala

```
// instance objekata digitalnih ulaza
DigitalInput tipkalo1(D0); // stvori objekt za tipkalo na pinu PD0
DigitalInput tipkalo2(D1); // stvori objekt za tipkalo na pinu PD1
DigitalInput tipkalo3(F6); // stvori objekt za tipkalo na pinu PF6
DigitalInput tipkalo4(F7); // stvori objekt za tipkalo na pinu PF7
```

Primijetite kako u ovom pristupu nema inicijalizacijske funkcije `inicijalizacija()`. Razlog tome je taj što se objekti trebaju tretirati kao i varijable jer egzistiraju samo u bloku u kojem su definirani. Dakle, objekti `tipkalo1, tipkalo2, tipkalo3, tipkalo4` dohvatljivi su samo u `main()` funkciji.

Klasa `DigitalInput` ima brojne metode (funkcije klase) koje se mogu pozvati nad objektima:

- `pullup_on()` - metoda uključuje pritezni otpornik na pinu,
- `pullup_off()` - metoda isključuje pritezni otpornik na pinu,
- `state()` - metoda vraća `bool` stanje ulaznog digitalnog pina objekta,
- `isRising_edge()` - metoda služi za detekciju rastućeg brida. Vraća vrijednost `true` ako se na pinu pojavio rastući brid, a `false` ako se brid nije pojavio.
- `isFalling_edge()` - metoda služi za detekciju padajućeg brida. Vraća vrijednost `true` ako se na pinu pojavio padajući brid, a `false` ako se brid nije pojavio.
- `filtered_state(bool stanje, uint16_t T)` - metoda koja provjerava je li na pinu `stanje` (prvi argument metode) 90 % vremena `T` definiranog drugim argumentom metode. Ova metoda omogućuje filtriranje istitravanja tipkala.

Metode se nad objektima pozivaju operatorom `.` pa se stanje objekta `tipkalo1` može dohvatiti naredbom `tipkalo1.state()`;

Da bi stanja tipkala bila ispravno detektirana, potrebno je uključiti pritezne otpornike na digitalnim ulazima na koje su spojena tipkala. Uključivanje priteznog otpornika za digitalne ulaze na koje su spojena tipkala prikazano je programskim kodom 4.25. Taj kod potrebno je napisati u `main()` funkciji nakon instanciranja objekata za tipkala.

Programski kod 4.25: Uključivanje priteznog otpornika za digitalne ulaze na koje su spojena tipkala

```
tipkalo1.pullup_on(); //ukljuci pull up za pin PD0
tipkalo2.pullup_on(); //ukljuci pull up za pin PD1
tipkalo3.pullup_on(); //ukljuci pull up za pin PF6
tipkalo4.pullup_on(); //ukljuci pull up za pin PF7
```

Funkcionalnost zadana zadatkom definirana je u beskonačnoj petlji programskim kodom 4.26. Objasnimo sada pozive metoda nad objektima ulaznih pinova:

- `if (tipkalo1.state()== false)` - provjeravanje je li pritisnuto tipkalo na pinu PD0. Ako je pritisnuto, stanje tipkala će biti `false`.
- `if (tipkalo2.filtered_state(false, 25)== false)` - provjeravanje je li pritisnuto tipkalo na pinu PD1 90 % vremena od 25 ms.
- `if (tipkalo3.isFalling_edge())` - provjeravanje je li se pojavio padajući brid na tipkalu spojenom na pin PF6.
- `if (tipkalo4.isRising_edge())` - provjeravanje je li se pojavio rastući brid na tipkalu spojenom na pin PF7.

U `while(1)` petlju prepisite programski kod 4.26.

Programski kod 4.26: Funkcionalnost zadana zadatkom u beskonačnoj petlji

```
if (tipkalo1.state() == false) {
    crvena.on();
}
else {
    crvena.off();
}

if (tipkalo2.filtered_state(false, 25) == false) {
    zuta.on();
}
else {
    zuta.off();
}

if (tipkalo3.isFalling_edge()){
    zelena.toggle();
}

if (tipkalo4.isRising_edge()){
    plava.toggle();
}
```

Prevedite datoteku `vjezba426.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba426.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

## 4.2.2 Zadaci - digitalni ulazi mikroupravljača ATmega32U4

### Zadatak 4.2.1

Napravite program koji će na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti crvenu i zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PD1. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

### Zadatak 4.2.2

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti plava LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

### Zadatak 4.2.3

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti plava LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Za provjeru stanja ulaznog pina koristite makronaredbu `get_pin`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

### Zadatak 4.2.4

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti plava LED dioda ako je pritisnuto tipkalo spojeno na pin PD0, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PD1, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PF6 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PF7. Za provjeru stanja ulaznog pina koristite funkciju `filtered_pin_state`. Funkcija `filtered_pin_state` može filtrirati smetnje koje se javljaju zbog istitravanja tipkala. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

### Zadatak 4.2.5

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 promijeniti stanje plave LED diode na rastući brid signala tipkala spojenog na pin PD0, zelene LED diode na padajući brid signala tipkala spojenog na pin PD1, žute LED diode na rastući brid signala tipkala spojenog na pin PF6 i crvene LED diode na padajući brid signala tipkala spojenog na pin PF7. Za detekciju padajućeg brida koristite funkciju `isFalling_edge`, a za detekciju rastućeg brida funkciju `isRising_edge`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

### Zadatak 4.2.6

Napravite program kojim će se na razvojnom okruženju s mikroupravljačem ATmega32U4 uključiti plava LED dioda ako nije pritisnuto tipkalo spojeno na pin PD0, uključiti zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PD1 (filtrirati istitravanje tipkala), promijeniti stanje žute LED diode na rastući brid signala tipkala spojenog na pin PF6 te promijeniti stanje crvene LED diode na padajući brid signala tipkala spojenog na pin PF7. Implementaciju rješenja potrebno je izvesti pomoću biblioteke `DigitalIO.h` i klasa `DigitalOutput` i `DigitalInput`. Sheme spajanja tipkala i LED dioda na mikroupravljač ATmega32U4 prikazane su na slikama 4.1 i 4.5.

---

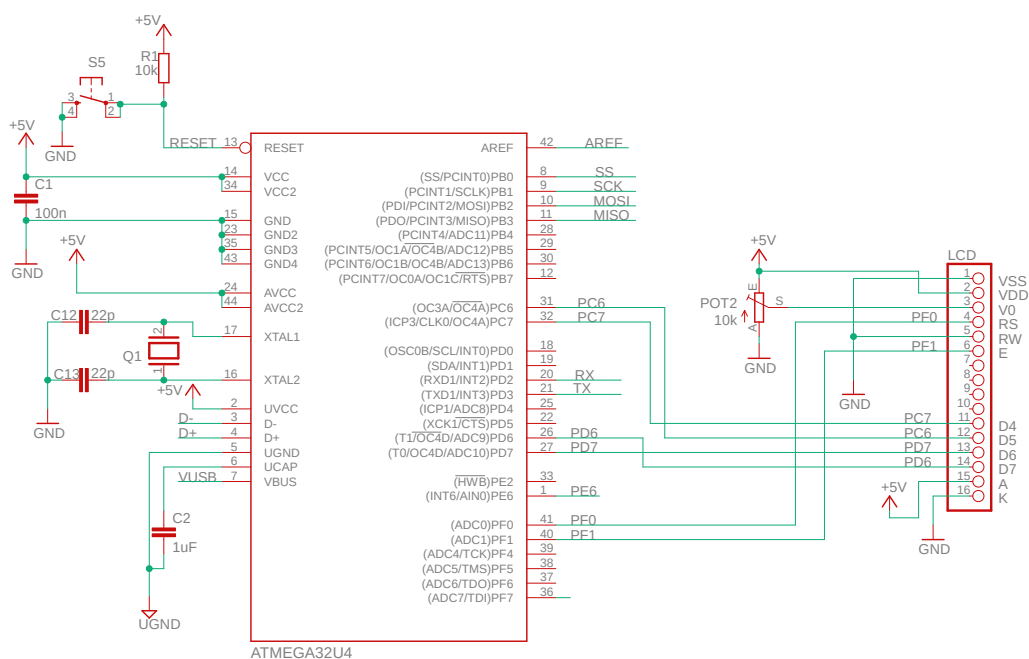


## Poglavlje 5

# LCD displej

LCD displej koristi se za prikazivanje varijabli sustava koje mogu biti temperatura, vlaga zraka, brzina vrtnje, broj proizvedenih proizvoda i drugo. Često se koristi za izbornik kojim se može napraviti pregled i izmjena parametara sustava. Na razvojnom okruženju s mikroupravljačem ATmega32U4 prikazanom na slici 3.1 nalazi se konektor za LCD displej GDM1602E. On ima mogućnost prikaza dvaju redaka sa 16 znakova u jednom retku. U pojedini redak moguće je zapisati 40 znakova, no samo ih je 16 vidljivo. LCD displej posjeduje funkcije za pomicanje teksta ulijevo i udesno te tako možemo prikazati ukupno 40 znakova u jednom retku. On ima vlastiti mikroupravljač koji se brine o dekodiranju podataka koji pristižu na njegove ulazne pinove. Znakovi koje LCD displej prikazuje nalaze se u njegovu memorijskom prostoru. Za prikaz na LCD displeju dostupni su svi znakovi engleske abecede, dok se slova kao što su č, ć, đ, š i ž mogu definirati i pohraniti u njegovu memoriju. Detalje o LCD displeju GDM1602E možete pronaći u literaturi [2].

### 5.1 Vježbe - LCD displej



Slika 5.1: Shema spajanja LCD displeja na mikroupravljač ATmega32U4

Na slici 5.1 prikazana je shema spajanja LCD displeja na mikroupravljač ATmega32U4. Nova komponenta u ovoj vježbi bit će LCD displej koji ima mogućnost prijenosa podataka pomoću 4-bitne i 8-bitne podatkovne sabirnice. Mi ćemo koristiti 4-bitnu podatkovnu sabirnicu radi uštede digitalnih pinova potrebnih za komunikaciju s LCD displejom. U 4-bitnom načinu rada mikroupravljač je potrebno spojiti na pinove LCD displeja D4, D5, D6 i D7. Bilo koja četiri pina mikroupravljača možemo spojiti na pinove D4, D5, D6 i D7. Na razvojnom okruženju s mikroupravljačem ATmega32U4 4-bitna podatkovna sabirnica LCD displeja spojena je na pinove PC7, PC6, PD7 i PD6. Pinovi RS i E na LCD displeju upravljački su pinovi te su spojeni na pinove PF0 i PF1 (slika 5.1). Na LCD displej moguće je zapisivati podatke i čitati podatke. Na razvojnom okruženju s mikroupravljačem ATmega32U4 omogućeno je samo zapisivanje podataka na LCD displej, što je sasvim dovoljno za svaku primjenu. Pin R/W spojen je na masu kako bi na LCD displeju bilo omogućeno samo zapisivanje podataka. 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. U većini slučajeva odluka o spajanju LCD displeja na digitalne pinove mikroupravljača dolazi na kraju, kada se na mikroupravljač spoje senzori i aktuatori koji koriste namjenske pinove mikroupravljača. Potencijometar POT2 na slici 5.1 spojen je na pin V0 na LCD displeju, a služi za promjenu kontrasta LCD displeja.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku `LCD display.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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se *Ivica Ivic*. Datoteku `LCD display.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atsln` u datoteci `\\LCD display\vjezbe`. U otvorenom projektu nalaze se sve 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.

Neposredno prije nego što počnemo ispisivati tekst na LCD displej pomoću mikroupravljača ATmega32U4, potrebno je konfigurirati pinove mikroupravljača koji se koriste za komunikaciju s LCD displejom. U otvorenom projektu nalazi se mapa LCD u kojoj se nalazi zaglavlje `lcd.h`. Otvorite zaglavlje `lcd.h`.

Programski kod 5.1: Konfiguracija pinova LCD displeja u zaglavlju `lcd.h`

```
//-----
// korisnik mijenja samo konfiguraciju displeja pri korištenju ove biblioteke
// konfiguracija LCD displeja
#define LCD_D4_PORT    PORTC // port za 4 bitnu komunikaciju - D4 na LCD-u
#define LCD_D5_PORT    PORTC // port za 4 bitnu komunikaciju - D5 na LCD-u
#define LCD_D6_PORT    PORTD // port za 4 bitnu komunikaciju - D6 na LCD-u
#define LCD_D7_PORT    PORTD // port za 4 bitnu komunikaciju - D7 na LCD-u
#define LCD_D4_PIN     PC7   // pin za 4 bitnu komunikaciju - D4 na LCD-u
#define LCD_D5_PIN     PC6   // pin za 4 bitnu komunikaciju - D5 na LCD-u
#define LCD_D6_PIN     PD7   // pin za 4 bitnu komunikaciju - D6 na LCD-u
#define LCD_D7_PIN     PD6   // pin za 4 bitnu komunikaciju - D7 na LCD-u

#define LCD_RS_PORT    PORTF // port za odabir registra - RS na LCD-u
#define LCD_RS_PIN     PF0   // pin za odabir registra - RS na LCD-u
#define LCD_E_PORT     PORTF // port za odobrenje upisa - EN naLCD-u
#define LCD_E_PIN      PF1   // pin za odobrenje upisa - EN na LCD-u

#define LCD_LINES      2     // broj vidljivih linija na LCD-u
#define LCD_DISP_LENGTH 16   // broj vidljivih znakova po liniji LCD-u
// kraj konfiguracije LCD displeja
//-----
```



U programskom kodu 5.1 prikazana je konfiguracija pinova LCD displeja u zaglavlju `lcd.h`. Konfiguracija pinova mikroupravljača neizostavna je ako želite da LCD displej radi ispravno. U programskom kodu 5.1 nalaze se definicije koje se koriste u datoteci `lcd.c` u kojoj se nalaze definicije svih funkcija potrebnih za rad s LCD displejom. Komunikacija između mikroupravljača i LCD displeja 4-bitna je. Podatkovna sabirnica konfigurira se na sljedeći način:

- `#define LCD_D4_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja D4,
- `#define LCD_D5_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja D5,
- `#define LCD_D6_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja D6,
- `#define LCD_D7_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja D7,
- `#define LCD_D4_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja D4,
- `#define LCD_D5_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja D5,
- `#define LCD_D6_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja D6,
- `#define LCD_D7_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja D7.

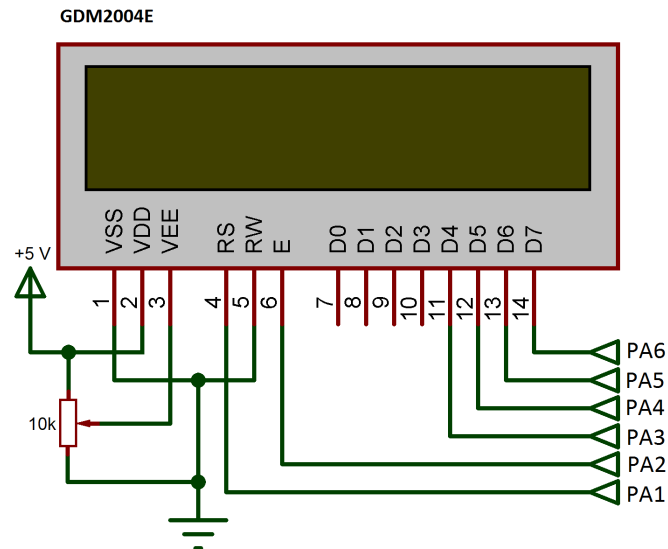
Upravljački pinovi konfiguriraju se na sljedeći način:

- `#define LCD_RS_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja RS,
- `#define LCD_RS_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja RS,
- `#define LCD_E_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikroupravljača na kojem je spojen pin LCD displeja E,
- `#define LCD_E_PIN Pxi`, ( $x = A, B, C, D$ ;  $i = 0,1,\dots,7$ ) - pozicija pina mikroupravljača koji je spojen na pin LCD displeja E.

Broj redaka LCD displeja i broj znakova u jednom retku konfiguriraju se na sljedeći način:

- `#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  $y = 10, 16, 20$ ).

Na slici 5.2 prikazan je primjer spajanja LCD displeja GDM2004E na mikroupravljač ATmega32U4.



Slika 5.2: Primjer LCD displeja GDM2004E spojenog na proizvoljan mikroupravljač s portom A

Ovaj LCD displej ima 4 vidljiva retka i 20 vidljivih znakova u jednom retku. Svi pinovi LCD displeja (podatkovni i upravljački) spojeni su na port A mikroupravljača prema slici 5.2. Za ovako spojen LCD displej na mikroupravljač potrebno je napraviti konfiguraciju pinova u zaglavlju `lcd.h`. Konfiguracija pinova mikroupravljača spojenih na LCD displej prikazana je u programskom kodu 5.2.

Programski kod 5.2: Konfiguracija pinova LCD displeja prikazanog na slici 5.2 u zaglavlju `lcd.h`

```
//-----
// korisnik mijenja samo konfiguraciju displeja pri korištenju ove biblioteke
// konfiguracija LCD displeja
#define LCD_D4_PORT    PORTA // port za 4 bitnu komunikaciju - D4 na LCD-u
#define LCD_D5_PORT    PORTA // port za 4 bitnu komunikaciju - D5 na LCD-u
#define LCD_D6_PORT    PORTA // port za 4 bitnu komunikaciju - D6 na LCD-u
#define LCD_D7_PORT    PORTA // port za 4 bitnu komunikaciju - D7 na LCD-u
#define LCD_D4_PIN     PA3    // pin za 4 bitnu komunikaciju - D4 na LCD-u
#define LCD_D5_PIN     PA4    // pin za 4 bitnu komunikaciju - D5 na LCD-u
#define LCD_D6_PIN     PA5    // pin za 4 bitnu komunikaciju - D6 na LCD-u
#define LCD_D7_PIN     PA6    // pin za 4 bitnu komunikaciju - D7 na LCD-u

#define LCD_RS_PORT    PORTA // port za odabir registra - RS na LCD-u
#define LCD_RS_PIN     PA1    // pin za odabir registra - RS na LCD-u
#define LCD_E_PORT     PORTA // port za odobrenje upisa - EN na LCD-u
#define LCD_E_PIN      PA2    // pin za odobrenje upisa - EN na LCD-u

#define LCD_LINES      4     // broj vidljivih linija na LCD displeju
#define LCD_DISP_LENGTH 20   // broj vidljivih znakova po liniji na LCD displeju
// kraj konfiguracije LCD displeja
//-----
```



### Vježba 5.1.1

Napravite program koji će na LCD displeju na početku prvog retka ispisati Vaše ime, a na početku drugog retka Vaše prezime bez dijakritičkih znakova. Postavite kašnjenje u programu od dvije sekunde pa ispišite tekst **Veleuciliste** u **Bjelovaru** pravilno raspoređen u dva retka LCD displeja. Ponovno postavite kašnjenje u programu od dvije sekunde. Navedene ispise na LCD displeju neprestano izmjenjujte. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba511.cpp`. Omogućite samo prevođenje datoteke `vjezba511.cpp`. Početni sadržaj datoteke `vjezba511.cpp` prikazan je programskim kodom 5.3.

Programski kod 5.3: Početni sadržaj datoteke `vjezba511.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {
    inicijalizacija();

    while(1) {
        lcd_clrscr();
        lcd_home();
        lcd_print("Ivica\nIvic");
        _delay_ms(2000);

        lcd_clrscr();
        lcd_home();
        lcd_gotoxy(0,1);
        lcd_print("Veleuciliste u");
        lcd_gotoxy(1,3);
        lcd_print("Bjelovaru");
        _delay_ms(2000);
    }
    return 0;
}
```

Funkcije koje se koriste za ispisivanje teksta na LCD displej definirane su u zaglavlju `lcd.h`. Naredba kojom uključujemo zaglavlje `lcd.h` u datoteku koja se prevodi jest `#include "LCD/lcd.h"`. U programskom kodu 5.3 nalazi se niz naredbi koje služe za rad s LCD displejom:

- `lcd_init` - funkcija koja inicijalizira postavke LCD displeja. U ovoj se funkciji konfiguriraju podatkovni i upravljački pinovi mikroupravljača koji komuniciraju s LCD displejom. Osim konfiguracije, funkcija `lcd_init` namješta 4-bitnu komunikaciju, briše LCD displej i postavlja kursor u prvi redak i prvi stupac LCD displeja.
- `lcd_clrscr` - funkcija kojom se briše tekst na LCD displeju.

- `lcd_home` - funkcija kojom se kursor postavlja u prvi redak i prvi stupac LCD displeja.
- `lcd_gotoxy` - funkcija koja prima dva argumenta. Prvi argument postavlja kursor u redak  $x$ , a drugi argument postavlja kursor u stupac  $y$ . Prvi redak ima indeks  $x = 0$ , a prvi stupac ima indeks  $y = 0$ .
- `lcd_print` - funkcija koja služi za ispis teksta na LCD displej. Sintaksa funkcije `lcd_print` identična je sintaksi funkcije `printf` koja je standardna funkcija programskog jezika C.

Sve funkcije inicijalizacije mikroupravljača uvijek ćemo pozivati u funkciji `inicijalizacija` u kojoj se nalazi i funkcija za inicijalizaciju LCD displeja `lcd_init`.

Prevedite datoteku `vjezba511.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite ispis imena i prezimena na sredinu LCD displeja, a tekst `Veleuciliste u Bjelovaru` poravnajte desno na LCD displeju.

Prevedite datoteku `vjezba511.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba511.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.2

Napravite program koji će na LCD displeju na početku prvog retka ispisati proizvoljan cijeli broj tipa `int`, a na početku drugog retka proizvoljan realan broj tipa `float` na dva decimalna mjesta. Postavite kašnjenje u programu od dvije sekunde, a zatim na početku prvog retka ispišite proizvoljan cijeli broj tipa `int32_t`, a na početku drugog retka proizvoljan cijeli broj tipa `uint32_t`. Ponovno postavite kašnjenje u programu od dvije sekunde. Navedene ispise na LCD displeju neprestano izmjenjujte. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba512.cpp`. Omogućite samo prevođenje datoteke `vjezba512.cpp`. Početni sadržaj datoteke `vjezba512.cpp` prikazan je programskim kodom 5.4.

Programski kod 5.4: Početni sadržaj datoteke `vjezba512.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {
    inicijalizacija();

    // deklaracija podataka
    int a = -123;
    float b = 3.14;
    lcd_clrscr();
    lcd_home();
    lcd_print("int: %d \n", a);
```

```

    lcd_print("float: %0.2f ", b);
    return 0;
}

```

U programskom kodu 5.4 prikazan je dio programskog koda koji samo jednom na LCD displej u gornjem retku ispiše cijeli broj -123, a u donjem retku realan broj 3.14. Primijetite da funkcija `lcd_print` prima iste argumente kao i funkcija `printf` u programskom jeziku C. Tipovi podataka koji se koriste u programskom razvojnom okruženju *Atmel Studio 7* (tablica 5.1) razlikuju se od tipova podataka koji se koriste na standardnim računalima s operacijskim sustavom *Windows*. Na primjer, cjelobrojni podatak tipa `int` na standardnim računalima širine je 32 bita, dok je na AVR mikroupravljačima širine 16 bitova. U programskom razvojnom okruženju *Atmel Studio 7* postoje definirani tipovi podataka koji su izvedeni iz standardnih tipova podataka (tablica 5.1). Na primjer, `int8_t` cijeli je broj s predznakom širine 8 bitova, dok je `uint16_t` cijeli broj bez predznaka širine 16 bitova. U tablici 5.1 prikazani su svi definirani i standardni tipovi podataka s brojem bitova koje zauzimaju u podatkovnoj memoriji, minimalnim i maksimalnim vrijednostima te formatom za ispis pomoću funkcije `lcd_print`.

Tablica 5.1: Tipovi podataka koji se koriste u programskom razvojnom okruženju *Atmel Studio 7*

Definirani tip podatka	Standardni tip podatka	Broj bitova	Min.	Max.	printf format
-	<code>char</code>	8	-128	127	<code>%c</code>
<code>int8_t</code>	<code>signed char</code>	8	-128	127	<code>%d</code>
<code>uint8_t</code>	<code>unsigned char</code>	8	0	255	<code>%u</code>
<code>int16_t</code>	<code>int</code> <code>signed int</code>	16	-32768	32767	<code>%d</code>
<code>uint16_t</code>	<code>unsigned int</code>	16	0	65535	<code>%u</code>
<code>int32_t</code>	<code>long int</code> <code>signed long int</code>	32	-2147483648	2147483647	<code>%ld</code>
<code>uint32_t</code>	<code>unsigned long int</code>	32	0	4294967295	<code>%lu</code>
-	<code>float</code>	32	$1.175494 \times 10^{-38}$	$3.402823 \times 10^{38}$	<code>%f</code>
-	<code>double</code>	32	$1.175494 \times 10^{-38}$	$3.402823 \times 10^{38}$	<code>%f</code>

U programski kod 5.4 ubacite `while` petlju u kojoj će se svake dvije sekunde izmjenjivati ispis na LCD displeju prema sljedećim uputama:

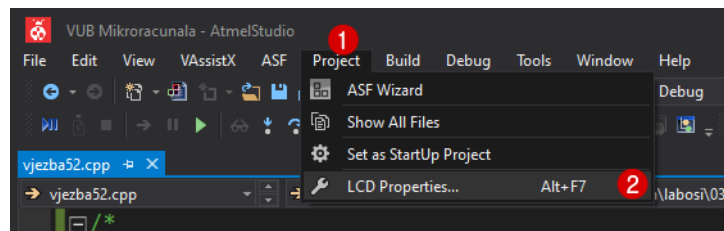
- na početku prvog retka ispisati proizvoljan cijeli broj tipa `int`, a na početku drugog retka ispisati proizvoljan realan broj tipa `float` na dva decimalna mjesta,
- postavite kašnjenje u programu od dvije sekunde,
- na početku prvog retka ispisati proizvoljan cijeli broj tipa `int32_t`, a na početku drugog retka ispisati proizvoljan cijeli broj tipa `uint32_t`,
- postavite kašnjenje u programu od dvije sekunde.

Pri izvedbi ove vježbe držite se definiranih tipova podataka i `printf` formata iz tablice 5.1.

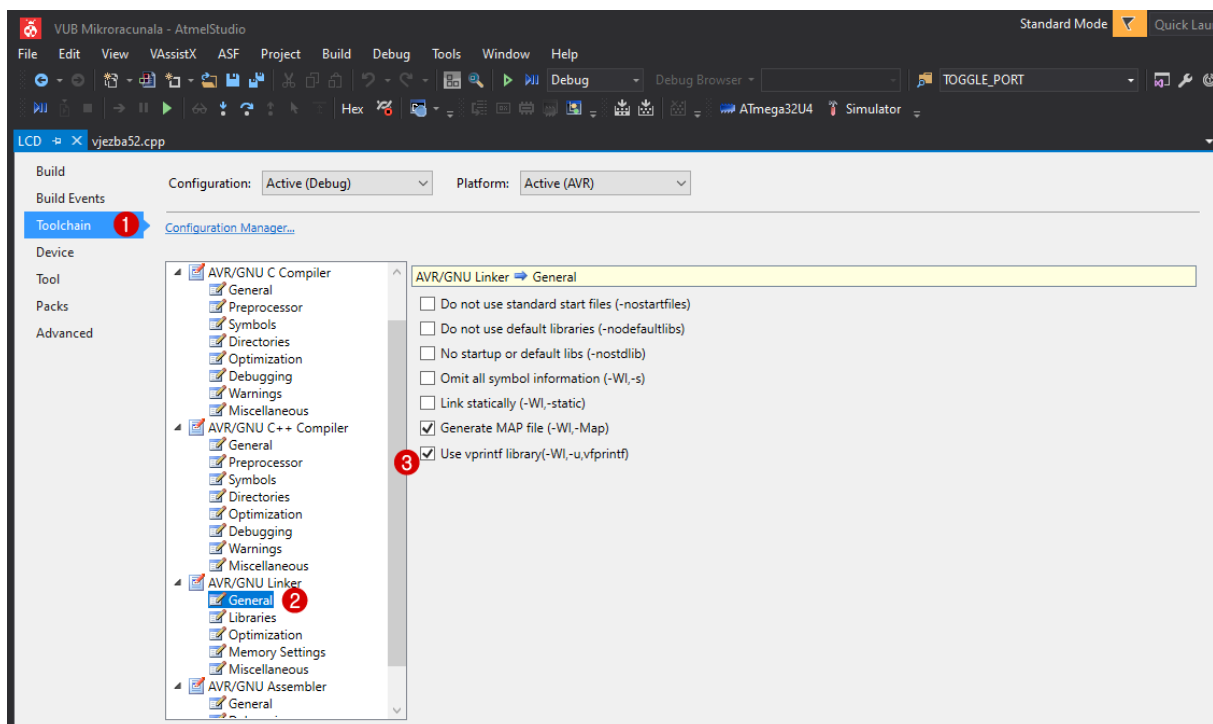
Prevedite datoteku `vjezba512.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ukoliko pri ispisu brojeva s posmičnim zarezom (engl. *floating point*) pomoću funkcije `lcd_print` na LCD displeju umjesto željenog broja dobijete `?`, potrebno je omogućiti ispis brojeva s posmičnim zarezom. U programskom okruženju *Atmel Studio 7* početne su postavke takve da

je onemogućen ispis brojeva s posmičnim zarezom u funkcijama koje pozivaju funkcije `printf` (kao što je `lcd_print`) ili `printf` radi uštede programske memorije mikroupravljača. Ako je nužan ispis brojeva s posmičnim zarezom, tada je potrebno napraviti sljedeće korake:

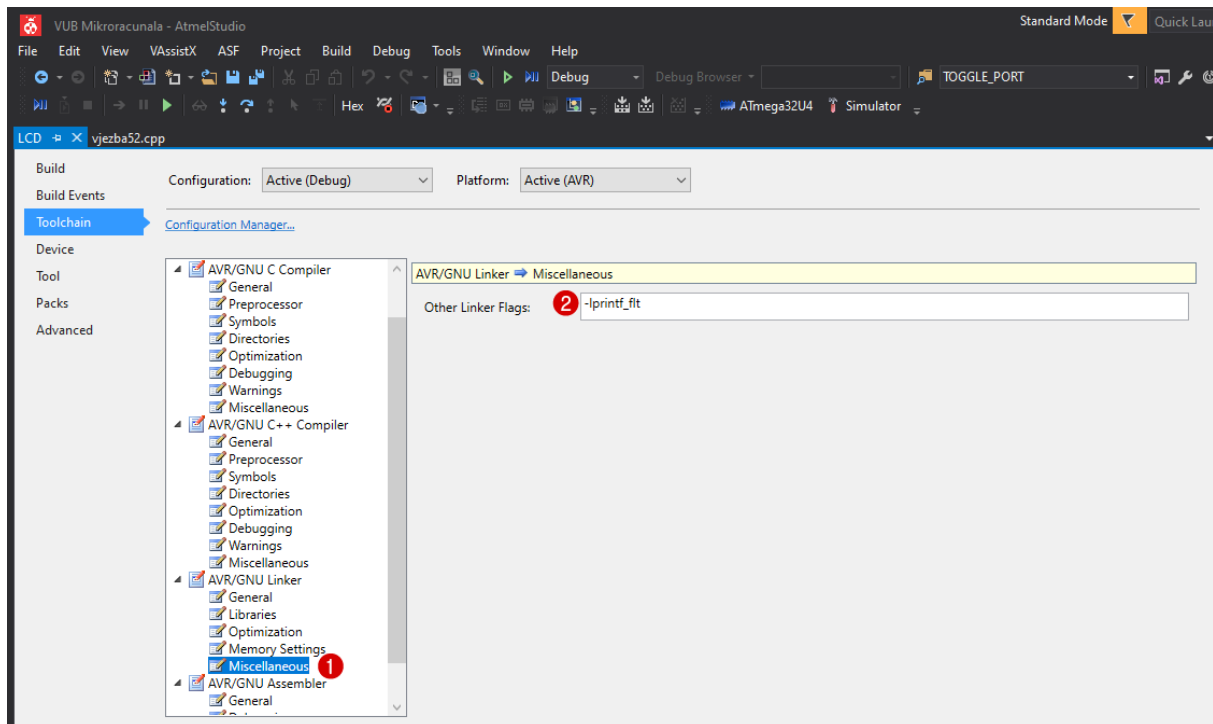
- u programskom okruženju *Atmel Studio 7* odaberite izbornik *Project* → *LCD Properties...* ili pritisnite *Alt+F7* (slika 5.3),
- u otvorenom prozoru LCD odaberite (slika 5.4):
  1. *Toolchain*,
  2. *General* u *AVR/GNU Linker*,
  3. označite *Use vprintf library(-Wl,-u,vprintf)*.
- u *Toolchain* i *AVR/GNU Linker* (slika 5.5):
  1. odaberite *Miscellaneous*,
  2. u tekstualni okvir *Other Linker Flags* upišite `-lprintf_flt`.
- snimite nove postavke programskog okruženju *Atmel Studio 7*.



Slika 5.3: Printanje brojeva s posmičnim zarezom (engl. *floating point*) na AVR mikroupravljačima u programskom okruženju *Atmel Studio 7* (1)



Slika 5.4: Printanje brojeva s posmičnim zarezom (engl. *floating point*) na AVR mikroupravljačima u programskom okruženju *Atmel Studio 7* (2)



Slika 5.5: Printanje brojeva s posmičnim zarezom (engl. *floating point*) na AVR mikroupravljačima u programskom okruženju *Atmel Studio 7* (3)

Ponovno prevedite datoteku `vjezba512.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 i provjerite imate li ispravan ispis brojeva s posmičnim zarezom.

Promijenite tipove podataka za ispis te ponovno prevedite datoteku `vjezba512.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba512.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.3

Napravite program koji će u prvom retku LCD displeja svaku sekundu ispisivati slova engleske abecede redom A, B, C, ..., Z, A, B, ... . U drugom je retku LCD displeja po istom principu potrebno ispisivati mala slova engleske abecede svaku sekundu. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba513.cpp`. Omogućite samo prevođenje datoteke `vjezba513.cpp`. Početni sadržaj datoteke `vjezba513.cpp` prikazan je programskim kodom 5.5.

Programski kod 5.5: Početni sadržaj datoteke `vjezba513.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>

void inicijalizacija() {
}

```

```

int main(void) {
    inicijalizacija();

    char a = 'A';

    while (1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("Veliko slovo: %c\n", a);
        lcd_print("Malo slovo: ");

        _delay_ms(1000);

        if (++a > 'Z') {
            a = 'A';
        }
    }
    return 0;
}

```

U programskom kodu 5.5 nedostaje inicijalizacija LCD displeja u funkciji `inicijalizacija()`. Inicijalizirajte LCD displej. Također, potrebno je uključiti zaglavlje `lcd.h` u kojem su definirane sve funkcije za rad s LCD displejom.

U `while` petlji u programskom kodu 5.5 ispisuju se velika slova engleske abecede u prvom retku, dok se u drugom retku ispisuje samo tekst `Malo slovo:`. Nadopunite argument funkcije `lcd_print("Malo slovo: ")`; tako da se u drugom retku ispisuju mala slova engleske abecede. Imajte na umu da su mala slova od velikih slova po ASCII kodu udaljena za 32 (npr. ASCII kod velikog slova A je 65, a malog slova a je 97).

Engleska abeceda sadrži 26 slova. Naredbom `if (++a > 'Z') a = 'A';` provjerava se je li ASCII kod sljedećeg znaka veći od ASCII koda slova Z. Ako je uvjet zadovoljen, varijabla `a` u kojoj se izmjenjuju znakovi postavlja se na vrijednost slova A. Na taj način ostvarili smo neprestano ispisivanje svih slova engleske abecede.

Prevedite datoteku `vjezba513.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba513.cpp` i onemogućite prevođenje ove datoteke.



#### Vježba 5.1.4

Napravite program koji će u prvom retku LCD displeja svaku sekundu ispisivati kut od 0 do 180 °, a u drugom retku sinus tog kuta na tri decimalna mjesta. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba514.cpp`. Omogućite samo prevođenje datoteke `vjezba514.cpp`. Početni sadržaj datoteke `vjezba514.cpp` prikazan je programskim kodom 5.6.

Programski kod 5.6: Početni sadržaj datoteke `vjezba514.cpp`

```

#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>

```



```

#define PI 3.14159

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {
    inicijalizacija();

    uint8_t kut = 0;

    while (1) {
        lcd_clrscr();
        lcd_home();

        // ispis kuta i sinusa kuta

        _delay_ms(1000);

        if(++kut >= 180) {
            kut = 0;
        }
    }
    return 0;
}

```

Kada se u programu koriste matematičke funkcije kao što su sinus, kosinus, logaritam i drugo, tada je u programski kod potrebno uključiti standardno zaglavlje `math.h`. Matematička funkcija sinus prima argument kuta u radijanima pa je kut u stupnjevima potrebno pretvoriti u radijane. Varijablu `kut` deklarirali smo kao cjelobrojnu varijablu bez predznaka širine 8 bitova. Ispod funkcije `lcd_home` u programskom kodu 5.6 upišite sljedeće tri naredbe:

- `lcd_print("Kut:%u%c", kut, 223);` - ispis kuta od 0 do 180 °. U argumentu funkcije `lcd_print` pojavljuje se varijabla `kut` i broj 223. Broj 223 predstavlja ASCII kod znaka ° na LCD displeju.
- `lcd_gotoxy(1,0);` - funkcija koja novi ispis postavlja na početak drugog retka.
- `lcd_print("Sinus kuta:%.3f", sin(PI*kut/180));` - ispis sinusa kuta na tri decimalna mjesta. Pretvorba stupnjeva u radijane postignuta je relacijom  $\frac{\pi \cdot kut}{180}$ .

Na različitim LCD displejima mogući su različiti ASCII kodovi za znak °. Najjednostavniji način za pronalaženje znaka ° jest proći kroz sve ASCII znakove LCD displeja koristeći naredbu `lcd_print("ASCII:%u %c", znak, znak++)`; u petlji. Na ovaj način prezentirate sve ASCII kodove i znakove koji su dostupni na LCD displeju.

Prevedite datoteku `vjezba514.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite preciznost ispisa na više decimalnih mjesta (ne više od sedam). Ograničite promjenu kuta od 30 do 90 °. Prevedite datoteku `vjezba514.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba514.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.5

Napravite program koji će na LCD displeju prikazivati sekunde, minute i sate od trenutka uključanja mikroupravljača. Svake sekunde promijenite stanje plave LED diode. Kada se pritisne tipkalo spojeno na pin PD0, neka se isključi plava LED dioda, a sekunde, minute i sati neka se postave na nulu. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba515.cpp`. Omogućite samo prevođenje datoteke `vjezba515.cpp`. Početni sadržaj datoteke `vjezba515.cpp` prikazan je programskim kodom 5.7.

Programski kod 5.7: Početni sadržaj datoteke `vjezba515.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {

    output_port(DDRB,PB7); //pin PB7 postavljen kao izlaz
    input_port(DDRD,PD0); //pin PD0 postavljen kao ulaz
    set_port(PORTD,PD0,1); // uključen pritezni otpornik na PD0

    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {

    inicijalizacija();

    uint8_t sec = 0;
    uint8_t min = 0;
    uint16_t sat = 0;

    while (1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("%dh:%dm:%ds", sat,min,sec);

        _delay_ms(1000);

        toggle_port(PORTB,PB7);

        if(++sec >= 60) {
            sec = 0;
            min++;
        }

        // napravite niz naredbi koje će računati minute i sate

        if(get_pin(PIND,PD0) == 0) {
            sec = 0;
            min = 0;
            sat = 0;
            set_port(PORTB,PB7,0);
        }
    }
}
```

```
    return 0;
}
```

Cilj je ove vježbe napraviti prikaz vremena rada mikroupravljača u satima, minutama i sekundama. U vježbi se koriste plava LED dioda i tipkalo spojeno na pin PD0, pa je u programskom kodu 5.7 u funkciji `inicijalizacija()` napravljena konfiguracija pinova. Na LCD displeju potrebno je prikazivati sekunde, minute i sate. Proračun sekundi, minuta i satova morat ćemo napraviti na temelju kašnjenja `while` petlje. U `while` petlji postavili smo kašnjenje od jedne sekunde. Pretpostavimo da se ostali niz naredbi u `while` petlji izvodi u vremenu koje je zanemarivo naspram jedne sekunde. Svaki će prolaz kroz `while` petlju tada trajati jednu sekundu. U svakom prolazu kroz `while` petlju varijablu `sec` uvećavamo za jedan i mijenjamo stanje plave LED diode makronaredbom `toggle_port`. Kada broj sekundi u varijabli `sec` dosegne 60, varijablu `sec` postavljamo na nulu, a varijablu `min` uvećamo za jedan. Dovršite niz naredbi koje će, kada varijabla `min` dosegne 60, varijablu `min` postaviti na nulu, a varijablu `sat` uvećati za jedan. Koliko je maksimalno vrijeme koje se može prikazati na LCD displeju? Zašto?

Stanje tipkala spojenog na pin PD0 provjeravamo makronaredbom `get_pin`. Ako makronaredba vrati vrijednost 0 (slučaj kada je tipkalo pritisnuto), tada će se isključiti plava LED dioda, a varijable `sec`, `min` i `sat` postaviti u vrijednost 0.

Prevedite datoteku `vjezba515.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba515.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.6

Napravite program koji će na LCD displeju ispisivati dijakritičke znakove č, ć, đ, š i ž. Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Za definiranje znakova potrebno je koristiti aplikaciju LCD konverter dostupnu na mrežnoj stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba516.cpp`. Omogućite samo prevođenje datoteke `vjezba516.cpp`. Početni sadržaj datoteke `vjezba516.cpp` prikazan je programskim kodom 5.8.

Programski kod 5.8: Početni sadržaj datoteke `vjezba516.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
    lcd_define_char(); //definiranje novih znakova
}

int main(void) {
    inicijalizacija();
    lcd_clrscr();
    lcd_home();
    lcd_char(0x00);
    lcd_print("%c%c%c%c%c%c%c", 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07);
    return 0;
}
```

LCD displej ima slobodan memorijski prostor za definiranje osam vlastitih znakova. Ti znakovi nalaze se na prvih osam memorijskih lokacija LCD displeja s adresama od 0x00 do 0x07. Broj memorijske lokacije znaka na LCD displeju odgovara ASCII kodu tog znaka. Funkcija `lcd_print` na LCD displej šalje putem ASCII kodova adrese memorijskih lokacija znakova koje trebaju biti prezentirane u nekom polju LCD displeja.

U programskom kodu 5.8 nalazi se funkcija `lcd_define_char`. Ova funkcija definira znakove na LCD displeju odmah nakon inicijalizacije LCD displeja. Jednom kad definirate znakove u LCD displeju, oni tamo ostaju pohranjeni bez obzira na napajanje LCD displeja.

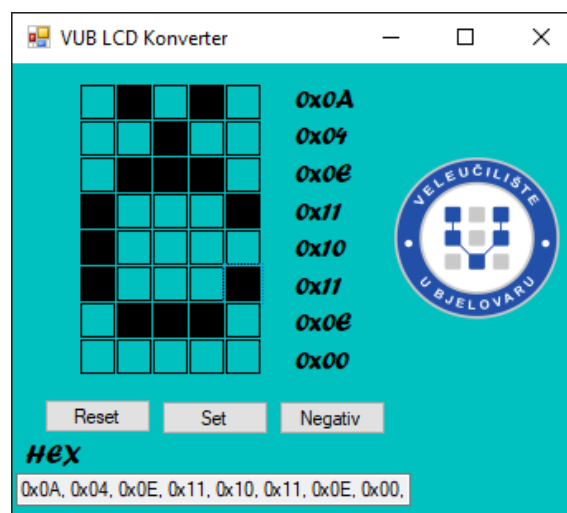
Iznad funkcije `lcd_define_char` pritisnite desni gumb miša i odaberite *Goto Implementation*. Otvorit će se datoteka `lcd.cpp` u kojoj se nalazi definicija funkcije `lcd_define_char`. Na samom početku definicije funkcije `lcd_define_char` nalazi se dvodimenzionalno polje prikazano programskim kodom 5.9.

Programski kod 5.9: Dvodimenzionalno polje za definiranje znakova

```
const uint8_t symbol[8][8] = { //definirajte 8 znakova

    /* 0x00 */ 0x0A, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00, //č
    /* 0x01 */ 0x02, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00, //ć
    /* 0x02 */ 0x0A, 0x04, 0x0E, 0x10, 0x0E, 0x01, 0x1E, 0x00, //š
    /* 0x03 */ 0x02, 0x07, 0x02, 0x0E, 0x12, 0x12, 0x0E, 0x00, //đ
    /* 0x04 */ 0x0A, 0x04, 0x1F, 0x02, 0x04, 0x08, 0x1F, 0x00, //ž
    /* 0x05 */ 0x02, 0x05, 0x04, 0x04, 0x04, 0x14, 0x08, 0x00, //integral
    /* 0x06 */ 0x15, 0x1B, 0x11, 0x0E, 0x0F, 0x0E, 0x11, 0x1F, //č negativ
    /* 0x07 */ 0x15, 0x1B, 0x11, 0x0F, 0x11, 0x1E, 0x01, 0x1F, //š negativ
};
```

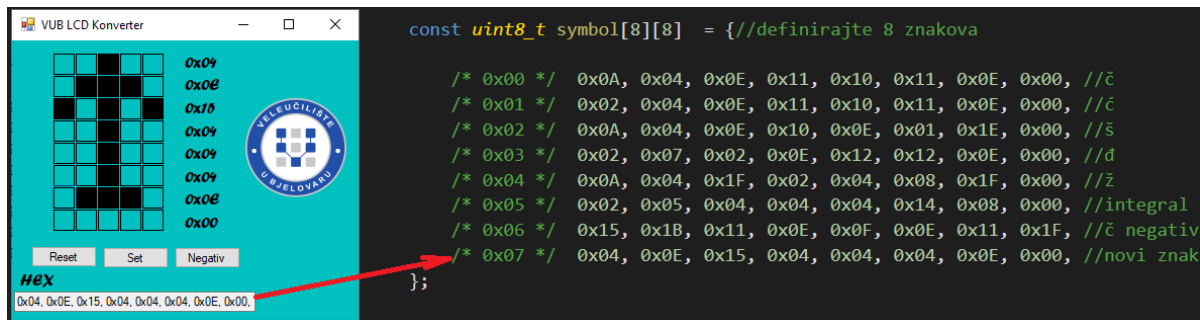
U programskom kodu 5.9 blok komentarima označene su memorijske lokacije definiranih znakova. Svaki je znak jedan redak u dvodimenzionalnom polju. Ako želite definirati vlastiti znak, potrebno je koristiti aplikaciju LCD konverter dostupnu na stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Skinite aplikaciju i pokrenite ju. Aplikacija LCD konverter prikazana je na slici 5.6.



Slika 5.6: Aplikacija LCD konverter

Matrica za izradu znakova na aplikaciji LCD konverter dimenzijom odgovara znaku na LCD displeju. Aplikacija LCD konverter stvara 8 bajtova podataka u heksadecimalnom zapisu koje je potrebno kopirati u dvodimenzionalno polje koje se nalazi u tijelu funkcije `lcd_define_char` (slika 5.7) u jedan od osam redaka. Kada kopiramo novi znak u funkciju `lcd_define_char`, moći

ćemo ga koristiti pomoću indeksa memorijske lokacije koji je jednak indeksu retka u koji je znak kopiran.



Slika 5.7: Definiranje novog znaka

Definirane znakove možemo ispisivati na LCD displej pomoću dviju funkcija:

- `lcd_char(char adresa)` - funkcija koja će na trenutnu poziciju kursora LCD displeja ispisati znak na memorijskoj lokaciji `adresa`.
- `lcd_print("%c%c...", adresa1, adresa2, ...)`; - funkcija koja će na LCD displej ispisati znakove koji su na memorijskoj lokaciji `adresa1, adresa2, ...`.

Napomenuli smo da funkcija `lcd_print` ima istu sintaksu kao standardna funkcija `printf` programskog jezika C. Tekst koji kreira funkcija `lcd_print` niz je ASCII kodova koji se proslijeđuju na LCD displej. U programskom jeziku C svaki je niz znakova (engl. *string*) „zaključan” tzv. *null* znakom čiji je ASCII kod jednak 0x00. Iz tog se razloga jedino memorijska lokacija 0x00 ne može koristiti u funkciji `lcd_print` jer bi prevoditelj znak s ASCII kodom 0x00 shvatio kao kraj teksta koji ispisuje. Zbog toga se znak na adresi 0x00 mora ispisivati pomoću funkcije `lcd_char`.

Pretpostavimo da je slovo č definirano na memorijskoj lokaciji 0x01. Ako pozovemo funkciju `lcd_print("Mikroračunala", 0x01)`;, na LCD displeju ispisat će se tekst Mikroračunala jer se na mjesto specifikatora `%c` ispisuje znak koji se nalazi na adresi 0x01.

Prevedite datoteku `vjezba516.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Definirajte osam vlastitih znakova, pohranite ih u dvodimenzionalno polje u funkciji `lcd_define_char` te ponovno prevedite datoteku `vjezba516.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba516.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.7

Napravite program koji će na LCD displeju svake dvije sekunde izmjenjivati tekstove Veleučilište u Bjelovaru, Stručni studij Mehatronika i Stručni studij Računarstvo. Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Za definiranje znakova potrebno je koristiti aplikaciju LCD konverter dostupnu na stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba517.cpp`. Omogućite samo prevođenje datoteke `vjezba517.cpp`. Početni sadržaj datoteke `vjezba517.cpp` prikazan je programskim kodom 5.10.

Programski kod 5.10: Početni sadržaj datoteke `vjezba517.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
    lcd_define_char(); //definiranje novih znakova
}

int main(void) {
    inicijalizacija();

    while (1) {

        lcd_clrscr();
        lcd_home();

        // nastavite ...
    }
    return 0;
}
```

Na početku vježbe prvo definirajte znakove koji se koriste u ispisu tekstova **Veleučilište u Bjelovaru, Stručni studij Mehatronika i Stručni studij Računarstvo** na LCD displej. Ukoliko neki od korištenih znakova definirate na lokaciji 0x00, za ispis tog znaka koristite funkciju `lcd_char`.

U programskom kodu 5.10 potrebno je nastaviti niz naredbi koje će na LCD displeju svake dvije sekunde izmjenjivati tekstove **Veleučilište u Bjelovaru, Stručni studij Mehatronika i Stručni studij Računarstvo**.

Prevedite datoteku `vjezba517.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Proizvoljno ispišite tekst koji sadrži dijakritičke znakove na LCD displej te ponovno prevedite datoteku `vjezba517.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba517.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 5.1.8

Napravite program koji će na LCD displeju u gornjem retku ispisati tekst **Materijali se nalaze na web stranici:**, a u donjem retku `www.vub.hr/mikroracunala`. Tekst je potrebno svaku sekundu pomaknuti za jedno mjesto ulijevo. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

U projektnom stablu otvorite datoteku `vjezba518.cpp`. Omogućite samo prevođenje datoteke `vjezba518.cpp`. Početni sadržaj datoteke `vjezba518.cpp` prikazan je programskim kodom 5.11.

Programski kod 5.11: Početni sadržaj datoteke vjezba518.cpp

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>

void inicijalizacija() {

    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {

    inicijalizacija();

    lcd_clrscr();
    lcd_home();
    lcd_print("Materijali se nalaze na web stranici:\n");
    lcd_print("www.vub.hr/mikroracunala");

    while (1) {
        lcd_instr(LCD_ENTRY_INC_SHIFT);
        lcd_instr(LCD_MOVE_DISP_LEFT);
        _delay_ms(1000);
    }
    return 0;
}
```

Tekstovi Materijali se nalaze na web stranici: i [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) imaju više od 16 znakova, što izlazi iz vidnog područja LCD displeja GMD1602E. Svaki redak može pohraniti najviše 40 znakova koje možemo pomicati ulijevo ili udesno. U programskom kodu 5.11 u `while` petlji nalaze se naredbe koje omogućuju pomicanje teksta i određuju smjer pomicanja. U svrhu pomicanja teksta korištena je funkcija `lcd_instr` koja šalje instrukcije na LCD displej. Instrukcija `LCD_ENTRY_INC_SHIFT` omogućuje pomicanje teksta na LCD displeju, dok instrukcija `LCD_MOVE_DISP_LEFT` određuje smjer pomicanja teksta ulijevo. Popis instrukcija koje se mogu koristiti za rad LCD displeja nalaze se u zaglavlju `lcd.h`.

Prevedite datoteku `vjezba518.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba518.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 5.1.1 Zadaci - LCD displej

#### **Zadatak 5.1.1**

Napravite program u `while` petlji koji će na LCD displeju u prvom retku desno poravnato ispisati Vaše ime, a u drugom retku desno poravnato Vaše prezime bez diakritičkih znakova. Postavite kašnjenje u programu od tri sekunde pa ispišite tekst `Položit cemo Mikroracunala` pravilno raspoređen u dva retka LCD displeja. Ponovno postavite kašnjenje u programu od tri sekunde. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---

#### **Zadatak 5.1.2**

Napravite program koji će na LCD displeju na početku prvog retka ispisati proizvoljan cijeli broj tipa `int8_t`, a na početku drugog retka proizvoljan realan broj tipa `double` na četiri decimalna mjesta. U programu postavite kašnjenje od dvije sekunde, a zatim na početku prvog retka ispisati proizvoljan cijeli broj tipa `int16_t`, a na početku drugog retka proizvoljan cijeli broj tipa `uint16_t`. Ponovno postavite kašnjenje u programu od dvije sekunde. Navedene ispise na LCD displeju neprestano izmjenjujte. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---

#### **Zadatak 5.1.3**

Napravite program koji će u prvom retku LCD displeja svake sekunde ispisivati znakove dostupne na LCD displeju. To su znakovi s ASCII kodom od 0 do 255. U drugom je retku LCD displeja potrebno ispisati ASCII kod znaka iz prvog retka. ASCII kod ispisujte `printf` formatom `%u`. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---

#### **Zadatak 5.1.4**

Napravite program koji će u prvom retku LCD displeja svake sekunde ispisivati kut od 0 do 180 °, a u drugom retku kosinus tog kuta na dva decimalna mjesta. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.


---

#### **Zadatak 5.1.5**

Napravite program koji će na LCD displeju prikazivati desetinke sekunde, sekunde, minute i sate od trenutka uključenja mikroupravljača. Svake sekunde promijenite stanje zelene LED diode. Ukoliko se pritisne tipkalo spojeno na pin PF6, neka se isključi zelena LED dioda, a desetinke sekunde, sekunde, minute i sati neka se postave na nulu. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---



** Zadatak 5.1.6**

Napravite program koji će na LCD displeju ispisivati proizvoljne znakove definirane pomoću aplikacije LCD konverter dostupne na mrežnoj stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---

** Zadatak 5.1.7**

Napravite program koji će na LCD displeju svake dvije sekunde izmjenjivati tekstove MIKRORAČUNALO ATmega32U4 i <VJEŽBA>. Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Za definiranje znakova potrebno je koristiti aplikaciju LCD konverter dostupnu na mrežnoj stranici [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---

** Zadatak 5.1.8**

Napravite program koji će na LCD displeju u gornjem retku ispisati tekst **Veleučilište** u Bjelovaru, a u donjem retku **Stručni studiji Mehatronika i Računarstvo**. Tekst je potrebno svake sekunde pomaknuti za jedno mjesto ulijevo. Shema spajanja LCD displeja na razvojno okruženje s mikroupravljačem ATmega32U4 prikazana je na slici 5.1.

---



## Poglavlje 6

# EEPROM memorija

EEPROM memorija (engl. *Electrically Erasable Programmable Read-Only Memory*) vrsta je trajne memorije koja za čuvanje pohranjenih podataka ne treba električno napajanje. Najčešće se koristi za pohranu parametara sustava koji se mogu mijenjati tijekom rada sustava. Na primjer, PI regulator sustava ima dva parametra koji se u adaptivnom algoritmu regulacije mijenjaju. Ti se parametri ne smiju izgubiti bez obzira na gubitak napajanja u sustavu i bez obzira na privremeni prestanak rada sustava. U složenijim se sustavima u EEPROM memoriju snimaju složene konfiguracije sustava koje moraju biti dostupne prilikom inicijalizacije sustava.

### 6.1 Vježbe - EEPROM memorija

U mikroupravljaču ATmega32U4 nalazi se EEPROM memorija veličine 1024 bajta. Radni je vijek EEPROM memorije do najviše 100 000 ciklusa pisanja i čitanja. EEPROM memorija u mikroupravljaču ATmega32U4 odvojena je memorija koja nije u sklopu arhitekture mikroupravljača ATmega32U4 te se zbog toga za adresiranje podataka koristi poseban EEPROM registar [1].

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku EEPROM memorija.zip. Na radnoj površini stvorite praznu datoteku koju ćete nazvati *Vaše Ime i Prezime* ne koristeći pritom diakritičke znakove. Na primjer, ako je Vaše ime Ivica Ivić, datoteka koju ćete stvoriti zvat će se *Ivica Ivic*. Datoteku EEPROM memorija.zip raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite VUB mikroracunala.atstln u datoteci \\EEPROM memorija\vjezbe. U otvorenom projektu nalaze se sve vježbe koje ćemo obraditi u poglavlju EEPROM memorija. Vježbe ćemo pisati u datoteke s ekstenzijom \*.cpp.

U datoteci s vježbama nalaze se i rješenja vježbi koje možete koristiti za provjeru ispravnosti programskih zadataka.



#### Vježba 6.1.1

Napravite program u kojem će se cijeli brojevi tipa `int8_t`, `int16_t` i `int32_t` te jedan realan broj tipa `float` spremati u EEPROM memoriju na padajući brid tipkala spojenog na pin PD0. Nad navedenim brojevima potrebno je izvršiti proizvoljne jednostavne matematičke operacije. Brojeve je potrebno prikazivati na LCD displeju. Prilikom uključivanja mikroupravljača brojeve tipa `int8_t`, `int16_t`, `int32_t` i `float` inicijalizirajte čitanjem iz EEPROM memorije.

---

U projektnom stablu otvorite datoteku `vjezba611.cpp`. Omogućite samo prevođenje datoteke `vjezba611.cpp`. Početni sadržaj datoteke `vjezba611.cpp` prikazan je programskim kodom 6.1.

Programski kod 6.1: Početni sadržaj datoteke `vjezba611.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <avr/eeprom.h>

int8_t cijeli8_t;
int16_t cijeli16_t;
int32_t cijeli32_t;
float realan;

void inicijalizacija() {

    input_port(DDRD,PDO);
    set_port(PORTD,PDO,1);

    cijeli8_t = eeprom_read_byte(addr_byte(0x00));
    cijeli16_t = eeprom_read_word(addr_word(0x01));
    cijeli32_t = eeprom_read_dword(addr_dword(0x03));
    realan = eeprom_read_float(addr_float(0x07));

    lcd_init();
}

int main(void) {

    inicijalizacija();

    while(1) {

        cijeli8_t++;
        cijeli16_t +=2;
        cijeli32_t +=3;
        realan = 1.1 * cijeli8_t;

        // ispitivanje padajućeg brida na PDO
        if(isFalling_edge(PD, PDO) == true) {
            eeprom_write_byte(addr_byte(0x00),cijeli8_t);
            eeprom_write_word(addr_word(0x01),cijeli16_t);
            eeprom_write_dword(addr_dword(0x03),cijeli32_t);
            eeprom_write_float(addr_float(0x07),realan);
        }

        lcd_clrscr();
        lcd_home();
        lcd_print("C8: %d,C16: %d", cijeli8_t, cijeli16_t);
        lcd_print("\nC32: %ld,R: %.2f", cijeli32_t, realan);
        _delay_ms(1000);
    }
    return 0;
}
```

Za čitanje podataka iz EEPROM memorije i pisanje podataka u EEPROM memoriju koriste se sljedeće funkcije:

- `uint8_t eeprom_read_byte(addr_byte(x))` - funkcija koja čita cijeli broj širine 8 bitova s adrese `x`.

- `uint16_t eeprom_read_word(addr_word(x))` - funkcija koja čita cijeli broj širine 16 bitova s adrese `x`.
- `uint32_t eeprom_read_dword(addr_dword(x))` - funkcija koja čita cijeli broj širine 32 bita s adrese `x`.
- `float eeprom_read_float(addr_float(x))` - funkcija koja čita realan broj širine 32 bita s adrese `x`.
- `eeprom_write_byte(addr_byte(x), data)` - funkcija koja zapisuje cijeli broj `data` širine 8 bitova na adresu `x`.
- `eeprom_write_word(addr_word(x), data)` - funkcija koja zapisuje cijeli broj `data` širine 16 bitova na adresu `x`.
- `eeprom_write_dword(addr_dword(x), data)` - funkcija koja zapisuje cijeli broj `data` širine 32 bita na adresu `x`.
- `eeprom_write_float(addr_float(x), data)` - funkcija koja zapisuje realan broj `data` širine 32 bita na adresu `x`.

Navedene funkcije definirane su u standardnom zaglavlju `eeprom.h`. Zaglavlje `eeprom.h` u programski je kod 6.1 uključeno naredbom `#include <avr/eeprom.h>`. Za kreiranje adresa kojima se pristupa u EEPROM memoriji koriste se makronaredbe koje se nalaze u zaglavlju `avrvub.h`:

- `addr_byte(x)` - makro naredba koja inicijalizira pokazivač tipa `uint8_t` na adresu `x`,
- `addr_word(x)` - makro naredba koja inicijalizira pokazivač tipa `uint16_t` na adresu `x`,
- `addr_dword(x)` - makro naredba koja inicijalizira pokazivač tipa `uint32_t` na adresu `x`,
- `addr_float(x)` - makro naredba koja inicijalizira pokazivač tipa `float` na adresu `x`.

U programskom kodu 6.1 deklarirali smo tri cijela broja tipa `int8_t`, `int16_t` i `int32_t` te jedan realan broj tipa `float`. Ovi brojevi deklarirani su kao globalne varijable kako bi bili dostupni svim funkcijama u datoteci `vjezba611.cpp`. U funkciji `inicijalizacija()` iz EEPROM memorije čitamo podatke.

Na padajući brid signala tipkala spojenog na pin PD0 trenutne vrijednosti tri cijela broja tipa `int8_t`, `int16_t` i `int32_t` te jednog realnog broj tipa `float` spremaju se u EEPROM memoriju. Prilikom čitanja podatka i zapisivanja podatka u EEPROM memoriju moramo voditi računa o veličini podatka koji čitamo, odnosno zapisujemo. Jedna memorijska lokacija EEPROM memorije širine je 8 bitova (1 bajt). Na primjer, ako na memorijsku lokaciju `0x01` zapisujemo cijeli broj širine 16 bitova, tada taj broj zauzima memorijske lokacije `0x01` i `0x02` (slika 6.1). Sljedeća slobodna lokacija za zapisivanje podataka jest `0x03`. Ako na memorijsku lokaciju `0x03` zapisujemo cijeli broj širine 32 bita, tada taj broj zauzima memorijske lokacije `0x03`, `0x04`, `0x05` i `0x06` (slika 6.1). Sljedeća slobodna lokacija za zapisivanje podataka jest `0x07`. Podatke moramo čitati s onih memorijskih lokacija na koje smo ih zapisali. Pri tome je potrebno voditi brigu o veličini podatka koji čitamo.

Adresa EEPROM memorije	EEPROM memorija
0x00	<code>int8_t</code>
0x01	<code>int16_t</code>
0x02	
0x03	<code>int32_t</code>
0x04	
0x05	
0x06	
0x07	<code>float</code>
0x08	
0x09	
0x10	
0x11	
0x12	

Slika 6.1: EEPROM meomorija - zauzeće memorije

Prevedite datoteku `vjezba611.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 tako da u jednom trenutku pritisnete tipkalo spojeno na pin PD0. Nakon toga isključite napajanje mikroupravljača ATmega32U4 te ga ponovno uključite. Jesu li podaci ostali sačuvani bez obzira na nestanak napajanja?

Zatvorite datoteku `vjezba611.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 6.1.1 Zadaci - EEPROM memorija

#### Zadatak 6.1.1

Napravite program u kojem će se cijeli brojevi tipa `uint8_t`, `uint16_t`, `int8_t`, `int16_t` i `uint32_t` te jedan realan broj tipa `double` spremati u EEPROM memoriju na padajući brid tipkala spojenog na pin PD1. Nad navedenim brojevima potrebno je izvršiti proizvoljne jednostavne matematičke operacije. Brojeve je potrebno prikazivati na LCD displeju. Prilikom uključivanja mikroupravljača brojeve tipa `uint8_t`, `uint16_t`, `int8_t`, `int16_t`, `uint32_t` i `double` inicijalizirajte čitanjem iz EEPROM memorije.

## Poglavlje 7

# Analogno-digitalna pretvorba

Analogno-digitalna pretvorba koristi se za prezentaciju analognih veličina s analognih senzora u mikroupravljaču. Analogni senzori mjerni su uređaji koji najčešće neelektrične veličine pretvaraju u električne (npr. temperaturu u napon, tlak u napon, vlagu u napon i drugo). 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. Podatak o mjerenoj veličini koristi se pri upravljanju i nadzoru sustava. Mikroupravljači u pravilu imaju pinove koji se mogu koristiti za analogno-digitalnu pretvorbu.

U praktičnoj primjeni često se koriste sljedeći analogni senzori:

- potenciometar - koristi se za mjerenje pozicije ili za namještanje referentnih veličina,
- NTC otpornik - koristi se za mjerenje temperature,
- LM35 - koristi se za mjerenje temperature s visokom preciznošću,
- ACS712 - koristi se za mjerenje struje,
- termopar K tipa - koristi se za mjerenje temperature,
- HR202 - koristi se za mjerenje tlaka ukoliko je poznata temperatura.

### 7.1 Vježbe - analogno-digitalna pretvorba

Mikroupravljač ATmega32U4 ima 12 analognih ulaza raspoređenih na sljedeće pinove: ADC0 (PF0), ADC1 (PF1), ADC4 (PF4), ADC5 (PF5), ADC6 (PF6), ADC7 (PF7), ADC8 (PD4), ADC9 (PD6), ADC10 (PD7), ADC11 (PB4), ADC12 (PB5) i ADC13 (PB6).

Kao što možemo primijetiti, svi navedeni pinovi višenamjenski su jer osim što se mogu koristiti kao digitalni, mogu se koristiti i kao analogni. Analogno-digitalni pretvornik u mikroupravljaču ATmega32U4 pretvara analognu vrijednost napona u digitalnu riječ širine 10 bitova metodom sukcesivne aproksimacije [3]. Brzina analogno-digitalne pretvorbe može biti od 65 do 260  $\mu$ s. Shema spajanja analognih senzora na mikroupravljač ATmega32U4 prikazana je na slici 7.1. Analogni senzori spojeni na mikroupravljač ATmega32U4 jesu potenciometar, NTC otpornik i temperaturni senzor LM35. Potenciometar je preko kratkospojnika JP1 (između trnova 2 i 3) spojen na pin PF5 (ADC5), a NTC otpornik je preko kratkospojnika JP2 (između trnova 1 i 2) spojen na pin PF4 (ADC4). Temperaturni senzor LM35 spojen je na pin PF5 (ADC5) preko kratkospojnika JP1 (između trnova 1 i 2), a koristit će se u budućim vježbama.





Pretpostavimo da napon napajanja analogno-digitalnog pretvornika iznosi 5 V i da je za referentni naponski izvor izabran napon napajanja analogno-digitalnog pretvornika AVCC ( $V_{REF} = 5$  V). Neka napon analognog senzora spojenog na pin ADC2 iznosi 2,14 V ( $U_{ADC2} = 2,14$  V). Prema relaciji (7.1), cjelobrojni rezultat analogno-digitalne pretvorbe na pinu PA2 iznosi:

$$ADC_2 = \frac{U_{ADC2} \cdot 1024}{5} = 438. \quad (7.2)$$

Pogreška kod proračuna digitalne riječi  $ADC_2$  može biti  $\pm 1$ . Analogni senzor mjeri neku fizikalnu veličinu i na svom izlazu daje analognu vrijednost napona koji mjeri pin  $ADC_i$ . Napon na pinu  $ADC_i$  na temelju rezultata analogno-digitalne pretvorbe na pinu  $ADC_i$  možemo dobiti pomoću relacije 7.3:

$$U_{ADC_i} = \frac{ADC_i \cdot V_{REF}}{1024}. \quad (7.3)$$

Pretpostavimo da napon napajanja analogno-digitalnog pretvornika iznosi 5 V i da je za referentni naponski izvor izabran napon napajanja analogno-digitalnog pretvornika AVCC ( $V_{REF} = 5$  V). Neka cjelobrojni rezultat analogno-digitalne pretvorbe na pinu ADC2 iznosi 765 ( $ADC_2 = 765$ ). Prema relaciji (7.3), napon analognog senzora spojenog na pin ADC2 iznosi:

$$U_{ADC2} = \frac{ADC_2 \cdot 5 \text{ V}}{1024} = 3.735 \text{ V}. \quad (7.4)$$

Kada znamo napon  $U_{ADC_i}$ , vrijednost fizikalne veličine možemo izračunati na temelju tehničkih specifikacija analognog senzora. Frekvencija rada analogno-digitalne pretvorbe odabire se u registru `ADCSRA` prema tablici 24-5 u literaturi [1].

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) 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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se `Ivica Ivic`. Datoteku `ADC.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atsln` u datoteci `\\ADC\vjezbe`. U otvorenom projektu nalaze se sve 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.

Neposredno prije rada s analogno-digitalnim pretvornikom potrebno je konfigurirati analogno-digitalnu pretvorbu u zaglavlju `adc.h` koje se nalazi u datoteci `ADC`. Otvorite zaglavlje `adc.h`. U programskom kodu 7.1 prikazan je odabir referentnog iznosa napona analogno-digitalne pretvorbe. Definirane su tri maske koje se dodjeljuju konstanti `ADC_REFERENCE`. Prema programskom kodu 7.1, za referentni iznos napona odabran je napon napajanja analogno-digitalnog pretvornika AVCC.

Programski kod 7.1: Odabir referentnog iznosa napona u zaglavlju `adc.h`

```
// definirane konstante za odabir referentnog iznosa napona
#define ADC_REFERENCE_AREF ((0 << REFS1) | (0 << REFS0)) // AREF pin
#define ADC_REFERENCE_AVCC ((0 << REFS1) | (1 << REFS0)) // AVCC pin
#define ADC_REFERENCE_RSVD ((1 << REFS1) | (0 << REFS0)) // Rezervirana
#define ADC_REFERENCE_256V ((1 << REFS1) | (1 << REFS0)) // REF je 2.56 V

// odabir referentnog iznosa napona od prethodna tri moguća
#define ADC_REFERENCE ADC_REFERENCE_AVCC
```

Ukoliko želite promijeniti referentni iznos napona na unutarnjih 2,56 V, konstanti `ADC_REFERENCE` dodijelite konstantu `ADC_REFERENCE_256V`.

U programskom kodu 7.2 prikazan je odabir frekvencije rada analogno-digitalne pretvorbe. Definirano je sedam maski koje predstavljaju djelitelje frekvencije mikroupravljača. Maske se dodjeljuju konstanti `ADC_PRESCALE`. Što je veći djelitelj frekvencije, frekvencija je rada analogno-digitalne pretvorbe manja, a rezultat je analogno-digitalne pretvorbe pouzdaniji. Prema programskom kodu 7.2, za djelitelj frekvencije odabran je 128.

Programski kod 7.2: Odabir frekvencije rada analogno-digitalne pretvorbe u zaglavlju `adc.h`

```
// definirani djelitelji frekvencije
#define ADC_PRESCALE_DIV2      ((0 << ADPS2) | (0 << ADPS1) | (1 << ADPS0))
#define ADC_PRESCALE_DIV4      ((0 << ADPS2) | (0 << ADPS1) | (0 << ADPS0))
#define ADC_PRESCALE_DIV8      ((0 << ADPS2) | (1 << ADPS1) | (1 << ADPS0))
#define ADC_PRESCALE_DIV16     ((1 << ADPS2) | (0 << ADPS1) | (0 << ADPS0))
#define ADC_PRESCALE_DIV32     ((1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0))
#define ADC_PRESCALE_DIV64     ((1 << ADPS2) | (0 << ADPS1) | (0 << ADPS0))
#define ADC_PRESCALE_DIV128    ((1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0))
// odabir djelitelja frkevencije
#define ADC_PRESCALE           ADC_PRESCALE_DIV128
```

Ukoliko želite povećati frekvenciju rada analogno-digitalne pretvorbe za dva puta, konstanti `ADC_PRESCALE` dodijelite konstantu `ADC_PRESCALE_DIV64`.



### Vježba 7.1.1

Napravite program koji će na početku prvog retka LCD displeja ispisati rezultat analogno-digitalne pretvorbe na pinu ADC5 (PF5), a na početku drugog retka napon koji se trenutno mjeri na pinu ADC5. Analogno-digitalnu pretvorbu provodite jednom u sekundi. Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra jest 10 kΩ.

U projektnom stablu otvorite datoteku `vjezba711.cpp`. Omogućite samo prevođenje datoteke `vjezba711.cpp`. Početni sadržaj datoteke `vjezba711.cpp` prikazan je programskim kodom 7.3.

Programski kod 7.3: Početni sadržaj datoteke `vjezba711.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "ADC/adc.h"

void inicijalizacija() {

    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_5;
    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon
```

```

while(1) {

    ADC_5 = adc_read(ADC5); // ADC na kanalu ADC5
    U_ADC5 = ADC_5 * V_REF / 1024; // pretvorba u napon

    lcd_clrscr();
    lcd_home();
    lcd_print("ADC5 = %d", ADC_5);
    lcd_print("\nUADC5 = %.2fV", U_ADC5);
    _delay_ms(1000);
}
return 0;
}

```

Funkcije koje se koriste za analogno-digitalnu pretvorbu definirane su u zaglavlju `adc.h`. Naredba kojom uključujemo zaglavlje `adc.h` u datoteku koja se prevodi jest `#include "ADC/adc.h"`.

U programskom kodu 7.3 nalazi se niz naredbi koje služe za rad s analogno-digitalnim pretvornikom:

- `adc_init` - funkcija koja inicijalizira analogno-digitalnu pretvorbu. U ovoj se funkciji konfiguriraju referentni napon analogno-digitalne pretvorbe i frekvencija rada analogno-digitalne pretvorbe na temelju konstanti `ADC_REFERENCE` i `ADC_PRESCALE` koje smo odabrali u zaglavlju `adc.h` te se omogućuje analogno-digitalna pretvorba.
- `adc_read(uint8_t ch)` - funkcija koja kao argument prima kanal analogno-digitalne pretvorbe `ch` na kojem želite napraviti analogno-digitalnu pretvorbu. Dostupni kanali istoimeni su alternativnim nazivima pinova mikroupravljača ATmega32U4: `ADC0`, `ADC1`, `ADC4`, `ADC5`, `ADC6`, `ADC7`, `ADC8`, `ADC9`, `ADC10`, `ADC11`, `ADC12`, `ADC13`. Povratna je vrijednost funkcije `adc_read` vrijednost analogno-digitalne pretvorbe na pinu `ADCi`. Povratna je vrijednost širine 10 bitova. Na primjer, ako želimo pročitati analognu vrijednost s pina `ADC2`, pozvat ćemo funkciju `adc_read(ADC2)`.

Za referentni iznos napona odabran je napon napajanja analogno-digitalnog pretvornika `AVCC`. Taj napon spojen je i na potenciometar nazivne vrijednosti 10 k $\Omega$ . U programskom kodu 7.3 deklarirane su sljedeće varijable:

- `uint16_t ADC_5` - varijabla koja se koristi za spremanje rezultata analogno-digitalne pretvorbe na pinu `ADC5`,
- `float U_ADC5` - napon na pinu `ADC5`,
- `const float V_REF` - referentni iznos napona `AVCC = 5 V`.

Sve funkcije inicijalizacije mikroupravljača uvijek ćemo pozivati u funkciji `inicijalizacija` u kojoj se nalazi i funkcija za inicijalizaciju analogno-digitalne pretvorbe `adc_init`.

Funkcija `adc_read(ADC5)` vraća rezultat analogno-digitalne pretvorbe na pinu `ADC5`. Vrijednost napona na pinu `ADC5` dobivena je pomoću relacije (7.3) i ispisana na LCD displej na preciznost od dva decimalna mjesta. U `while` petlji postavljeno je kašnjenje od jedne sekunde, što svake sekunde osigurava novo mjerenje na pinu `ADC5`.

Prevedite datoteku `vjezba711.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Što je sve potrebno napraviti ako želimo da referentni iznos napona bude unutarnjih 2,56 V? Može li se uopće prema shemi 7.1 koristiti unutarnji referentni napon iznosa 2,56 V?

Zatvorite datoteku `vjezba711.cpp` i onemogućite prevođenje ove datoteke.



## Vježba 7.1.2

Napravite program koji će na početku prvog retka LCD displeja ispisati srednju vrijednost 10 uzoraka analogno-digitalne pretvorbe na pinu ADC5 uzetih u vremenu od jedne sekunde. Na početku drugog retka LCD displeja ispišite srednju vrijednost napona na pinu ADC5. Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra jest 10 k $\Omega$ .

U projektnom stablu otvorite datoteku `vjezba712.cpp`. Omogućite samo prevođenje datoteke `vjezba712.cpp`. Početni sadržaj datoteke `vjezba712.cpp` prikazan je programskim kodom 7.4.

Programski kod 7.4: Početni sadržaj datoteke `vjezba712.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
    // inicijalizacija AD pretvorbe
}

int main(void) {
    inicijalizacija();

    uint16_t ADC_5;
    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon

    while(1) {
        ADC_5 = 0;

        for(uint8_t i = 0; i < 10; i++) {
            // zbroj 10 mjerenja na pinu ADC5 u 1000 ms
            _delay_ms(100);
        }

        ADC_5 = ADC_5 / 10;
        // srednja vrijednost napona na pinu ADC5

        lcd_clrscr();
        lcd_home();
        lcd_print("ADC5 = %d", ADC_5);
        lcd_print("\nUADC5 = %.2fV", U_ADC5);
    }
    return 0;
}
```

Usrednjavanje mjernih uzoraka najjednostavniji je način filtriranja mogućih poremećaja koji se mogu pojaviti na izlazu mjernih senzora. U ovoj vježbi napraviti ćemo usrednjavanje 10 mjernih uzoraka uzetih u jednoj sekundi.

U programskom kodu 7.4 nedostaje inicijalizacija analogno-digitalne pretvorbe u funkciji `inicijalizacija()`. Inicijalizirajte analogno-digitalnu pretvorbu. Uključite zaglavlja koja nedostaju.

U programskom kodu 7.4 nalazi se `for` petlja koja se izvodi 10 puta. U bloku naredbi `for` petlje potrebno je sumirati 10 mjerenja na pinu ADC5 u jednoj sekundi. Prema tome, u jednoj iteraciji `for` petlje potrebno je ostvariti kašnjenje od 100 ms. Početna vrijednost varijable `ADC_5` u koju će se sumirati 10 mjerenja jest 0.

U `for` petlju upišite naredbu `ADC_5 = ADC_5 + adc_read(ADC5)`; te naredbu koja će u `for` petlji ostvariti kašnjenje od 100 ms. Nakon što `for` petlja izvrši 10 iteracija, u varijabli `ADC_5` nalazi se suma od 10 mjerenja u digitalnom zapisu. Srednju vrijednost 10 mjerenja dobit ćemo tako da vrijednost varijable `ADC_5` podijelimo s 10. Prema relaciji (7.3) izračunajte srednju vrijednost napona na pinu ADC5 ispod naredbe `ADC_5 = ADC_5 / 10`; . Vrijednosti varijable `ADC_5` i srednje vrijednosti napona ispisuju se na LCD displej.

Prevedite datoteku `vjezba712.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite broj mjerenih uzoraka u jednoj sekundi te ponovno prevedite datoteku `vjezba712.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba712.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 7.1.3

Napravite program koji će na početku prvog retka LCD displeja ispisati vrijednost dobivenu pomoću digitalnog filtra prvog reda:

$$y[k] = 0.2y[k - 1] + 0.8ADC_5[k]$$

gdje je:

- $y[k]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k$ ,
- $y[k - 1]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k - 1$  (filtrirana vrijednost analogno-digitalne pretvorbe iz prethodnog mjerenja),
- $ADC_5[k]$  - rezultat analogno-digitalne pretvorbe na pinu ADC5 u koraku  $k$ .

Početna vrijednost digitalnog filtra prvog reda jest  $y[-1] = 0$ . Vrijeme između uzimanja uzoraka  $k$  i  $k - 1$  neka bude 100 ms. Na početku drugog retka ispišite napon dobiven na temelju filtrirane vrijednosti analogno-digitalne pretvorbe na pinu ADC5. Dodatno, napravite signalizaciju LED diodama na sljedeći način:

- ako je napon manji od 1,25 V, neka je uključena samo plava LED dioda,
- ako je napon veći i jednak 1,25 V i manji od 2,5 V, neka su uključene plava i zelena LED dioda,
- ako je napon veći i jednak 2,5 V i manji od 3,75 V, neka su uključene plava, zelena i žuta LED dioda,
- ako je napon veći i jednak 3,75 V, neka su uključene sve LED diode.

Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra je 10 kΩ.

U projektnom stablu otvorite datoteku `vjezba713.cpp`. Omogućite samo prevođenje datoteke `vjezba713.cpp`. Početni sadržaj datoteke `vjezba713.cpp` prikazan je programskim kodom 7.5.

Programski kod 7.5: Početni sadržaj datoteke vj\_ezba713.cpp

```

#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "ADC/adc.h"

void inicijalizacija() {

    //pinovi PB7, PB6, PB5 i PB4 postavljeni kao izlazni

    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_5;
    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon
    float y[2] = {0, 0}; // {y[k-1], y[k]}
    uint8_t k = 1;

    while(1) {

        ADC_5 = adc_read(ADC5);

        y[k] = 0.2 * y[k - 1] + 0.8 * ADC_5;

        U_ADC5 = y[k] * V_REF / 1024.0;

        lcd_clrscr();
        lcd_home();
        lcd_print("ADC5 = %d", ADC_5);
        lcd_print("\nUADC5 = %.2fV", U_ADC5);

        _delay_ms(100);

        y[k-1] = y[k];

        if(U_ADC5 < 1.25) {
            set_port(PORTB, PB4, 1);
            set_port(PORTB, PB5, 0);
            set_port(PORTB, PB6, 0);
            set_port(PORTB, PB7, 0);
        }

        if(U_ADC5 >= 1.25 && U_ADC5 < 2.5) {
            set_port(PORTB, PB4, 1);
            set_port(PORTB, PB5, 1);
            set_port(PORTB, PB6, 0);
            set_port(PORTB, PB7, 0);
        }

        // nastavite za više napone
    }
    return 0;
}

```

Digitalni filtar prvog reda može filtrirati poremećaje koji se pojavljuju na izlazu mjernog senzora spojenog na analogni ulaz mikroupravljača. U ovoj vježbi koriste se LED diode za

signalizaciju. U funkciji `inicijalizacija()` u programskom kodu 7.5 konfigurirajte izlazne pinove PB4, PB5, PB6 i PB7.

U `while` petlji najprije se izvrši analogno-digitalna pretvorba na pinu ADC5. Rezultat pretvorbe sprema se u varijablu `ADC_5`. Nakon toga potrebno je izračunati filtriranu vrijednost analogno-digitalne pretvorbe pomoću relacije  $y[k] = 0.2y[k-1] + 0.8ADC5[k]$ . Napon koji ćemo prezentirati na LCD displeju dobiven je temeljem filtrirane vrijednosti analogno-digitalne pretvorbe  $y[k]$ . Napon se izračunava pomoću relacije (7.3) i sprema se u varijablu `U_ADC5`. Na LCD displej ispisuje se filtrirana vrijednost analogno-digitalne pretvorbe  $y[k]$  i napon `U_ADC5`.

Kašnjenje između dva uzorka  $y[k]$  i  $y[k-1]$  iznosi 100 ms. Nakon ovog kašnjenja korak  $k$  povećao se za jedan te vrijednost  $y[k]$  postaje  $y[k-1]$ . U programskom kodu 7.5 to znači da element polja `y[k-1]` poprima prethodnu vrijednost elementa polja `y[k]`.

U nastavku programskog koda 7.5 izvedena je signalizacija pomoću LED dioda. Završite niz naredbi za signalizaciju LED diodama za slučajeve kada je napon veći i jednak 2,5 V.

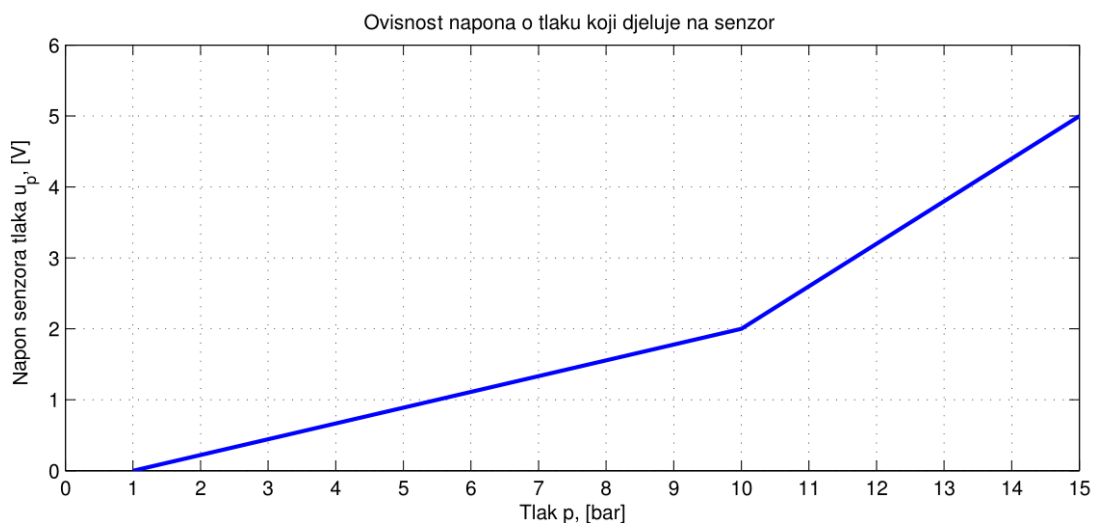
Prevedite datoteku `vjezba713.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba713.cpp` i onemogućite prevođenje ove datoteke.



#### Vježba 7.1.4

Potenciometrom spojenim na pin ADC5 simulirajte izlazni napon senzora tlaka čija se ovisnost napona o tlaku može pročitati s grafikona na slici 7.2.



Slika 7.2: Ovisnost napona o tlaku koji djeluje na senzor

Napraviti program koji će na početku prvog retka LCD displeja ispisati izlazni napon senzora tlaka, a u drugom retku tlak koji se trenutno mjeri senzorom tlaka. Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra jest 10 kΩ.

U projektnom stablu otvorite datoteku `vjezba714.cpp`. Omogućite samo prevođenje datoteke `vjezba714.cpp`. Početni sadržaj datoteke `vjezba714.cpp` prikazan je programskim kodom 7.6.

Programski kod 7.6: Početni sadržaj datoteke vj\_ezba714.cpp

```

#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "ADC/adc.h"

void inicijalizacija() {

    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_5; // vrijednost AD pretvorbe na pinu ADC5
    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon
    float u, p; // napon i tlak

    while(1) {

        //odredite ADC_5 i U_ADC5
        u = U_ADC5;

        if (u <= 2.0) {
            p = 9 * u / 2 + 1;
        }
        else {
            p = 5 * u / 3 + 20.0 / 3;
        }
        //ispišite napon i tlak na 3 decimalna mjesta
    }
    return 0;
}

```

Ovisnost napona o tlaku koji djeluje na senzor prikazana je na slici 7.2. Pretpostavimo da je ova ovisnost definirana tehničkim specifikacijama senzora tlaka. Kao što se može vidjeti na slici 7.2, ovisnost napona o tlaku aproksimacija je pravcima. Mogu se uočiti dva slučaja:

- 1. slučaj: - napon se u ovisnosti o tlaku mijenja po zakonitosti pravca kroz točke A(1 bar, 0 V) i B(10 bar, 2 V),
- 2. slučaj: - napon se u ovisnosti o tlaku mijenja po zakonitosti pravca kroz točke B(10 bar, 2 V) i C(15 bar, 5 V).

Kako bismo proračunali ovisnost napona o tlaku, koristit ćemo jednadžbu pravca kroz dvije točke. Za 1. slučaj vrijedi jednadžba pravca kroz točke A i B:

$$u - 0 = \frac{2 - 0}{10 - 1}(p - 1). \quad (7.5)$$

Nakon sređivanja relacija (7.6) dobije se:

$$u = \frac{2}{9}p - \frac{2}{9}. \quad (7.6)$$

Prilikom mjerenja napona na mjernom senzoru tlaka nepoznanica nam je tlak. Tlak se za 1.



slučaj može dobiti sređivanjem relacije (7.6):

$$p = \frac{9}{2}u + 1. \quad (7.7)$$

Za 2. slučaj vrijedi jednadžba pravca kroz točke B i C:

$$u - 2 = \frac{5 - 2}{15 - 10}(p - 10). \quad (7.8)$$

Nakon sređivanja relacija (7.8) dobije se:

$$u = \frac{3}{5}p - 4. \quad (7.9)$$

Prilikom mjerenja napona na mjernom senzoru tlaka, nepoznanica nam je tlak. Tlak se za 2. slučaj može dobiti sređivanjem relacije (7.9):

$$p = \frac{5}{3}u + \frac{20}{3}. \quad (7.10)$$

Izveli smo jednadžbe pravca koje nam daju ovisnost tlaka o naponu jer je tlak ono što nas zanima. Napon je samo posredna veličina preko koje analogno-digitalnom pretvorbom izračunavamo tlak. U programskom kodu 7.6 odredite rezultat analogno-digitalne pretvorbe na pinu ADC5 (varijabla `ADC_5`) pozivom funkcije `adc_read`. Odredite napon na pinu ADC5 (varijabla `U_ADC5`) prema relaciji 7.1.

U uvjetnom `if` bloku provjeravamo je li napon na ADC5 pinu manji i jednak 2 V. Ako je napon manji i jednak 2 V, tada tlak računamo prema relaciji (7.7). Ako napon nije manji i jednak 2 V, tada tlak računamo prema relaciji (7.10).

Ispišite na LCD displej u prvom retku napon, a u drugom retku tlak na tri decimalna mjesta. Na kraju postavite proizvoljno kašnjenje u `while` petlji kako bi osigurali čitljivost podataka na LCD displeju.

Prevedite datoteku `vjezba714.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba714.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 7.1.5

Napravite program koji će na početku prvog retka LCD displeja ispisati temperaturu u °C koju mjeri NTC otpornik. Ovisnost otpora NTC otpornika o temperaturi u njegovoj okolini prikazana je relacijom (7.11):

$$R_T = R_N e^{B\left(\frac{1}{T} - \frac{1}{T_N}\right)} \quad (7.11)$$

gdje su:

- $R_T$  - otpor NTC otpornika na temperaturi  $T$ , [ $\Omega$ ],
- $R_N = 10 \text{ k}\Omega$  - otpor NTC otpornika na nazivnoj temperaturi  $T_N$ , [ $\Omega$ ],
- $T$  - temperatura okoline NTC otpornika, [K],
- $T_N = 25 \text{ }^\circ\text{C} = 298,15 \text{ K}$  - nazivna temperatura okoline NTC otpornika, [K],
- $B = 3435 \text{ K}$  - konstanta ovisna o materijalu NTC otpornika, [K].

U programu napravite dio koda za signalizaciju LED diodama na sljedeći način:

- ako je temperatura manja od 25 °C, neka je uključena samo plava LED dioda,
- ako je temperatura veća i jednaka 25 °C i manja od 27,5 °C, neka su uključene plava i zelena LED dioda,
- ako je temperatura veća i jednaka 27,5 °C i manja od 30 °C, neka su uključene plava, zelena i žuta LED dioda,
- ako je temperatura veća i jednaka 30 °C, neka su uključene sve LED diode.

NTC otpornik spojen je u naponskom djelilu s otporom  $R = 4,7 \text{ k}\Omega$  prema shemi prikazanoj na slici 7.1. Pad napona na NTC otporniku mjeri se pomoću pina ADC4.

U projektnom stablu otvorite datoteku `vjezba715.cpp`. Omogućite samo prevođenje datoteke `vjezba715.cpp`. Početni sadržaj datoteke `vjezba715.cpp` prikazan je programskim kodom 7.7.

Programski kod 7.7: Početni sadržaj datoteke `vjezba715.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "ADC/adc.h"

void inicijalizacija() {

    //pinovi PB7, PB6, PB5 i PB4 postavljeni kao izlazni

    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_4; // vrijednost AD pretvorbe na pinu ADC4
    float T; // temperatura

    while(1) {

        ADC_4 = adc_read(ADC4);

        // dodajte relaciju za temperaturu NTC otpornika

        lcd_clrscr();
        lcd_home();
        lcd_print("T = %0.2f°C\n", T, 223);

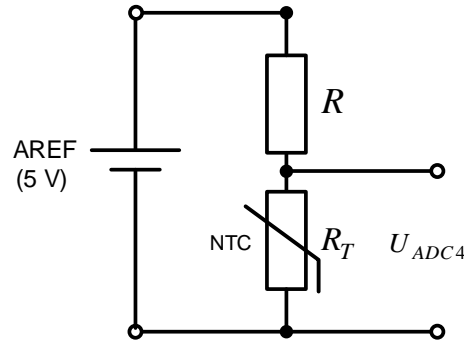
        // signalizacija po istom principu kao u datoteci vjezba713.c

        _delay_ms(1000);
    }
    return 0;
}
```

Svojstvo NTC (engl. *Negative Temperature Coefficient*) otpornika jest da se porastom temperature u okolini NTC otpornika njegov otpor smanjuje, a smanjenjem temperature njegov

se otpor povećava. NTC otpornici često se koriste u praksi za mjerenje temperature jer im je cijena niska. Ovisnost otpora NTC otpornika o temperaturi u njegovoj okolini eksponencijalna je. Temperatura se u relaciji (7.11) nalazi u nazivniku eksponenta. Cilj nam je temperaturu u relaciji (7.11) izraziti preko otpora NTC otpornika.

NTC otpornik spojen je u djelilu napona s otpornikom  $R = 4,7 \text{ k}\Omega$  prema slici 7.3.



Slika 7.3: NTC otpornik spojen u djelilu napona s otpornikom  $R = 4,7 \text{ k}\Omega$

Napon  $U_{ADC4}$  na pinu ADC4 prema slici 7.3 jednak je:

$$U_{ADC4} = \frac{R_T}{R_T + R} 5 \quad (7.12)$$

Iz relacije (7.11) potrebno je izraziti temperaturu  $T$  preko otpora NTC otpornika. Ako relaciju (7.11) logaritmiramo prirodnim logaritmom  $\ln$ , dobit ćemo sljedeću relaciju:

$$\begin{aligned} R_T = R_N e^{B\left(\frac{1}{T} - \frac{1}{T_N}\right)} / : R_N &\Rightarrow \frac{R_T}{R_N} = e^{B\left(\frac{1}{T} - \frac{1}{T_N}\right)} / \ln() \Rightarrow \\ \ln\left(\frac{R_T}{R_N}\right) = B\left(\frac{1}{T} - \frac{1}{T_N}\right) &\Rightarrow \frac{B}{T} = \ln\left(\frac{R_T}{R_N}\right) + \frac{B}{T_N}. \end{aligned} \quad (7.13)$$

Napravimo sada recipročnu vrijednost izvedene relacije (7.13) i pomnožimo ju s B:

$$T = \frac{B}{\ln\left(\frac{R_T}{R_N}\right) + \frac{B}{T_N}}. \quad (7.14)$$

Relacija (7.14) predstavlja funkcijsku ovisnost temperature o otporu NTC otpornika koji je jedina nepoznanica u ovoj relaciji. Izrazimo sada vrijednost otpora NTC otpornika iz relacije (7.12):

$$\begin{aligned} U_{ADC4} = \frac{R_T}{R_T + R} 5 / \cdot (R_T + R) &\Rightarrow (R_T + R) U_{ADC4} = R_T 5 \\ R_T = \frac{U_{ADC4}}{5 - U_{ADC4}} R \end{aligned} \quad (7.15)$$

Napon  $U_{ADC4}$  možemo izračunati pomoću analogno-digitalne pretvorbe prema relaciji (7.3). Vrijednost napona od 5 V u relaciji (7.15) prema relaciji (7.3) zamijenit ćemo digitalnom vrijednošću 1024. Prema tome, relaciju (7.15) možemo zapisati u sljedećem obliku:

$$R_T = \frac{ADC4}{1024 - ADC4} R. \quad (7.16)$$

Otpor NTC otpornika iz relacije (7.16) uvrstit ćemo u relaciju (7.14):

$$T = \frac{B}{\ln\left(\frac{ADC_4}{1024-ADC_4} \frac{R}{R_N}\right) + \frac{B}{T_N}}. \quad (7.17)$$

Pravilima logaritmiranja<sup>1</sup> možemo pojednostavniti relaciju (7.17):

$$T = \frac{B}{\ln\left(\frac{ADC_4}{1024-ADC_4}\right) + \ln\left(\frac{R}{R_N}\right) + \frac{B}{T_N}}. \quad (7.18)$$

Pojednostavljena relacija (7.19) manje je zahtjevna za izračun u mikroupravljaču jer smo dio proračuna napravili ručno. U praksi se preporučuje pojednostavljenje matematičkih relacija kako bi mikroupravljač brže proračunao te relacije. U relaciji (7.19) potrebno je uvrstiti poznate parametre, a to su otpor naponskog djelila  $R = 4,7 \text{ k}\Omega$ , nazivni otpor NTC otpornika  $R_N = 10 \text{ k}\Omega$ , nazivna temperatura  $T_N = 298,15 \text{ K}$  i konstanta  $B = 3435 \text{ K}$ :

$$T = \frac{3435}{\ln\left(\frac{ADC_4}{1024-ADC_4}\right) + 10,861}. \quad (7.19)$$

Dobivena temperatura u relaciji (7.19) izražena je u K. Temperatura u °C prikazana je relacijom (7.20).

$$T = \frac{3435}{\ln\left(\frac{ADC_4}{1024-ADC_4}\right) + 10,861} - 273,15 \quad (7.20)$$

Relacija (7.20) konačna je i potrebno ju je zapisati u sintaksi programskog jezika C/C++ u programski kod 7.7. Funkcija koja u programskom jeziku C računa prirodni logaritam  $\ln()$  jest `log(double)`. U programskom je kodu napisana naredba za analogno-digitalnu pretvorbu na pinu ADC4, a rezultat pretvorbe sprema se u varijablu ADC\_4.

Često se u praktičnoj primjeni senzora najprije mora provesti proračun koji će nam omogućiti prezentaciju fizikalne veličine koju senzor mjeri. Prethodni proračun najbolji je primjer toga.

Za signalizaciju LED diodama potrebno je konfigurirati izlaze PB4, PB5, PB6 i PB7. Napravite niz naredbi koje će LED diodama signalizirati vrijednost temperature prema uputama iz vježbe. Signalizacija je slična onoj u vježbi 7.1.3. Nakon signalizacije, ostvarili smo kašnjenje programa od jedne sekunde.

Prevedite datoteku `vjezba715.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba715.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

<sup>1</sup>Logaritam umnoška dvaju operanada jednak je sumi logaritama tih dvaju operanada,  $\ln(ab) = \ln a + \ln b$ .

### 7.1.1 Zadaci - analogno-digitalna pretvorba

#### Zadatak 7.1.1

Napravite program koji će na početku prvog retka LCD displeja ispisati rezultat analogno-digitalne pretvorbe na pinu ADC5, a na početku drugog retka otpor potencijometra spojenog na pin ADC5. Nazivna vrijednost otpora potencijometra jest 10 kΩ. Analogno-digitalnu pretvorbu provodite jednom u sekundi. Na pinu ADC5 spojen je potencijometar prema shemi prikazanoj na slici 7.1.

---

#### Zadatak 7.1.2

Napravite program koji će na početku prvog retka LCD displeja ispisati srednju vrijednost šest uzoraka analogno-digitalne pretvorbe na pinu ADC5 uzetih u vremenu od 900 ms. Na početku drugog retka LCD displeja ispišite srednju vrijednost napona na pinu ADC5. Na pinu ADC5 spojen je potencijometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potencijometra jest 10 kΩ.

---

#### Zadatak 7.1.3

Napravite program koji će na početku prvog retka LCD displeja ispisati vrijednost dobivenu pomoću digitalnog filtra drugog reda:

$$y[k] = 0.2y[k-1] + 0.1y[k-2] + 0.7ADC5[k]$$

gdje je:

- $y[k]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k$ ,
- $y[k-1]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k-1$ ,
- $y[k-2]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k-2$ ,
- $ADC5[k]$  - rezultat analogno-digitalne pretvorbe na pinu ADC5 u koraku  $k$ .

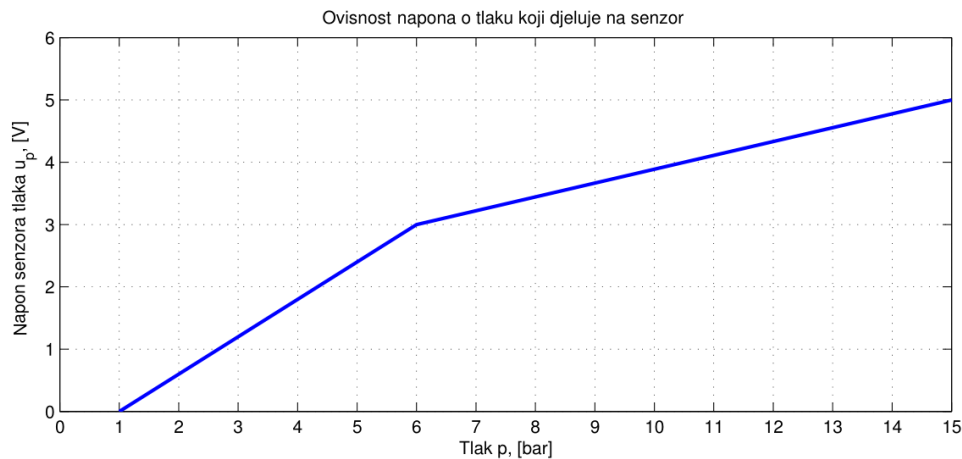
Početne vrijednosti digitalnog filtra drugog reda jesu  $y[-1] = 0$  i  $y[-2] = 0$ . Vrijeme između uzimanja uzoraka  $k$  i  $k-1$  neka bude 200 ms. Na početku drugog retka ispišite napon dobiven na temelju filtrirane vrijednosti analogno-digitalne pretvorbe na pinu ADC5. Dodatno, napravite signalizaciju LED diodama na sljedeći način:

- ako je napon manji od 1,25 V, neka je uključena samo crvena LED dioda,
- ako je napon veći i jednak 1,25 V i manji od 2,5 V, neka su uključene crvena i žuta LED dioda,
- ako je napon veći i jednak 2,5 V i manji od 3,75 V, neka su uključene crvena, žuta i zelena LED dioda,
- ako je napon veći i jednak 3,75 V, neka su uključene sve LED diode.

Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra jest  $10\text{ k}\Omega$ .

#### Zadatak 7.1.4

Potenciometrom spojenim na pin ADC5 simulirajte izlazni napon senzora tlaka čija se ovisnost napona o tlaku može pročitati s grafikona na slici 7.4.



Slika 7.4: Ovisnost napona o tlaku koji djeluje na senzor

Napravite program koji će na početku prvog retka LCD displeja ispisati izlazni napon senzora tlaka, a u drugom retku tlak koji se trenutno mjeri senzorom tlaka. Na pinu ADC5 spojen je potenciometar prema shemi prikazanoj na slici 7.1. Nazivna vrijednost otpora potenciometra jest  $10\text{ k}\Omega$ .

#### Zadatak 7.1.5

Napravite program koji će na početku prvog retka LCD displeja ispisati temperaturu u K koju mjeri NTC otpornik, a na početku drugog retka otpor NTC otpornika. Ovisnost otpora NTC otpornika o temperaturi u njegovoj okolini prikazana je relacijom (7.21):

$$R_T = R_N e^{B\left(\frac{1}{T} - \frac{1}{T_N}\right)} \quad (7.21)$$

gdje su:

- $R_T$  - otpor NTC otpornika na temperaturi  $T$ ,  $[\Omega]$ ,
- $R_N = 1\text{ k}\Omega$  - otpor NTC otpornika na nazivnoj temperaturi  $T_N$ ,  $[\Omega]$ ,
- $T$  - temperatura okoline NTC otpornika,  $[\text{K}]$ ,
- $T_N = 25\text{ }^\circ\text{C} = 298,15\text{ K}$  - nazivna temperatura okoline NTC otpornika,  $[\text{K}]$ ,
- $B = 3435\text{ K}$  - konstanta ovisna o materijalu NTC otpornika,  $[\text{K}]$ .

NTC otpornik spojen je u naponskom djelilu s otporom  $R = 10\text{ k}\Omega$  prema shemi prikazanoj na slici 7.1. Pad napona na NTC otporniku mjeri se pomoću pina ADC4.

## Poglavlje 8

# Tajmeri i brojači

Tajmer (engl. *timer*) važan je dio svakog mikroupravljača i većina mikroupravljača ima jedan ili više tajmera s rezolucijom od 8 i/ili 16 bitova [4]. Primjena tajmera široka je, a najčešće se koriste za sinkronizaciju događaja u mikroupravljaču, generiranje periodičnih valnih oblika, uzimanje mjernih uzoraka u točnim vremenskim trenucima i u digitalnoj izvedbi PID regulatora. Tajmer je u osnovi brojač (engl. *counter*) koji povećava ili smanjuje svoj iznos za 1 na svaki  $k$ -ti<sup>1</sup> impuls radnog takta. Tajmer i brojač na hardverskoj su razini isti sklop koji se može konfigurirati ili kao tajmer ili kao brojač. Ako je izvor impulsa  $k$ -ti impuls radnog takta, onda je ovaj sklop konfiguriran kao tajmer, a ako je izvor impulsa vanjski uređaj, tada je ovaj sklop konfiguriran kao brojač.

Rezolucija tajmera i brojača jest broj bitova u registru koji se koriste za brojanje impulsa s izvora 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 8, tada se vrijednosti njihova registra kreću u rasponu od  $[0, 255]$ . Na svaki impuls s izvora impulsa vrijednost se u registru povećava ili smanjuje za 1. Ako je vrijednost registra tajmera 255 i njegova se vrijednost povećava sa svakim impulsom, tada će nakon sljedećeg impulsa njegova vrijednost biti 0. U prijelazu iz 255 u 0 dogodio se preljev (engl. *overflow*) jer broj 256 ne stane u registar širine 8 bitova. Tajmeri u većini mikroupravljača generiraju prekid u trenutku kada se dogodi preljev. Prekid (engl. *interrupt*) je mehanizam mikroupravljača koji omogućava da se na neke događaje odgovori u trenutku kada se oni dogode iako se u glavnom programu izvodi programski kod. Generiranje prekida u trenutku preljeva omogućuje da se dio programskog koda izvodi u jednakim vremenskim razmacima.

Poziv prekidne rutine u trenutku kada se dogodio prekid ilustriran je slikom 8.1. U glavnom programu u beskonačnoj `while` petlji izvodi se dio programskog koda koji na LCD displeju prikazuje tlak i napon senzora tlaka spojenog na pin PA5. Prekid se može dogoditi u bilo kojem trenutku izvođenja beskonačne `while` petlje. Čim se dogodi prekid, poziva se prekidna rutina u kojoj se nalazi dio programskog koda. Nakon što se izvede programski kod prekidne rutine, mikroupravljač nastavlja izvoditi programski kod na mjestu gdje je prekidna rutina pozvana (slika 8.1). Mikroupravljači imaju više izvora prekida. Ukoliko se istovremeno dogodi više prekida s različitih izvora, prekidi se izvode prema prioritetu.

Tajmeri u mikroupravljaču mogu mjeriti vrijeme jer broje impulse dobivene na temelju radnog takta čija je frekvencija poznata. Radni takt moguće je sklopovski podijeliti brojem  $k$ , ( $k = 1, 2, 8, 32, 64, 128, 256, 1024$ ). Ukoliko je radni takt sklopovski podijeljen brojem  $k$ , tada se vrijednost registra u tajmeru povećava ili smanjuje za 1 na svaki  $k$ -ti impuls radnog takta. Sklop za dijeljenje frekvencije radnog takta zove se djelitelj frekvencije radnog takta (engl. *prescaler*).

---

<sup>1</sup>Broj  $k$  ovisi o mikroupravljaču, a najčešće je  $k = 1, 2, 8, 32, 64, 128, 256, 1024$ .

```

inicijalizacija();

uint16_t ADC_4; // vrijednost AD pretvorbe na pinu ADC4
float T; // temperatura

while(1) {      Glavni program
                (beskonačna petlja)

    ADC_4 = adc_read(ADC4);
    T = 3450 / (Log(ADC_4/(1023.0 - ADC_4))+10.861) - 273.15;

    lcd_clrscr();
    lcd_home();
    lcd_print("T = %.2f%cC\n", T, 223);

    // signalizacija po istom principu kao u datoteci vjezba713.c
    if(T < 25.0) {
        Točka u kojoj se dogodio prekid
        set_port(PORTB,PB4,1);
        set_port(PORTB,PB5,0);
        set_port(PORTB,PB6,0);
        set_port(PORTB,PB7,0);
    }

    if(T >= 25.0 && T < 27.5) {

        set_port(PORTB,PB4,1);
        set_port(PORTB,PB5,1);
    }
}

ISR(TIMER0_OVF_vect){
    TCNT0 = 0;
    pozicija++;
}

```

Slika 8.1: Poziv prekidne rutine u trenutku kada se dogodio prekid

Mikroupravljač ATmega32U4 ima četiri tajmera i brojača od kojih ćemo tri dolje navedena koristiti u svrhu vježbi:

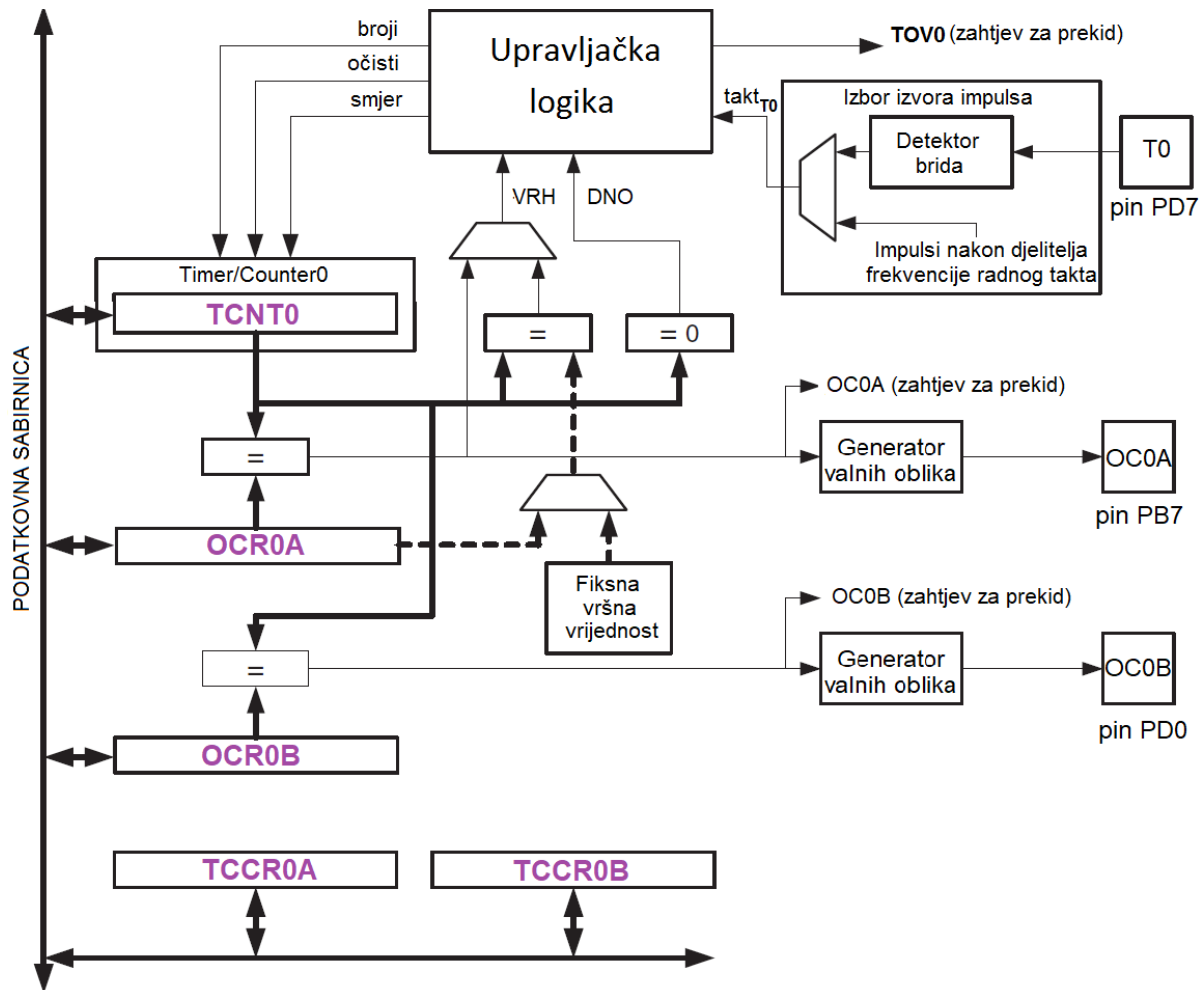
- *Timer/Counter0* - tajmer i brojač rezolucije 8 bitova s rasponom brojanja od [0, 255],
- *Timer/Counter1* - tajmer i brojač rezolucije 16 bitova s rasponom brojanja od [0, 65535],
- *Timer/Counter3* - tajmer i brojač rezolucije 16 bitova s rasponom brojanja od [0, 65535].

Rad tajmera i brojača objasniti ćemo na sklopu *Timer/Counter0*. U nastavku teksta tajmer i brojač s indeksom 0 zvat ćemo izvornim imenom *Timer/Counter0* iz literature [1]. Na slici 8.2 prikazana je blokovska shema za sklop *Timer/Counter0* rezolucije 8 bitova.

Sklop *Timer/Counter0* ima pet 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.
- **OCROA**, **OCROB** - (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*.





Slika 8.2: Blokovska shema za sklop *Timer/Counter0* rezolucije 8 bitova

Na blokovskoj shemi sklopa *Timer/Counter0* nalaze se sljedeći signali:

- **broji** - uvećaj ili smanji vrijednost registra **TCNT0** za 1,
- **očisti** - postavi vrijednost registra **TCNT0** na 0,
- **smjer** - uvećavaj ili smanjuj za 1,
- **takt<sub>T0</sub>** - 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,
- **OC0A, OC0B** - prekidi koji se generiraju kada je vrijednost registra **TCNT0** jednaka vrijednosti registara **OCR0A, OCR0B**.

Registri kojima se konfigurira rad sklopa *Timer/Counter0* prikazani su na slikama 8.3 i 8.4. Svaki bit u registrima **TCCROA** i **TCCROB** ima svoju funkciju i određuje rad sklopa *Timer/Counter0*.

Bit	7	6	5	4	3	2	1	0
Registar <b>TCCROA</b>	<b>COMOA1</b>	<b>COMOA0</b>	<b>COMOB1</b>	<b>COMOB0</b>	—	—	<b>WGM01</b>	<b>WGM00</b>

Slika 8.3: Registar **TCCROA**

Bit	7	6	5	4	3	2	1	0
Registar <b>TCCROB</b>	<b>FOCOA</b>	<b>FOCOB</b>	—	—	<b>WGM02</b>	<b>CS02</b>	<b>CS01</b>	<b>CS00</b>

Slika 8.4: Registar **TCCROB**

Na slici 8.2 prikazan je blok **Izbor izvora impulsa**. Izvor impulsa mogu biti ili impulsi nakon djelitelj frekvencije radnog takta ili impulsi koji se dovode na pin PD7 (T0). Ponovno možemo primijetiti da pin PD7 ima alternativnu funkciju. Izvor impulsa odabire se u registru **TCCROB** namještanjem bitova **CS02**, **CS01** i **CS00** prema tablici 8.1. Prvih šest redaka u tablici 8.1 konfiguriraju sklop *Timer/Counter0* kao tajmer, a zadnja dva retka konfiguriraju sklop *Timer/Counter0* kao brojač. Radni takt definiran je *Fuse* bitovima i konstantom **F\_CPU**.

Tablica 8.1: Izbor izvora impulsa sklopa *Timer/Counter0* [1]

<b>CS02</b>	<b>CS01</b>	<b>CS00</b>	<b>Izvor impulsa</b>
0	0	0	Nema impulsa, <i>Timer/Counter0</i> isključen
0	0	1	<b>F_CPU</b>
0	1	0	<b>F_CPU/8</b>
0	1	1	<b>F_CPU/64</b>
1	0	0	<b>F_CPU/256</b>
1	0	1	<b>F_CPU/1024</b>
1	1	0	Na padajući brid signala pina T0, uvećaj TCNT0 za 1
1	1	1	Na rastući brid signala pina T0, uvećaj 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.

U vježbama ćemo koristiti normalan način rada, *Fast PWM* i *Phase Correct PWM* način rada te ćemo ih detaljnije opisati u nastavku. Opis preostalih načina rada nalazi se u literaturi [1] na stranici 98. Način rada sklopa *Timer/Counter0* odabire se pomoću bitova **WGM02**, **WGM01** i **WGM00** u registrima **TCCROA** i **TCCROB** prema tablici 8.2. Navedenu tablicu koristit ćemo za konfiguriranje načina rada tajmera.

Tablica 8.2: Odabir načina rada tajmera

WGM02	WGM01	WGM00	Način rada	Vršna vrijednost brojača
0	0	0	Normalni način rada	0xFF
0	0	1	<i>Phase Correct</i> način rada	0xFF
0	1	0	<i>CTC</i> način rada	OCROA
0	1	1	<i>Fast PWM</i> način rada	0xFF
1	0	0	Rezervirano	—
1	0	1	<i>Phase Correct</i> način rada	OCROA
1	1	0	Rezervirano	—
1	1	1	<i>Fast PWM</i> način rada	OCROA

## 8.1 Vježbe - Normalan način rada tajmera

Normalan način rada tajmera najjednostavniji je način rada. U ovom načinu rada uvijek se vrijednost registra `TCNT0` povećava za 1. Kada vrijednost u registru `TCNT0` prelazi iz 255 u 0, generira se prekid `TOV0` koji poziva prekidnu rutinu.

Prekidne se rutine u razvojnom programskom okruženju *Atmel Studio 7* pozivaju makronaredbom `ISR(vector)` (engl. *Interrupt Service Routines*). Ova makronaredba zahtijeva uključenje zaglavlja `#include <avr/interrupt.h>`. Makronaredba `ISR` kao argument prima prekidni vektor koji je definiran tablicom 9-1 u literaturi [1]. Prekidni vektor za preljev u registru `TCNT0` naziva se `TIMERO_OVF_vect`. Da bi se prekid mogao generirati potrebno je globalno omogućiti prekide te omogućiti svaki prekid zasebno. Globalno se prekidi omogućuju makronaredbom `sei()`, a onemogućuju makronaredbom `cli()`. Također, ako koristite zaglavlje koje je pripremio autor (`#include "Interrupt/interrupt.h"`), globalno se prekidi mogu omogućiti funkcijom `interrupt_enable()`, a onemogućiti funkcijom `interrupt_disable()`.

Generiranje prekida za preljev u registru `TCNT0` možemo omogućiti u registru `TIMSK0` tako da bit 0 (konstanta za bit 0 je `TOIE0`) postavimo u 1. Primjer konfiguracije tajmera prikazan je u programskom kodu 8.1.

Programski kod 8.1: Konfiguracija tajmera za normalan način rada

```
sei(); //globalno omogućen prekid
TIMSK0 |= (1 << TOIE0); // omogućenje prekida za timer0
TCCR0A |= (0 << WGM01) | (0 << WGM00); // Normalan način rada
TCCR0B |= (0 << WGM02); // Normalan način rada
TCCR0B |= (1 << CS02) | (0 << CS01) | (0 << CS00); // F_CPU/256
```

Makronaredbom `sei()` globalno smo omogućili prekide, a naredbom `TIMSK0 |= (1 << TOIE0);` prekid koji se generira kada se u registru `TCNT0` dogodi preljev. Normalan način rada tajmera omogućili smo naredbama `TCCR0A |= (0 << WGM01) | (0 << WGM00);` i `TCCR0B |= (0 << WGM02);` prema tablici 8.2. Pretpostavimo da je frekvencija radnog takta 16 MHz. Kada ne bi bilo dijeljenja frekvencije radnog takta, registar `TCNT0` svoju bi vrijednost uvećao za 1 ukupno 16 milijuna puta u jednoj sekundi i generirao 62500 prekida preljevom registra `TCNT0`. U registar `TCCR0B` na mjesto zadnjih triju bitova upisana je vrijednost 0x04 naredbom `TCCR0B |= (1 << CS02) | (0 << CS01) | (0 << CS00);`. Prema tablici 8.1, za ovu kombinaciju bitova `CS02`, `CS01` i `CS00` djelitelj frekvencije radnog takta je 256. Registar `TCNT0` u ovom slučaju svoju vrijednost uvećava za jedan na svaki 256. impuls radnog takta. Na taj način registar `TCNT0` svoju će vrijednost uvećati za 1 samo  $16000000/256 = 62500$  puta u jednoj

sekundi i generirati 244 prekida. Vrijeme između dvaju prekida bilo bi  $1 \text{ s}/244 = 0.0041 \text{ s} = 4.1 \text{ ms}$ , a ono se općenito može izračunati pomoću sljedeće relacije (vrijedi za tajmer rezolucije 8 bitova):

$$t_{T0} = \frac{PRESCALER}{F\_CPU} \cdot (256 - TCNT0_0) \quad (8.1)$$

gdje je:

- $t_{T0}$  - vrijeme između dvaju prekida, [s],
- *PRESCALER* - djelitelj frekvencije radnog takta,
- *F\_CPU* - frekvencija radnog takta mikroupravljača,
- $256 - 2^8$  gdje broj 8 predstavlja rezoluciju sklopa *Timer/Counter0* (za *Timer/Counter1* i *Timer/Counter3* umjesto 256 potrebno je staviti  $2^{16} = 65536$ ),
- $TCNT0_0$  - početna vrijednost registra **TCNT0**. Razlika  $256 - TCNT0_0$  odgovara broju impulsa koji na frekvenciji radnog takta uz zadani djelitelj frekvencije radnog takta traje  $t_{T0}$  vremena.

U normalnom načinu rada tajmer uvećava svoju vrijednost za 1 na svaki impuls koji dolazi s djelitelja frekvencije radnog takta. Ako vrijeme između dvaju prekida traje 100 impulsa, tada početna vrijednost registra **TCNT0** nije 100, već  $256 - 100 = 156$ . Budući da tajmer uvećava svoju vrijednost za 1, tada će od početne vrijednosti registra **TCNT0** ( $TCNT0_0 = 156$ ) do prelaska iz 255 u 0 (preljev) proći točno 100 impulsa. Kada je početna vrijednost registra **TCNT0** jednaka 0, vrijeme između dvaju prekida traje 256 impulsa.

Primjena ove vrste prekida jest pozivanje prekidne rutine svakih  $t_{T0}$  vremena (npr. potrebno je mjeriti temperaturu peći na kruta goriva svakih 10 ms). Varijable u relaciji (8.1) koje možemo mijenjati jesu *PRESCALER*, *F\_CPU* i  $TCNT0_0$ . Frekvenciju radnog takta u pravilu ne mijenjamo, već ju fiksiramo na samom početku razvoja programskog rješenja. Ono što možemo mijenjati jesu varijable *PRESCALER* i  $TCNT0_0$ . Kako bismo dobili željeno vrijeme između dvaju prekida, potrebno je iz relacije (8.1) izračunati  $TCNT0_0$ :

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F\_CPU}{PRESCALER} \quad (8.2)$$

U relaciji (8.2) potrebno je pronaći najmanji djelitelj frekvencije radnog takta za koji će vrijediti da početna vrijednost registra **TCNT0** nije manja od 0. Pretpostavimo da je frekvencija radnog takta 16 MHz. Pokušajmo u relaciju (8.2) uvrstiti *PRESCALER* = 256 ( $t_{T0} = 10 \text{ ms} = 0,01 \text{ s}$ ):

$$TCNT0_0 = 256 - 0,01 \cdot \frac{16000000}{256} = -369. \quad (8.3)$$

Početna vrijednost registra **TCNT0** prema (8.3) manja je od 0 pa je potrebno pokušati s većim djeliteljem frekvencije radnog takta (npr. *PRESCALER* = 1024):

$$TCNT0_0 = 256 - 0,01 \cdot \frac{16000000}{1024} = 99,75 \approx 100. \quad (8.4)$$

Početna vrijednost registra **TCNT0** prema (8.4) veća je od 0 pa je proračun gotov. U programskom kodu 8.1 sada moramo zamijeniti naredbu za namještanje djelitelja frekvencije radnog takta s naredbom `TCCR0B |= (1 << CS02) | (0 << CS01) | (1 << CS00)`; prema tablici

(8.1). Rezultat nije cijeli broj pa ga je potrebno zaokružiti na najbliži cijeli broj. Uvrstimo u relaciju 8.1 početnu vrijednost registra `TCNT0` koju smo izračunali:

$$t_{T0} = \frac{1024}{16000000} \cdot (256 - 100) = 9,98ms. \quad (8.5)$$

Stvarno vrijeme između dvaju prekida neće biti 10 ms, već 9,98 ms. Razlog tome jest nedovoljna rezolucija sklopa *Timer/Counter0*. U programskom kodu 8.2 prikazana je prekidna rutina za prekidni vektor `TIMERO_OVF_vect`. U ovoj prekidnoj rutini svakih se 20 ms mjeri temperatura peći na kruta goriva. Prekid se javlja onog trenutka kada dođe do preljeva u registru `TCNT0`, odnosno kada vrijednost registra `TCNT0` prelazi iz 255 u 0. Čim dođe do preljeva, poziva se prekidna rutina `ISR(TIMERO_OVF_vect)` u kojoj se početna vrijednost registra `TCNT0` softverski postavlja u proračunatu vrijednost 100. Registar `TCNT0` prilikom brojanja impulsa u ovom slučaju poprima vrijednosti 100, 101, ..., 253, 254, 255, (0) → 100, 101, ... .

Programski kod 8.2: Prekidna rutina za prekidni vektor `TIMERO_OVF_vect`

```
ISR(TIMERO_OVF_vect){
    TCNT0 = 100;
    // niz naredbi koje mjere temperaturu peći na kruta goriva
}
```

U vježbama će biti prikazan niz funkcija koje su napisane za konfiguriranje tajmera, a koje se nalaze u biblioteci `timer.h`.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) 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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se `Ivica Ivic`. Datoteku `Timer.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atsln` u datoteci `\\Timer\vjezbe`. U otvorenom projektu nalaze se sve 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 8.1.1

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje crvene LED diode svakih 15 ms. Crvena LED dioda spojena je na pin PB4. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba811.cpp`. Omogućite samo prevođenje datoteke `vjezba811.cpp`. Početni sadržaj datoteke `vjezba811.cpp` prikazan je programskim kodom 8.3.

Programski kod 8.3: Početni sadržaj datoteke `vjezba811.cpp`

```
#include "AVR VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"
```

```

// prekidna rutina za timer0
ISR(TIMER0_OVF_vect) {
    TCNT0 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB4);
}

void inicijalizacija() {
    DDRB |= (1 << PB4); // PB4 izlazni pin
    // inicijalizacija za timer0
    timer0_set_normal_mode();
    timer0_set_prescaler(TIMER0_PRESCALER_1024);
    timer0_interrupt_OVF_enable();

    lcd_init(); // inicijalizacija LCD displeja
    //interrupt_enable(); // globalno omogući prekide //sei();
    TCNT0 = 0; // početna vrijednost registra
}

int main(void) {

    inicijalizacija();
    uint16_t minute = 0;

    while(1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje 60 s
    }
    return 0;
}

```

Funkcije koje se koriste za inicijalizaciju tajmera deklarirane su u zaglavlju `timer.h`. Naredba kojom uključujemo zaglavlje `timer.h` u datoteku koja se prevodi jest `#include "Timer/timer.h"`. U ovoj vježbi koristit ćemo sklop *Timer/Counter0* kao tajmer u normalnom načinu rada. Ovaj način koristi prekide pomoću kojih se prekidna rutina poziva u jednakim vremenskim razmacima. Kada se koriste prekidi, u programski kod potrebno je uključiti zaglavlje `interrupt.h` pomoću naredbe `#include <avr/interrupt.h>`. Autor je pripremio vlastito zaglavlje `interrupt.h` koje se uključuje naredbom `#include "Interrupt/interrupt.h"`, a unutar kojeg se nalazi poziv naredbe `#include <avr/interrupt.h>`.

U programskom kodu 8.3 u funkciji `inicijalizacija()` nalaze se sljedeće funkcije:

- `timer0_set_normal_mode()` - funkcija kojom se sklop *Timer/Counter0* postavlja u normalan način rada prema tablici 8.2,
- `timer0_set_prescaler(TIMER0_PRESCALER_1024)` - funkcija kojom se konfigurira djelitelj frekvencije radnog takta,
- `timer0_interrupt_OVF_enable()` - funkcija kojom se omogućuje prekid sklopa *Timer/Counter0* kada dođe do preljeva registra `TCNT0`.

Funkcija `timer0_set_prescaler(uint8_t prescaler)` kao argument (`uint8_t prescaler`) prima predefinirane konstante koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 8.4. Na primjer, ako želimo postaviti djelitelj frekvencije radnog takta na iznos 256, tada ćemo pozvati funkciju `timer0_set_prescaler(TIMER0_PRESCALER_256)`. Argument pozvane funkcije jest `TIMER0_PRESCALER_256`, a on prema tablici 8.1 i definiciji u programskom kodu 8.4 odgovara dijeljenju radnog takta s 256.

Programski kod 8.4: Odabir djelitelj frekvencije radnog takta u datoteci `timer.h`

```
//djelitelji frekvencije za timer0
#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))
```

U funkciji `inicijalizacija()` potrebno je pozvati funkciju `interrupt_enable()` koja globalno omogućuje prekide. Ispod poziva funkcije `interrupt_enable()` potrebno je postaviti početnu vrijednost registra `TCNT0`. Trenutna početna vrijednost jest 0, što ćemo kasnije promijeniti.

Uvijek kada se dogodi preljev registra `TCNT0`, poziva se prekidna rutina prikazana programskim kodom 8.5.

Programski kod 8.5: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT0`

```
ISR(TIMER0_OVF_vect) {
    TCNT0 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB4);
}
```

Kada se pozove prekidna rutina, potrebno je postaviti početnu vrijednost registra `TCNT0` kako bi sljedeći poziv prekidne rutine nastupio nakon željenog vremena. U tu svrhu koristit ćemo relaciju (8.2). U datoteci `timer.h` odabran je djelitelj frekvencije radnog takta 1024. U vježbi je prekidnu rutinu potrebno pozivati svakih 15 ms, pa vrijedi da je  $t_{T0} = 0,015$  s. Početnu vrijednost registra `TCNT0` možemo izračunati prema relaciji (8.2):

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F_{CPU}}{PRESCALER} = 256 - 0,015 \cdot \frac{16000000}{1024} = 22. \quad (8.6)$$

Početnu vrijednost registra `TCNT0` upišite u funkciju `inicijalizacija()` i u prekidnu rutinu `ISR(TIMER0_OVF_vect)`. U prekidnoj se rutini mijenja stanje crvene LED diode makronaredbom `toggle_port`.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikroupravljača u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikroupravljač u `while` petlji 60 sekundi ništa ne radi. Prekidna se rutina poziva bez obzira na kašnjenje u `while` petlji i to je velika korist prekidnih rutina uopće.

Prevedite datoteku `vjezba811.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ukoliko crvena LED dioda ne izmjenjuje stanje, izbrišite komentar ispred funkcije `interrupt_enable()`; u funkciji `inicijalizacija()`. Budući da prekidi nisu bili omogućeni na globalnoj razini, prekidna rutina `ISR(TIMER0_OVF_vect)` neće biti pozvana. Ponovno prevedite datoteku `vjezba811.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

U nastavku vježbe izbrisat ćemo funkcije `timer0_set_normal_mode()`, `timer0_set_prescaler(TIMER0_PRESCALER_1024)` i `timer0_interrupt_OVF_enable()` iz funkcije `inicijalizacija()` te napisati niz naredbi iz programskog koda 8.6.

U programskom kodu 8.6 direktno smo konfigurirali bitove registara `TCCR0A` i `TCCR0B` prema tablicama 8.1 i 8.2 te slikama 8.3 i 8.4. U registru `TIMSK0` omogućili smo prekid koji se događa prilikom preljeva registra `TCNT0`. Prevedite datoteku `vjezba811.cpp` u strojni kod i snimite ga na

mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Programski kod 8.6: Konfiguracija sklopa *Timer/Counter0* direktnom konfiguracijom registara

```
// postaviti normalna način rada za timer0
TCCR0A |= (0 << WGM01) | (0 << WGM00);
TCCR0B |= (0 << WGM02);
//postaviti djelitelj frekvencije na 1024
TCCR0B |= (1 << CS02) | (0 << CS01) | (1 << CS00);
// omogućiti prekid za timer0 - overflow
TIMSK0 |= (1 << TOIE0);
```

U zaglavlju `avr/vub.h` nalaze se makronaredbe koje omogućuju postavljanje bitova u registrima u vrijednosti 0 i 1:

- `set_bit_reg(reg, bit)` - makronaredba koja prima argument `reg` koji predstavlja registar u kojem će se na poziciji argumenta `bit` postaviti 1,
- `reset_bit_reg(reg, bit)` - makronaredba koja prima argument `reg` koji predstavlja registar u kojem će se na poziciji argumenta `bit` postaviti 0.

U programskom kodu 8.7 konfiguraciju sklopa *Timer/Counter0* napravili smo pomoću makronaredbi `set_bit_reg` i `reset_bit_reg` prema tablicama 8.1 i 8.2 te slikama 8.3 i 8.4. Ovaj način konfiguracije sklopa *Timer/Counter0* jednostavan je, a daje jasan pregled stanja bitova u registrima sklopa *Timer/Counter0*.

Programski kod 8.7: Konfiguracija sklopa *Timer/Counter0* pomoću makronaredbi `set_bit_reg` i `reset_bit_reg`

```
// postaviti normalna način rada za timer0
reset_bit_reg(TCCR0A, WGM00); //WGM00 = 0
reset_bit_reg(TCCR0A, WGM01); //WGM01 = 0
reset_bit_reg(TCCR0B, WGM02); //WGM02 = 0
//postaviti djelitelj frekvencije na 1024
set_bit_reg(TCCR0B, CS00); // CS00 = 1
reset_bit_reg(TCCR0B, CS01); // CS01 = 0
set_bit_reg(TCCR0B, CS02); // CS02 = 1
// omogućiti prekid za timer0 - overflow
set_bit_reg(TIMSK0, TOIE0); // TOIE0 = 1
```

Konfigurirajte sklop *Timer/Counter0* pomoću niza naredbi u programskom kodu 8.7, a zatim prevedite datoteku `vjezba811.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba811.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.2

Napravite program u kojem ćete pomoću sklopa *Timer/Counter1* mijenjati stanje žute LED diode svakih 200 ms. Žuta LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba812.cpp`. Omogućite samo prevođenje datoteke `vjezba812.cpp`. Početni sadržaj datoteke `vjezba812.cpp` prikazan je programskim kodom 8.8.



Programski kod 8.8: Početni sadržaj datoteke vjezba812.cpp

```

#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

// prekidna rutina za timer1
ISR(TIMER1_OVF_vect) {
    TCNT1 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB5);
}

void inicijalizacija() {
    DDRB |= (1 << PB5); // PB5 izlazni pin
    // inicijalizacija za timer1 - prvi način
    timer1_set_normal_mode();
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_interrupt_OVF_enable();

    lcd_init(); // inicijalizacija LCD displeja
    interrupt_enable(); // globalno omogući prekide //sei();
    TCNT1 = 0; // početna vrijednost registra
}

int main(void) {

    inicijalizacija();
    uint16_t minute = 0;

    while(1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje 60 s
    }
    return 0;
}

```

U ovoj vježbi koristit ćemo sklop *Timer/Counter1* kao tajmer u normalnom načinu rada. Rezolucija sklopa *Timer/Counter1* jest 16 bitova pa se raspon vrijednosti u registru u kojem se broje impulsi kreće od [0, 65535]. Registar u kojem se broje impulsi zove se **TCNT1**. Primijetite da se ime registra sklopa *Timer/Counter1* od imena registra sklopa *Timer/Counter0* razlikuje samo u indeksu (0 je zamijenjena s 1).

Konfiguracija sklopa *Timer/Counter1* izvršava se pomoću dvaju registara: **TCCR1A** i **TCCR1B**. Detalje o konfiguraciji sklopa *Timer/Counter1* pogledajte u literaturi [1] u tablicama 14-1 do 14-5.

U programskom kodu 8.8 u funkciji **inicijalizacija()** nalaze se sljedeće funkcije:

- **timer1\_set\_normal\_mode()** - funkcija kojom se sklop *Timer/Counter1* postavlja u normalan način rada prema tablici 14-4 u literaturi [1],
- **timer1\_set\_prescaler(TIMER1\_PRESCALER\_64)** - funkcija kojom se konfigurira djelitelj frekvencije radnog takta,
- **timer1\_interrupt\_OVF\_enable()** - funkcija kojom se omogućuje prekid sklopa *Timer/Counter1* kada dođe do preljeva registra **TCNT1**.

Funkcija `timer1_set_prescaler(uint8_t prescaler)` kao argument (`uint8_t prescaler`) prima predefinirane konstante koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 8.9. Na primjer, ako želimo postaviti djelitelj frekvencije radnog takta na iznos 8, tada ćemo pozvati funkciju `timer1_set_prescaler(TIMER1_PRESCALER_8)`. Argument pozvane funkcije jest `TIMER1_PRESCALER_8`, a on prema tablici 14-5 u literaturi [1] i definiciji u programskom kodu 8.9 odgovara dijeljenju radnog takta s 8.

Programski kod 8.9: Odabir djelitelj frekvencije radnog takta u datoteci `timer.h`

```
//djelitelji frekvencije za timer1
#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))
```

Prema funkcijama za inicijalizaciju sklopa *Timer/Counter1* koje su pozvane u funkciji `inicijalizacija()`, sklop *Timer/Counter1* konfiguriran je kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 64. Prekid koji izaziva preljev u registru `TCNT1` omogućuje se upisivanjem broja 1 na mjesto bita `TOIE1` u registru `TIMSK1`. Za globalno omogućavanje prekida pozvana je funkcija `interrupt_enable()`.

Vrijeme između dvaju poziva prekidne rutine mora biti 200 ms ( $t_{T1} = 0,2$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 0,2 \cdot \frac{16000000}{64} = 15536. \quad (8.7)$$

Spomenuli smo ranije da je kod sklopa *Timer/Counter1* u relaciji (8.2) broj 256 potrebno zamijeniti brojem 65536. Početna vrijednost registra `TCNT1` jest 15536. Upišite tu vrijednost na odgovarajuća mjesta u programskom kodu 8.8.

Prekidna rutina `ISR`, koja se poziva kada se dogodi preljev registra `TCNT1`, prima prekidni vektor `TIMER1_OVF_vect`. U prekidnoj rutini postavlja se početna vrijednost registra `TCNT1` te se mijenja stanje žute LED diode makronaredbom `toggle_port`.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikroupravljača u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikroupravljač u `while` petlji 60 sekundi ništa ne radi. Prekidna rutina poziva se bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba812.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

U programskom kodu 8.10 konfiguraciju sklopa *Timer/Counter1* napravili smo pomoću makronaredbi `set_bit_reg` i `reset_bit_reg` prema tablicama 14-4 i 14-5 u literaturi [1]. Zamijenite sada konfiguraciju sklopa *Timer/Counter1* u programskom kodu 8.8 programskim kodom 8.10.

Programski kod 8.10: Konfiguracija sklopa *Timer/Counter1* pomoću makronaredbi `set_bit_reg` i `reset_bit_reg`

```
// postaviti normalna nacin rada za timer1
reset_bit_reg(TCCR1A, WGM10); //WGM10 = 0
reset_bit_reg(TCCR1A, WGM11); //WGM11 = 0
reset_bit_reg(TCCR1B, WGM12); //WGM12 = 0
reset_bit_reg(TCCR1B, WGM13); //WGM13 = 0
```

```

//postaviti djelitelj frekvencije na 64
set_bit_reg(TCCR1B, CS10); // CS10 = 1
set_bit_reg(TCCR1B, CS11); // CS11 = 1
reset_bit_reg(TCCR1B, CS12); // CS12 = 0
// omoguciti prekid za timer1 - overflow
set_bit_reg(TIMSK1, TOIE1); // TOIE1 = 1

```

Prevedite datoteku `vjezba812.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite vrijeme između poziva prekidne rutine u 500 ms. Ponovno prevedite datoteku `vjezba812.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba812.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.3

Napravite program u kojem ćete pomoću sklopa *Timer/Counter3* mijenjati stanje zelene LED diode svakih 1000 ms. Zelena LED dioda spojena je na pin PB6. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba813.cpp`. Omogućite samo prevođenje datoteke `vjezba813.cpp`. Početni sadržaj datoteke `vjezba813.cpp` prikazan je programskim kodom 8.11.

Programski kod 8.11: Početni sadržaj datoteke `vjezba813.cpp`

```

#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

// prekidna rutina za timer3
ISR(TIMER3_OVF_vect) {
    TCNT3 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB6);
}

void inicijalizacija() {
    DDRB |= (1 << PB6); // PB6 izlazni pin
    // inicijalizacija za timer3
    timer3_set_normal_mode();
    timer3_set_prescaler(TIMER3_PRESCALER_256);
    timer3_interrupt_OVF_enable();

    lcd_init(); // inicijalizacija LCD displeja
    interrupt_enable(); // globalno omogući prekide //sei();
    TCNT3 = 0; // početna vrijednost registra
}

int main(void) {

    inicijalizacija();
    uint16_t minute = 0;

    while(1) {

```

```

        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje 60 s
    }
    return 0;
}

```

U ovoj vježbi koristit ćemo sklop *Timer/Counter3* kao tajmer u normalnom načinu rada. Rezolucija sklopa *Timer/Counter3* jest 16 bitova pa se raspon vrijednosti u registru u kojem se broje impulsi kreće od [0, 65535]. Registar u kojem se broje impulsi zove se **TCNT3**.

Konfiguracija sklopa *Timer/Counter3* izvršava se pomoću dvaju registara: **TCCR3A** i **TCCR3B**. Detalje o konfiguraciji sklopa *Timer/Counter3* pogledajte u literaturi [1] u tablicama 14-1 do 14-5.

U programskom kodu 8.11 u funkciji `inicijalizacija()` nalaze se sljedeće funkcije:

- `timer3_set_normal_mode()` - funkcija kojom se sklop *Timer/Counter3* postavlja u normalan način rada prema tablici 14-4 u literaturi [1],
- `timer3_set_prescaler(TIMER3_PRESCALER_256)` - funkcija kojom se konfigurira djelitelj frekvencije radnog takta,
- `timer3_interrupt_OVF_enable()` - funkcija kojom se omogućuje prekid sklopa *Timer/Counter3* kada dođe do preljeva registra **TCNT3**.

Funkcija `timer3_set_prescaler(uint8_t prescaler)` kao argument (`uint8_t prescaler`) prima predefinirane konstante koje se nalaze u zaglavlju `timer.h` i prikazane su programskim kodom 8.12.

Programski kod 8.12: Odabir djelitelj frekvencije radnog takta u datoteci `timer.h`

```

//djelitelji frekvencije za timer1
#define TIMER3_PRESCALER_OFF      ((0 << CS32) | (0 << CS31) | (0 << CS30))
#define TIMER3_PRESCALER_1       ((0 << CS32) | (0 << CS31) | (1 << CS30))
#define TIMER3_PRESCALER_8       ((0 << CS32) | (1 << CS31) | (0 << CS30))
#define TIMER3_PRESCALER_64      ((0 << CS32) | (1 << CS31) | (1 << CS30))
#define TIMER3_PRESCALER_256     ((1 << CS32) | (0 << CS31) | (0 << CS30))
#define TIMER3_PRESCALER_1024    ((1 << CS32) | (0 << CS31) | (1 << CS30))
#define TIMER3_EXTERNAL_FALL_EDGE ((1 << CS32) | (1 << CS31) | (0 << CS30))
#define TIMER3_EXTERNAL_RISI_EDGE ((1 << CS32) | (1 << CS31) | (1 << CS30))

```

Prema funkcijama za inicijalizaciju sklopa *Timer/Counter3* koje su pozvane u funkciji `inicijalizacija()`, sklop *Timer/Counter3* konfiguriran je kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 256. Prekid koji izaziva preljev u registru **TCNT3** omogućuje se upisivanjem broja 1 na mjesto bita **TOIE3** u registru **TIMSK3**. Za globalno omogućavanje prekida pozvana je funkcija `interrupt_enable()`.

Vrijeme između dvaju poziva prekidne rutine mora biti 1000 ms ( $t_{T3} = 1,0$  s). Početnu vrijednost registra **TCNT3** izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT3_0 = 65536 - t_{T3} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 1,0 \cdot \frac{16000000}{256} = 3036. \quad (8.8)$$

Kod sklopa *Timer/Counter3* u relaciji (8.2) broj 256 potrebno je zamijeniti brojem 65536 jer je registar **TCNT3** širine 16 bitova. Početna vrijednost registra **TCNT3** jest 3036. Upišite tu vrijednost na odgovarajuća mjesta u programskom kodu 8.11.

Prekidna rutina `ISR`, koja se poziva kada se dogodi preljev registra `TCNT3`, prima prekidni vektor `TIMER3_OVF_vect`. U prekidnoj rutini postavlja se početna vrijednost registra `TCNT3` te se mijenja stanje zelene LED diode makronaredbom `toggle_port`.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikroupravljača u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikroupravljač u `while` petlji 60 sekundi ništa ne radi. Prekidna rutina poziva se bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba813.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite vrijeme između poziva prekidne rutine u 1500 ms. Ponovno prevedite datoteku `vjezba813.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba813.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.4

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje crvene LED diode svakih 300 ms, pomoću sklopa *Timer/Counter1* mijenjati stanje žute LED diode svakih 300 ms, a pomoću sklopa *Timer/Counter3* mijenjati stanje zelene LED diode svakih 300 ms. Crvena LED dioda spojena je na pin PB4, žuta LED dioda spojena je na pin PB5, a zelena LED dioda spojena je na pin PB6. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba814.cpp`. Omogućite samo prevođenje datoteke `vjezba814.cpp`. Početni sadržaj datoteke `vjezba814.cpp` prikazan je programskim kodom 8.13.

Programski kod 8.13: Početni sadržaj datoteke `vjezba814.cpp`

```
#include "AVR_VUB/avrubb.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

// umetnite programski kod 8.14

// prekidna rutina za timer1
ISR(TIMER1_OVF_vect) {
    TCNT1 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB5);
}

// prekidna rutina za timer3
ISR(TIMER3_OVF_vect) {
    TCNT3 = 0; // početna vrijednost registra
    toggle_port(PORTB, PB6);
}

void inicijalizacija() {
    DDRB |= (1 << PB4) | (1 << PB5) | (1 << PB6); // PB4, PB5 i PB6 izlazni pinovi

    // inicijalizacija za timer0
```

```

timer0_set_normal_mode();
//timer0_set_prescaler(TIMER0_PRESCALER_);
timer0_interrupt_OVF_enable();

// inicijalizacija za timer1
timer1_set_normal_mode();
//timer1_set_prescaler(TIMER1_PRESCALER_);
timer1_interrupt_OVF_enable();

// inicijalizacija za timer3
timer3_set_normal_mode();
//timer3_set_prescaler(TIMER3_PRESCALER_);
timer3_interrupt_OVF_enable();

lcd_init(); // inicijalizacija LCD displeja
interrupt_enable(); // globalno omogući prekide //sei();

TCNT0 = 0; // pocetna vrijednost registra za timer0
TCNT1 = 0; // pocetna vrijednost registra za timer1
TCNT3 = 0; // pocetna vrijednost registra za timer3
}

int main(void) {

    inicijalizacija();
    uint16_t minute = 0;

    while(1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje 60 s
    }
    return 0;
}

```

U programskom kodu 8.13 u funkciji `inicijalizacija()` nalaze se funkcije koje konfiguriraju rad sklopova *Timer/Counter0*, *Timer/Counter1* i *Timer/Counter3* u normalnom načinu rada. Omogućen je i prekid za sve tajmere. Da bi se ostvarila zadana izmjena stanja crvene, žute i zelene LED diode svakih 300 ms, potrebno je proračunati početna stanja registara `TCNT0`, `TCNT1` i `TCNT3` te odrediti djelitelje frekvencije radnog takta za sve tajmere.

Vrijeme između dvaju poziva prekidne rutine `ISR(TIMERO_OVF_vect)` mora biti 300 ms ( $t_{T0} = 0,3$  s). Početnu vrijednost registra `TCNT0` izračunat ćemo pomoću relacije (8.2):

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F_{CPU}}{PRESCALER_0} = 256 - 0,3 \cdot \frac{16000000}{1024} = -4431,5. \quad (8.9)$$

Početna vrijednost registra `TCNT0` dobivena relacijom (8.9) manja je od 0. To znači da s najvećim djeliteljem frekvencije radnog takta nije moguće ostvariti vrijeme između dvaju poziva prekidne rutine od 300 ms. Maksimalno vrijeme između dvaju poziva prekidne rutine `ISR(TIMERO_OVF_vect)` jest ono vrijeme za koje početna vrijednost registra `TCNT0` iznosi 0:

$$t_{T0} = \frac{PRESCALER}{F_{CPU}} \cdot (256 - TCNT0_0) = \frac{1024}{16000000} \cdot (256 - 0) = 16,384ms. \quad (8.10)$$

Prema relaciji (8.10), vremena između dvaju poziva prekidne rutine mogu se podesiti tako da budu manja od 16,384 s. Kako sada ipak ostvariti izmjenu stanja crvene LED diode svakih

300 ms? Za ovaj problem postoji rješenje koje ćemo prikazati u nastavku. Na primjer, poziv između dviju prekidnih rutina u vremenskom razmaku od 300 ms možemo podijeliti na 20 poziva prekidne rutine u vremenskom razmaku od 15 ms. Stanje crvene LED diode u prekidnoj rutini tada ćemo mijenjati u svakom dvadesetom pozivu prekidne rutine.

Izračunajmo novu početnu vrijednost registra `TCNT0` za dvadesetinu vremena od 300 ms:

$$TCNT0_0 = 256 - \frac{t_{T0}}{20} \cdot \frac{F_{CPU}}{PRESCALER\_0} = 256 - 0,015 \cdot \frac{16000000}{1024} = 21,625 \approx 22. \quad (8.11)$$

U programski kod 8.13 napišite prekidnu rutinu `ISR(TIMER0_OVF_vect)` prikazanu programskim kodom 8.14.

Programski kod 8.14: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT0`

```
uint8_t prazan_hod_timer0 = 0;

// prekidna rutina za timer0
ISR(TIMER0_OVF_vect) {
    TCNT0 = 0; // početna vrijednost registra
    if(++prazan_hod_timer0 == 20) {
        toggle_port(PORTB, PB4);
        prazan_hod_timer0 = 0;
    }
}
```

U programskom kodu 8.14 deklarirana je globalna varijabla `prazan_hod_timer0`. Ova varijabla koristi se kao brojač poziva prekidne rutine `ISR(TIMER0_OVF_vect)`. U svakom dvadesetom pozivu prekidne rutine mijenjamo stanje crvene LED diode. Vrijeme između dvaju poziva prekidne rutine jest 15 ms te se stanje crvene LED diode mijenja svakih 300 ms (15 ms · 20 = 300 ms).

Upišite početnu vrijednost registra `TCNT0`, uklonite komentar funkcije `timer0_set_prescaler` te dodijelite argument funkciji `timer0_set_prescaler` kojim će se konfigurirati djelitelj frekvencije radnog takta iznosa 1024.

Vrijeme između dvaju poziva prekidne rutine `ISR(TIMER1_OVF_vect)` mora biti 300 ms ( $t_{T1} = 0,3$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER\_1} = 65536 - 0,3 \cdot \frac{16000000}{256} = 46786. \quad (8.12)$$

Upišite početnu vrijednost registra `TCNT1`, uklonite komentar funkcije `timer1_set_prescaler` te dodijelite argument funkciji `timer1_set_prescaler` kojim će se konfigurirati djelitelj frekvencije radnog takta iznosa 256.

Vrijeme između dvaju poziva prekidne rutine `ISR(TIMER3_OVF_vect)` mora biti 300 ms ( $t_{T3} = 0,3$  s). Početnu vrijednost registra `TCNT3` izračunat ćemo pomoću relacije (8.2):

$$TCNT3_0 = 65536 - t_{T3} \cdot \frac{F_{CPU}}{PRESCALER\_3} = 65536 - 0,3 \cdot \frac{16000000}{256} = 46786. \quad (8.13)$$

Upišite početnu vrijednost registra `TCNT3`, uklonite komentar funkcije `timer3_set_prescaler` te dodijelite argument funkciji `timer3_set_prescaler` kojim će se konfigurirati djelitelj frekvencije radnog takta iznosa 256.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikroupravljača u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikroupravljač

u `while` petlji 60 sekundi ništa ne radi. Prekidna rutina poziva se bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba814.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Komentirajte zašto se izmjena stanja crvene diode vremenski razilazi s izmjenama stanja žute i zelene diode?

Zatvorite datoteku `vjezba814.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.5

Napravite program koji će omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 250 ms i to redoslijedom crvena → žuta → zelena → plava → crvena → ... . Izmjenu stanja LED dioda svakih 250 ms ostvarite pomoću sklopa *Timer/Counter1*. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba815.cpp`. Omogućite samo prevođenje datoteke `vjezba815.cpp`. Početni sadržaj datoteke `vjezba815.cpp` prikazan je programskim kodom 8.15.

Programski kod 8.15: Početni sadržaj datoteke `vjezba815.cpp`

```
#include "AVR_VUB/avrvub.h"
#include "LCD/lcd.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

uint8_t brojac = 0;

ISR(TIMER1_OVF_vect) { // prekidna rutina za timer1
    TCNT1 = 0; // početna vrijednost registra
    switch (++brojac) {
        case 1:
            set_port(PORTB, PB4, 1);
            set_port(PORTB, PB7, 0);
            break;
        case 2:
            set_port(PORTB, PB5, 1);
            set_port(PORTB, PB4, 0);
            break;
        case 3:
            set_port(PORTB, PB6, 1);
            set_port(PORTB, PB5, 0);
            break;
        case 4:
            set_port(PORTB, PB7, 1);
            set_port(PORTB, PB6, 0);
            brojac = 0;
            break;
        default:
            break;
    }
}
```



```

void inicijalizacija() {
    // PB4, PB5, PB6, PB7 izlazni pinovi
    DDRB |= (1 << PB4) | (1 << PB5) | (1 << PB6) | (1 << PB7);
    // inicijalizacija za timer1
    timer1_set_normal_mode();
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_interrupt_OVF_enable();

    lcd_init(); // inicijalizacija LCD displeja
    interrupt_enable(); // globalno omogući prekide //sei();
    TCNT1 = 0; // početna vrijednost registra
}

int main(void) {

    inicijalizacija();
    uint16_t minute = 0;

    while(1) {

        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje 60 s
    }
    return 0;
}

```

U ovoj vježbi koristit ćemo sklop *Timer/Counter1* kao tajmer u normalnom načinu rada. U programskom kodu 8.13 u funkciji `inicijalizacija()` nalaze se funkcije koje konfiguriraju rad sklopa *Timer/Counter1* u normalnom načinu rada, definiraju djelitelj frekvencije radnog takta i omogućuju prekid preljevom registra `TCNT1`.

U vježbi je potrebno ostvariti trčeće LED diode redosljedom crvena → žuta → zelena → plava → crvena → ... . Izmjenu stanja LED dioda svakih 250 ms potrebno je ostvariti pomoću sklopa *Timer/Counter1*. Sličan program napravili smo u vježbi 4.1.4, no u toj vježbi koristili smo vremenska kašnjenja.

U prekidnoj rutini `ISR(TIMER1_OVF_vect)` nalazi se `switch case` blok koji ima četiri slučaja. Svaki slučaj zadužen je za uključenje sljedeće LED diode i isključenje prethodno uključene LED diode (npr. `case 2`: crvena LED dioda se isključi, a žuta LED dioda se uključi). Slučaj u `switch case` bloku odabire se pomoću varijable `brojac`. U svakom pozivu prekidne rutine varijabla `brojac` uveća se za 1. U slučaju broj 4 (`case 4`:) vrijednost varijable `brojac` postavlja se u 0 kako bi se ostvarilo ponovno trčanje LED dioda.

Vrijeme između dvaju poziva prekidne rutine mora biti 250 ms ( $t_{T1} = 0,25$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 0,25 \cdot \frac{16000000}{64} = 3036. \quad (8.14)$$

Upišite početnu vrijednost registra `TCNT1` u programski kod 8.15 na za to predviđeno mjesto. U `while` petlji na LCD displej ispisuje se vrijeme rada mikroupravljača u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikroupravljač u `while` petlji 60 sekundi ništa ne radi. Prekidna rutina poziva se bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba815.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite smjer trčanja LED dioda te ponovno prevedite datoteku `vjezba815.cpp` u strojni

kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba815.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.6

Napravite program u kojem će se, ako je pritisnuto tipkalo spojeno na pin PD0, uzimati uzorci s analognog senzora spojenog na pin ADC5. Vrijeme uzorkovanja neka iznosi 800 ms. Uzimanje uzoraka ostvarite u prekidnoj rutini `ISR(TIMER3_OVF_vect)`. Vrijednost napona na pinu ADC5 ispišite na LCD displej u `while` petlji samo kada se uzimaju uzorci.

U projektnom stablu otvorite datoteku `vjezba816.cpp`. Omogućite samo prevođenje datoteke `vjezba816.cpp`. Početni sadržaj datoteke `vjezba816.cpp` prikazan je programskim kodom 8.16.

Programski kod 8.16: Početni sadržaj datoteke `vjezba816.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
// uključi sva potrebna zaglavlja

uint8_t uzorak_procitan = 0; // logička varijabla
uint16_t ADC_5; // vrijednost AD pretvorbe na pinu ADC5

// prekidna rutina za timer3
ISR(TIMER3_OVF_vect) {
    TCNT3 = 0; // početna vrijednost registra
    ADC_5 = adc_read(ADC5); // ADC na kanalu ADC5
    uzorak_procitan = 1;
}

void inicijalizacija() {

    // inicijalizacija za timer3

    // inicijalizacija LCD displeja
    // inicijalizacija AD pretvorbe

    // globalno omogući prekide //sei();
    TCNT3 = 0; // početna vrijednost registra
}

int main(void) {

    inicijalizacija();

    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon

    DigitalInput tipka(D0); // digitalni ulaz - tipka PD0
    tipka.pullup_on(); // uključenje pull up otpornika na PD0

    while(1) {

        if(tipka.state() == false) {
            timer3_interrupt_OVF_enable();
        }
        else {
            timer3_interrupt_OVF_disable();
        }
    }
}
```

```

        lcd_clrscr();
    }
    if(uzorak_procitan == 1) {
        U_ADC5 = ADC_5 * V_REF / 1023; // pretvorba u napon
        lcd_clrscr();
        lcd_home();
        lcd_print("UADC5 = %.2fV", U_ADC5);
        uzorak_procitan = 0;
    }
}
return 0;
}

```

U programskom kodu 8.16 prvo je potrebno uključiti sva zaglavlja koja nedostaju. Primijetite da se u vježbi koriste LCD displej, analogno-digitalna pretvorba, tajmer i prekidi. U vježbi je potrebno svakih 800 ms očitati uzorak s analognog senzora spojenog na pin ADC5 ako je pritisnuto tipkalo spojeno na pin PD0. U praksi se to uvijek radi s tajmerima jer uvijek postoji zahtjev za preciznost uzimanja uzoraka.

Mjerenje analognim senzorom provodi se pomoću prekidne rutine `ISR(TIMER3_OVF_vect)`. Uzorke je potrebno uzimati svakih 800 ms ( $t_{T3} = 0,8$  s) pa je početna vrijednost registra `TCNT3` izračunata prema korigiranoj relaciji (8.2):

$$TCNT3_0 = 65536 - t_{T3} \cdot \frac{F_{CPU}}{PRESCALER\_1} = 65536 - 0,8 \cdot \frac{16000000}{256} = 15536. \quad (8.15)$$

Upišite početnu vrijednost registra `TCNT3` na odgovarajuća mjesta u programskom kodu 8.16. U funkciji `inicijalizacija()` potrebno je inicijalizirati brojne uređaje. U programski kod 8.16 upišite sljedeće funkcije:

- `timer3_set_normal_mode()` - postavljanje sklopa *Timer/Counter3* u normalan način rada,
- `timer3_set_prescaler(TIMER1_PRESCALER_256)` - postavljanje djelitelj frekvencije radnog takta za sklop *Timer/Counter3* na iznos 256 (prema relaciji (8.15)),
- `lcd_init()` - inicijalizacija LCD displeja,
- `adc_init()` - inicijalizacija AD pretvorbe,
- `interrupt_enable()` - omogućenje prekida na globalnoj razini.

U funkciji `main()` stvoren je objekt `tipka` tipa digitalni ulaz na pinu PD0 te je uključen pritezni otpornik na pinu PD0. Prekidna rutina `ISR(TIMER3_OVF_vect)` poziva se samo ako je omogućen prekid koji izaziva preljev u registru `TCNT3`. Unutar `while(1)` petlje provjeravamo je li pritisnuto tipkalo te:

- ako je tipkalo pritisnuto (`tipka.state() == false`), tada je potrebno omogućiti prekid koji se izaziva preljevom registra `TCNT3` pomoću funkcije `timer3_interrupt_OVF_enable()`,
- inače, ako tipkalo nije pritisnuto, tada je potrebno onemogućiti prekid koji se izaziva preljevom registra `TCNT3` pomoću funkcije `timer3_interrupt_OVF_disable()` i izbrisati tekst na LCD displeju.

Na gore navedeni način ostvarili smo funkcionalnost uzimanja uzoraka samo kada je tipkalo spojeno na PD0 pritisnuto.

Vrijednost napona na analognom ulazu ADC5 ispisujemo na LCD displeju ako je zadovoljen uvjet `if (uzorak_procitan == 1)`. Varijablu `uzorak_procitan` koristimo kao indiciju da nam je na raspolaganju novi uzorak koji možemo ispisati na LCD displeju. Ovu varijablu deklarirali smo kao globalnu i dodijelili joj vrijednost 0 (nije dostupan novi uzorak za ispis na LCD displej). Uvijek kada se pozove prekidna rutina `ISR(TIMER3_OVF_vect)`, varijabli `uzorak_procitan` dodjeljujemo vrijednost 1 (dostupan je novi uzorak za ispis na LCD displej). Prilikom ispisa napona s analognog senzora na LCD displej, varijabli `uzorak_procitan` dodjeljujemo vrijednost 0. Ovaj postupak neophodan je kako bismo na LCD displeju spriječili neprestano ispisivanje napona koje bi rezultiralo nečitljivim prikazom na njemu. Primijetite kako je i varijabla `ADC_5` u koju se sprema rezultat analogno-digitalne pretvorbe na analognom ulazu ADC5 deklarirana kao globalna varijabla. Razlog tome je taj što se ovoj varijabli pristupa i u `main()` funkciji i u prekidnoj rutini `ISR(TIMER3_OVF_vect)`, stoga doseg ove varijable mora biti globalan.

Prevedite datoteku `vjezba816.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba816.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 8.1.1 Zadaci - Normalan način rada tajmera

#### Zadatak 8.1.1

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje plave LED diode svakih 16 ms. Plava LED dioda spojena je na pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

#### Zadatak 8.1.2

Napravite program u kojem ćete pomoću sklopa *Timer/Counter1* mijenjati stanje zelene LED diode svakih 180 ms. Zelena LED dioda spojena je na pin PB6. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

#### Zadatak 8.1.3

Napravite program u kojem ćete pomoću sklopa *Timer/Counter3* mijenjati stanje zelene LED diode svakih 1400 ms. Zelena LED dioda spojena je na pin PB6. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

#### Zadatak 8.1.4

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje crvene LED diode svakih 450 ms, pomoću sklopa *Timer/Counter1* mijenjati stanje žute LED diode svakih 450 ms, a pomoću sklopa *Timer/Counter3* mijenjati stanje zelene LED diode svakih 450 ms. Crvena LED dioda spojena je na pin PB4, žuta LED dioda spojena je na pin PB5, a zelena LED dioda spojena je na pin PB6. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

#### Zadatak 8.1.5

Napravite program koji će omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 320 ms i to redosljedom plava → zelena → žuta → crvena → plava →... . Izmjenu stanja LED dioda svakih 320 ms ostvarite pomoću sklopa *Timer/Counter3*. Crvena LED dioda spojena je na digitalni pin PB4, žuta LED dioda spojena je na digitalni pin PB5, zelena LED dioda spojena je na digitalni pin PB6, a plava LED dioda spojena je na digitalni pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikroupravljača u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

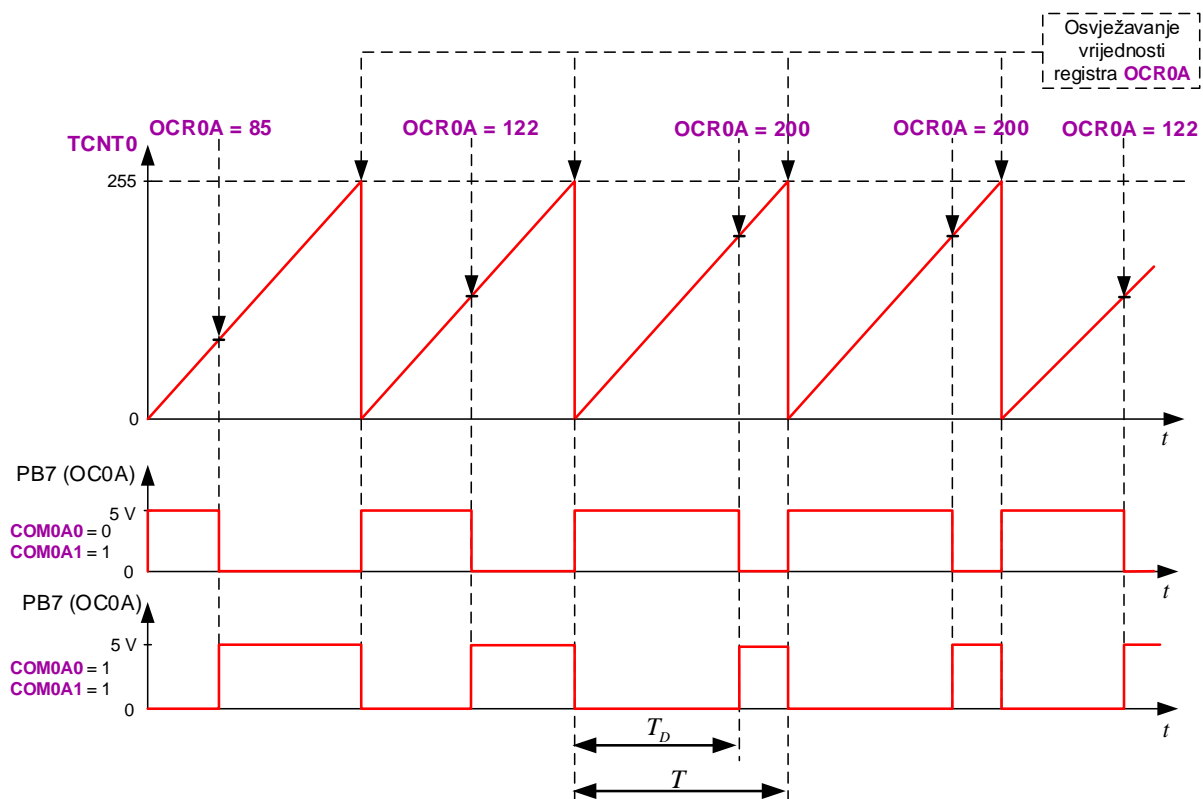
### Zadatak 8.1.6

Napravite program u kojem će se, ako je pritisnuto tipkalo spojeno na pin PF7, uzimati uzorci s analognog senzora spojenog na pin ADC5. Vrijeme uzorkovanja neka iznosi 700 ms. Uzimanje uzoraka ostvarite u prekidnoj rutini `ISR(TIMER1_OVF_vect)`. Vrijednost napona na pinu ADC5 ispišite na LCD displej u `while` petlji samo kada se uzimaju uzorci.

## 8.2 Vježbe - PWM način rada tajmera

### 8.2.1 *Fast* PWM način rada tajmera

*Fast* PWM način rada bit će objašnjen na sklopu *Timer/Counter0* za kanal A. U *Fast* PWM načinu rada tajmera na temelju vrijednosti u registrima `TCNT0` te registrima `OCR0A` (kanal A), odnosno `OCR0B` (kanal B) generira se pravokutni valni oblik visoke frekvencije na pinu PB7 (`OC0A`), odnosno pinu PD0 (`OC0B`). PWM najčešće se koristi kod upravljanja brzinom vrtnje istosmjernog motora, pretvarača napona, promjene inteziteta LED rasvjete i slično. Vremenski dijagram *Fast* PWM načina rada prikazan je na slici 8.5. Vrijednost registra `TCNT0` cijelo se vrijeme uvećava za 1 i kreće se od 0 (DNO) do 255 (VRH). Nakon što registar `TCNT0` postigne vrijednost 255, ponovno se postavlja u 0 te uvećava za 1 prema gore. 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.



Slika 8.5: Vremenski dijagram *Fast* PWM načina rada

Vrijednosti registara **TCNT0** i **OCROA** cijelo se vrijeme uspoređuju. U trenutku kada se vrijednosti ovih dvaju registara izjednače (**TCNT0** == **OCROA**) na pinu PB7 (OC0A) se dešava promjena stanja signala iz visokog u nisko ili obratno, što ovisi o konfiguraciji bitova **COMOA1** i **COMOAO**. Ovisno o stanju bitova **COMOA1** i **COMOAO**, prema tablici 8.3 na pinu PB7 (OC0A) generira se pravokutni valni oblik prikazan na slici 8.5.

Tablica 8.3: Postavke bitova **COMOA1** i **COMOAO** u *Fast* PWM načina rada (kanal A) [1]

<b>COMOA1</b>	<b>COMOAO</b>	Postavke <i>Fast</i> PWM načina rada
0	0	Pin PB7 (OC0A) isključen
0	1	Ako je <b>WGM02</b> = 0, tada je pin PB7 (OC0A) isključen. Ako je <b>WGM02</b> = 1, promjena stanja pina PB7 (OC0A) događa se kada je vrijednost registra <b>TCNT0</b> jednaka vrijednosti registra <b>OCROA</b> .
1	0	Pin PB7 (OC0A) postavlja se u nisko stanje kada je vrijednost registra <b>TCNT0</b> jednaka vrijednosti registra <b>OCROA</b> . Pin PB7 (OC0A) postavlja se u visoko stanje kada vrijednost registra <b>TCNT0</b> prijeđe iz 255 (VRH) u 0 (DNO).
1	1	PB7 (OC0A) postavlja se u visoko stanje kada je vrijednost registra <b>TCNT0</b> jednaka vrijednosti registra <b>OCROA</b> . PB7 (OC0A) postavlja se u nisko stanje kada vrijednost registra <b>TCNT0</b> prijeđe iz 255 (VRH) u 0 (DNO).

Ako je konfiguracija bitova **COMOA1** = 1 i **COMOAO** = 0, tada se pin PB7 (OC0A) postavlja u nisko stanje kada je vrijednost registra **TCNT0** jednaka vrijednosti registra **OCROA**, a u visoko stanje kada vrijednost registra **TCNT0** prijeđe iz 255 u 0. Ovakva vrsta konfiguracije generira neinvertirajući PWM signal (slika 8.5).

Ako je konfiguracija bitova **COMOA1** = 1 i **COMOAO** = 1, tada se pin PB7 (OC0A) postavlja u visoko stanje kada je vrijednost registra **TCNT0** jednaka vrijednosti registra **OCRO**, a u nisko stanje kada vrijednost registra **TCNT0** prijeđe iz 255 u 0. Ovakva vrsta konfiguracije generira invertirajući PWM signal (slika 8.5).

Blokovska shema na slici 8.2 prikazuje sklopovski dio koji provjerava jednakost registara **TCNT0** i **OCROA** te dio koji generira valni oblik na pinu PB7 (OC0A). PWM signal može se generirati i na pinu PD0 (OC0B) (kanal B) pomoću sklopa *Timer/Counter0*, no tada je potrebno pogledati konfiguraciju bitova **COMOB1** i **COMOBO** u literaturi [1] u tablici 13-5. PWM signal mogu generirati i ostala dva sklopa: *Timer/Counter1* i *Timer/Counter3*. Konfiguraciju PWM sklopova možete pronaći u literaturi [1] u tablicama 14-1 do 14-4.

Frekvencija PWM signala u *Fast* PWM načinu rada za sklop *Timer/Counter0* jest:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot (TOP + 1)} = \frac{F_{CPU}}{PRESCALER \cdot 256}, \quad (8.16)$$

a period PWM signala jest:

$$T = \frac{1}{F_{fPWM}} = \frac{PRESCALER \cdot (TOP + 1)}{F_{CPU}} = \frac{PRESCALER \cdot 256}{F_{CPU}}. \quad (8.17)$$

Frekvencija i period PWM signala ovise samo o djeljitelju frekvencije radnog takta, odnosno o varijabli *PRESCALER*. Varijablu *PRESCALER* odabiremo sukladno potrebama sustava na koji dovodimo PWM signal te o brzini elektroničkih komponenata koje u sustav dovode energiju. Varijabla *TOP* za sklop *Timer/Counter0* iznosi 255, dok za sklopove *Timer/Counter1*

i *Timer/Counter3* može iznositi 255, 511, 1023, kao i proizvoljan iznos u rasponu  $[0, 65535]^2$ . Širina impulsa neinvertirajućeg PWM signala na kanalu A određena je vremenom visokog stanja  $T_D$  (engl. *duty cycle*) u odnosu na vrijeme perioda  $T$  (slika 8.5) prema relaciji:

$$\frac{T_D}{T} = \frac{\text{OCROA}}{255}. \quad (8.18)$$

Širinu PWM signala na pinu PB7 (OC0A) mijenjamo promjenom vrijednosti registra **OCROA** prema relaciji:

$$\text{OCROA} = \frac{T_D}{T} \cdot 255. \quad (8.19)$$

Slučajevi u kojima registar **OCROA** poprima vrijednost 0 (DNO) ili 255 (VRH) specijalni su slučajevi. Kod neinvertirajućeg PWM signala (**COMOA1** = 1, **COMOAO** = 0) za specijalne slučajeve vrijedi:

- ako je **OCROA** = 255, tada je pin PB7 (OC0A) konstantno u visokom stanju,
- ako je **OCROA** = 0, tada će se se na pinu PB7 (OC0A) generirati uski šiljak.

Na primjer, ako upravljamo intezitetom LED diode u slučaju kada je **OCROA** = 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 sljedećem potpoglavlju.

Ako se koristi kanal B (uz **WGM02** = 0), tada širinu PWM signala na pinu PD0 (OC0B) mijenjamo promjenom vrijednosti registra **OCROB** prema relaciji:

$$\text{OCROB} = \frac{T_D}{T} \cdot 255. \quad (8.20)$$

Omjer  $\frac{T_D}{T}$  predstavlja postotak aktivnog stanja neinvertirajućeg PWM signala. Pretpostavimo da PWM signal koristimo u silaznom pretvaraču za koji vrijedi da se istosmjerni napon  $U_{DC}$  (npr.  $U_{DC} = 30$  V) pretvara u raspon od  $[0, U_{DC}]$  V. Neinvertirajući PWM signal na pinu PB7 (OC0A) upravlja radom sklopke u izvedbi unipolarnog tranzistora. Označimo izlazni napon na silaznom pretvaraču  $U_i$ . Izlazni napon može se izračunati prema relaciji:

$$U_i = \frac{T_D}{T} \cdot U_{DC} = \frac{\text{OCROA}}{255} \cdot U_{DC}. \quad (8.21)$$

Napon na izlazu iz silaznog pretvarača mijenjamo promjenom vrijednosti registra **OCROA** prema relaciji:

$$\text{OCROA} = \frac{U_i}{U_{DC}} \cdot 255. \quad (8.22)$$

U programskom kodu 8.17 prikazana je konfiguracija tajmera u *Fast* PWM načinu rada.

Programski kod 8.17: Konfiguracija tajmera u *Fast* PWM načinu 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
OCROA = 100;
```

<sup>2</sup>Detalje o proizvoljnom iznosu možete pronaći u literaturi [1] u tablicama 14-1 do 14-4. Preporuka autora jest da se koristi raspon  $[100, 65535]$  kako bi imali dovoljnu rezoluciju promjene širine impulsa PWM signala.



Naredbom `TCCROB |= (0 << CS02) | (1 << CS01) | (0 << CS00)`; frekvenciju radnog takta podijelili smo s 8, prema tablici 8.1. Frekvencija PWM signala prema relaciji 8.16 jest:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256} = \frac{16000000}{8 \cdot 256} = 7812,5 Hz. \quad (8.23)$$

*Fast* PWM način rada tajmera odabran je naredbom `TCCROA |= (1 << WGM01) | (1 << WGM00)`; prema tablici 8.2. Neinvertirajući način rada odabran je naredbom `TCCROA |= (1 << COM0A1) | (0 << COM0A0)`; U tom načinu rada vrijednost registra `OCROA` postavljena je na 100. Ukoliko je ovo konfiguracija PWM signala koji upravlja silaznim pretvaračem koji je prethodno spomenut, izlazni napon za konfiguraciju prikazanu programskim kodom 8.17 može se izračunati prema relaciji (8.21):

$$U_i = \frac{OCROA}{255} \cdot U_{DC} = \frac{100}{255} \cdot 30 = 11.765 V. \quad (8.24)$$

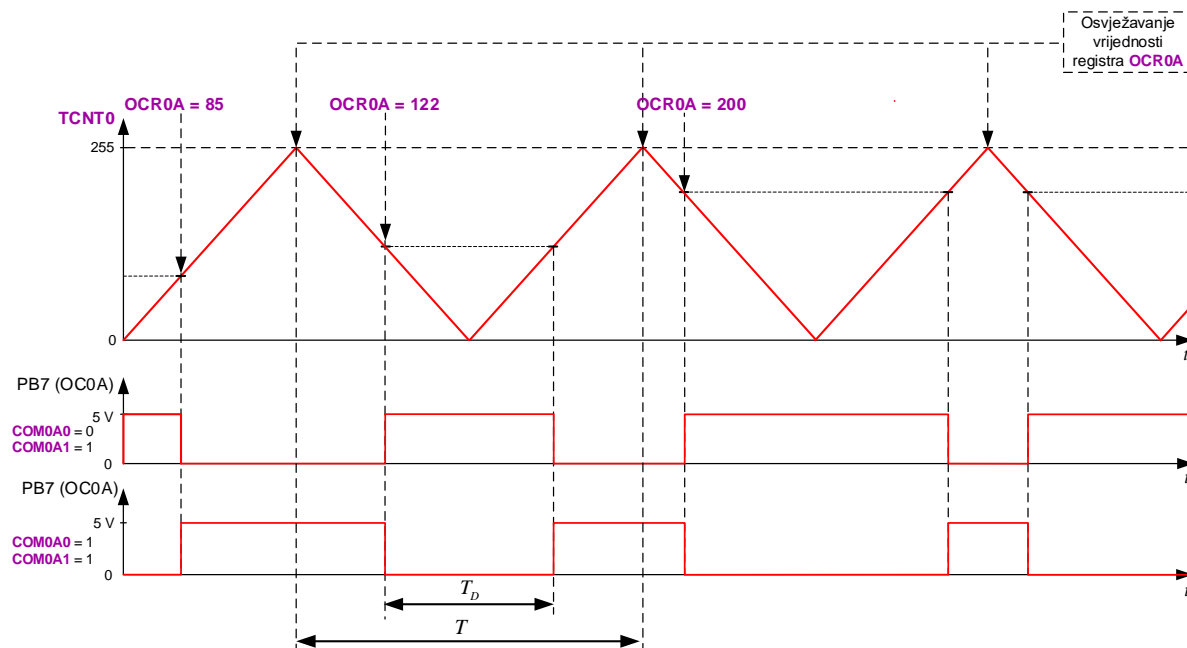
Ako na izlazu želimo postići vrijednost izlaznog napona 25 V, tada je u registar `OCROA` potrebno upisati vrijednost dobivenu relacijom (8.22):

$$OCROA = \frac{U_i}{U_{DC}} \cdot 255 = \frac{25}{30} \cdot 255 = 213. \quad (8.25)$$

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 [1].

### 8.2.2 Phase Correct PWM način rada tajmera

Druga vrsta PWM signala koja se može generirati mikroupravljačem ATmega32U4 jest *Phase Correct* PWM. I ovaj način rada bit će objašnjen na sklopu *Timer/Counter0* za kanal A. Za razliku od *Fast* PWM načina rada, *Phase Correct* PWM način rada ima dvostruko manju frekvenciju. Razlog tome je taj što u *Phase Correct* PWM načinu rada registar `TCNTO` uvećava svoju vrijednost za 1 od 0 (DNO) do 255 (VRH), a kada dođe do vrijednosti 255 počinje smanjivati svoju vrijednost za 1 do 0 i tako uzastopno. Na temelju vrijednosti u registrima `TCNTO` te registrima `OCROA` (kanal A), odnosno `OCROB` (kanal B) generira se pravokutni valni oblik na pinu PB7 (OC0A), odnosno pinu PD0 (OC0B). U registar `OCROA` možemo upisati vrijednosti u rasponu od [0, 255]. Vrijednost u registru `OCROA` osvježava se svaki put kada vrijednost registra `TCNTO` dosegne 255.

Slika 8.6: Vremenski dijagram *Fast PWM* načina rada

Vrijednosti registara  $TCNT0$  i  $OCR0A$  cijelo se vrijeme uspoređuju. U trenutku kada se vrijednosti ovih dvaju registara izjednače ( $TCNT0 == OCR0A$ ), na pinu  $PB7 (OC0A)$  dešava se promjena stanja signala iz visokog u nisko ili obratno, što ovisi o konfiguraciji bitova  $COM0A1$  i  $COM0A0$ . Ovisno o stanju bitova  $COM0A1$  i  $COM0A0$ , prema tablici 8.4 na pinu  $PB7 (OC0A)$  generira se pravokutni valni oblik prikazan na slici 8.6.

Tablica 8.4: Postavke bitova  $COM0A1$  i  $COM0A0$  u *Phase Correct PWM* načina rada (kanal A) [1]

$COM0A1$	$COM0A0$	Postavke <i>Phase Correct PWM</i> načina rada
0	0	Pin $PB7 (OC0A)$ isključen
0	1	Ako je $WGM02 = 0$ , tada je pin $PB7 (OC0A)$ isključen. Ako je $WGM02 = 1$ , promjena stanja pina $PB7 (OC0A)$ događa se kada je vrijednost registra $TCNT0$ jednaka vrijednosti registra $OCR0A$ .
1	0	Pin $PB7 (OC0A)$ postavlja se u nisko stanje kada je vrijednost registra $TCNT0$ jednaka vrijednosti registra $OCR0A$ pri brojanju prema gore. Pin $PB7 (OC0A)$ postavlja se u visoko stanje kada je vrijednost registra $TCNT0$ jednaka vrijednosti registra $OCR0A$ pri brojanju prema dolje.
1	1	Pin $PB7 (OC0A)$ postavlja se u visoko stanje kada je vrijednost registra $TCNT0$ jednaka vrijednosti registra $OCR0A$ pri brojanju prema gore. Pin $PB7 (OC0A)$ 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  $PB7 (OC0A)$  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 8.6).

Ako je konfiguracija bitova  $COM0A1 = 1$  i  $COM0A0 = 1$ , tada se pin  $PB7 (OC0A)$  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 8.6).

PWM signal može se generirati i na pinu PD0 (OC0B) (kanal B) pomoću sklopa *Timer/Counter0*, no tada je potrebno pogledati konfiguraciju bitova **COM0B1** i **COM0B0** u literaturi [1] u tablici 13-6.

Frekvencija PWM signala u *Phase Correct* PWM načinu rada za sklop *Timer/Counter0* jest:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot 2 \cdot TOP} = \frac{F_{CPU}}{PRESCALER \cdot 510}, \quad (8.26)$$

a period PWM signala jest:

$$T = \frac{1}{F_{pcPWM}} = \frac{PRESCALER \cdot 2 \cdot TOP}{F_{CPU}} = \frac{PRESCALER \cdot 510}{F_{CPU}}. \quad (8.27)$$

Frekvencija i period PWM signala ovise samo o djelitelju frekvencije radnog takta, odnosno o varijabli *PRESCALER*. Varijablu *PRESCALER* odabiremo sukladno potrebama sustava na koji dovodimo PWM signal te o brzini elektroničkih komponenata koje u sustav dovode energiju. Vrijednosti koje može poprimiti varijabla *TOP* opisane su u potpoglavlju o *Fast* PWM načinu rada. Širina impulsa neinvertirajućeg PWM signala na kanalu A određena je vremenom visokog stanja  $T_D$  (engl. *duty cycle*) u odnosu na vrijeme perioda  $T$  (slika 8.5) prema relaciji:

$$\frac{T_D}{T} = \frac{OCROA}{255}. \quad (8.28)$$

Širinu PWM signala na pinu PB7 (OC0A) mijenjamo promjenom vrijednosti registra **OCROA** prema relaciji:

$$OCROA = \frac{T_D}{T} \cdot 255. \quad (8.29)$$

Slučajevi u kojima registar **OCROA** poprima vrijednost 0 (DNO) ili 255 (VRH) specijalni su slučajevi. Kod neinvertirajućeg PWM signala (**COM0A1** = 1, **COM0A0** = 0) za specijalne slučajeve vrijedi:

- ako je **OCROA** = 255, tada je pin PB7 (OC0A) konstantno u visokom stanju,
- ako je **OCROA** = 0, tada je pin PB7 (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.

Ako se koristi kanal B (uz **WGM02** = 0), tada širinu PWM signala na pinu PD0 (OC0B) mijenjamo promjenom vrijednosti registra **OCROB** prema relaciji:

$$OCROB = \frac{T_D}{T} \cdot 255. \quad (8.30)$$

Omjer  $\frac{T_D}{T}$  predstavlja postotak aktivnog stanja neinvertirajućeg PWM signala. U programskom kodu 8.18 prikazana je konfiguracija tajmera u *Phase Correct* PWM načinu rada.

Programski kod 8.18: Konfiguracija tajmera u *Phase Correct* PWM načinu 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 0xFF
TCCR0A |= (1 << COM0A1) | (0 << COM0A0); // Neinvertirajući PWM
OCROA = 180;
```

Naredbom `TCCROB`  $|= (0 \ll \text{CS02}) | (1 \ll \text{CS01}) | (0 \ll \text{CS00})$ ; frekvenciju radnog takta podijelili smo s 8, prema tablici 8.1. *Phase Correct* PWM način rada tajmera odabran je naredbom `TCCROA`  $|= (0 \ll \text{WGM01}) | (1 \ll \text{WGM00})$ ; prema tablici 8.2. Neinvertirajući način rada odabran je naredbom `TCCROA`  $|= (1 \ll \text{COM0A1}) | (0 \ll \text{COM0A0})$ ; U tom načinu rada vrijednost registra `OCROA` postavljena je na 180. Srednja vrijednost napona na pinu PB7 (OC0A) za konfiguraciju prikazanu programskim kodom 8.18 može se izračunati prema relaciji (8.21):

$$U_i = \frac{\text{OCROA}}{255} \cdot 5 = \frac{180}{255} \cdot 5 = 3.529V. \quad (8.31)$$

Ako na pinu PB7 (OC0A) želimo postići srednju vrijednost napona iznosa  $U_{PB7} = 1,5$  V, tada je u registar `OCROA` potrebno upisati vrijednost dobivenu sljedećom relacijom (napon napajanja mikroupravljača ATmega32U4 iznosi  $U_{VCC} = 5$  V):

$$\text{OCROA} = \frac{U_{PB7}}{U_{VCC}} \cdot 255 = \frac{1,5}{5} \cdot 255 = 77. \quad (8.32)$$

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 [1].

### 8.2.3 Konfiguracija PWM načina rada tajmera pomoću zaglavlja "timer.h"

Autor je pripremio brojne funkcije koje omogućuju jednostavnu konfiguraciju PWM načina rada tajmera pomoću zaglavlja "timer.h" koje se nalazi u mapi "Timer". U nastavku je popis funkcija koje se koriste za generiranje PWM signala s opisom:

- `timer0_set_fast_PWM()` - podešavanje *Fast* PWM načina rada za sklop *Timer/Counter0*
- `timer0_set_phase_correct_PWM()` - podešavanje *Phase Correct* PWM načina rada za sklop *Timer/Counter0*,
- `timer0_OC0A_enable_non_inverted_PWM()` - omogućavanje generiranja neinvertirajućeg PWM signala na kanalu A (OC0A) za sklop *Timer/Counter0*,
- `timer0_OC0A_enable_inverted_PWM()` - omogućavanje generiranja invertirajućeg PWM signala na kanalu A (OC0A) za sklop *Timer/Counter0*,
- `timer0_OC0A_disable()` - onemogućavanje generiranja PWM signala na kanalu A (OC0A) za sklop *Timer/Counter0*,
- `OC0A_set_duty_cycle(float duty)` - postavljanje širine impulsa na kanalu A (OC0A) za sklop *Timer/Counter0*. Funkcija prima argument `duty` u rasponu [0, 100.0] koja predstavlja širinu impulsa PWM signala.
- `timer0_OC0B_enable_non_inverted_PWM()` - omogućavanje generiranja neinvertirajućeg PWM signala na kanalu B (OC0B) za sklop *Timer/Counter0*,
- `timer0_OC0B_enable_inverted_PWM()` - omogućavanje generiranja invertirajućeg PWM signala na kanalu B (OC0B) za sklop *Timer/Counter0*,
- `timer0_OC0B_disable()` - onemogućavanje generiranja PWM signala na kanalu B (OC0B) za sklop *Timer/Counter0*,

- `OC0B_set_duty_cycle(float duty)` - postavljanje širine impulsa na kanalu B (OC0B) za sklop *Timer/Counter0*. Funkcija prima argument `duty` u rasponu [0, 100.0] koja predstavlja širinu impulsa PWM signala.
- `timer1_set_fast_PWM_8bit()` - podešavanje *Fast* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 255 (brojanje u rasponu [0, 255], konstanta za određivanje raspona brojanja je `PWM_8BIT`),
- `timer1_set_fast_PWM_9bit()` - podešavanje *Fast* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 511 (brojanje u rasponu [0, 511], konstanta za određivanje raspona brojanja je `PWM_9BIT`),
- `timer1_set_fast_PWM_10bit()` - podešavanje *Fast* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 1023 (brojanje u rasponu [0, 1023], konstanta za određivanje raspona brojanja je `PWM_10BIT`),
- `timer1_set_fast_PWM_ICR1(uint16_t top)` - podešavanje *Fast* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću `top` (brojanje u rasponu [0, `top`], konstanta za određivanje raspona brojanja je `PWM_ICR1`). Vršna vrijednost `top` zapisuje se u registar `ICR1` i može poprimiti vrijednosti u rasponu [0, 65535].
- `timer1_set_phase_correct_PWM_8bit()` - podešavanje *Phase Correct* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 255 (brojanje u rasponu [0, 255], konstanta za određivanje raspona brojanja je `PWM_8BIT`),
- `timer1_set_phase_correct_PWM_9bit()` - podešavanje *Phase Correct* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 511 (brojanje u rasponu [0, 511], konstanta za određivanje raspona brojanja je `PWM_9BIT`),
- `timer1_set_phase_correct_PWM_10bit()` - podešavanje *Phase Correct* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću 1023 (brojanje u rasponu [0, 1023], konstanta za određivanje raspona brojanja je `PWM_10BIT`),
- `timer1_set_phase_correct_PWM_ICR1(uint16_t top)` - podešavanje *Phase Correct* PWM načina rada za sklop *Timer/Counter1* s vršnom vrijednošću `top` (brojanje u rasponu [0, `top`], konstanta za određivanje raspona brojanja je `PWM_ICR1`). Vršna vrijednost `top` zapisuje se u registar `ICR1` i može poprimiti vrijednosti u rasponu [0, 65535].
- `timer1_OC1A_enable_non_inverted_PWM()` - omogućavanje generiranja neinvertirajućeg PWM signala na kanalu A (OC1A) za sklop *Timer/Counter1*,
- `timer1_OC1A_enable_inverted_PWM()` - omogućavanje generiranja invertirajućeg PWM signala na kanalu A (OC1A) za sklop *Timer/Counter1*,
- `timer1_OC1A_disable()` - onemogućavanje generiranja PWM signala na kanalu A (OC1A) za sklop *Timer/Counter1*,
- `OC1A_set_duty_cycle(float duty, uint16_t top)` - postavljanje širine impulsa na kanalu A (OC1A) za sklop *Timer/Counter1*. Funkcija prima argument `duty` u rasponu [0, 100.0] koji predstavlja širinu impulsa PWM signala te argument `top` koji može poprimiti definirane konstante (vidi programski kod 8.19): `PWM_8BIT`, `PWM_9BIT`, `PWM_10BIT`, `PWM_ICR1`.
- `timer1_OC1B_enable_non_inverted_PWM()` - omogućavanje generiranja neinvertirajućeg PWM signala na kanalu B (OC1B) za sklop *Timer/Counter1*,

- `timer1_OC1B_enable_inverted_PWM()` - omogućavanje generiranja invertirajućeg PWM signala na kanalu B (OC1B) za sklop *Timer/Counter1*,
- `timer1_OC1B_disable()` - onemogućavanje generiranja PWM signala na kanalu B (OC1B) za sklop *Timer/Counter1*,
- `OC1B_set_duty_cycle(float duty, uint16_t top)` - postavljanje širine impulsa na kanalu B (OC1B) za sklop *Timer/Counter1*. Funkcija prima argument `duty` u rasponu [0, 100.0] koji predstavlja širinu impulsa PWM signala te argument `top` koji može poprimiti definirane konstante (vidi programski kod 8.19): `PWM_8BIT`, `PWM_9BIT`, `PWM_10BIT`, `PWM_ICR1`.
- `timer1_OC1C_enable_non_inverted_PWM()` - omogućavanje generiranja neinvertirajućeg PWM signala na kanalu C (OC1C) za sklop *Timer/Counter1*,
- `timer1_OC1C_enable_inverted_PWM()` - omogućavanje generiranja invertirajućeg PWM signala na kanalu B (OC1C) za sklop *Timer/Counter1*,
- `timer1_OC1C_disable()` - onemogućavanje generiranja PWM signala na kanalu C (OC1C) za sklop *Timer/Counter1*,
- `OC1C_set_duty_cycle(float duty, uint16_t top)` - postavljanje širine impulsa na kanalu C (OC1C) za sklop *Timer/Counter1*. Funkcija prima argument `duty` u rasponu [0, 100.0] koji predstavlja širinu impulsa PWM signala te argument `top` koji može poprimiti definirane konstante (vidi programski kod 8.19): `PWM_8BIT`, `PWM_9BIT`, `PWM_10BIT`, `PWM_ICR1`.

Programski kod 8.19: Definiranje konstanti koje predstavljaju vršne vrijednosti raspona brojanja

```
//TOP vrijednost za PWM signale
#define PWM_8BIT 255
#define PWM_9BIT 511
#define PWM_10BIT 1023
#define PWM_ICR1 ICR1
```

U sljedećim vježbama bit će prikazan niz funkcija koje su napisane za konfiguriranje PWM načina rada, a koje se nalaze u biblioteci `timer.h`.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) 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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se `Ivica Ivic`. Datoteku `PWM.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atsln` u datoteci `\\PWM\vjezbe`. U otvorenom projektu nalaze se sve 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 8.2.1

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Svakih 1000 ms u `while` petlji mijenjajte širinu impulsa PWM signala, tako da ona iznosi redom 10 % → 40 % → 90 % → 10 % perioda  $T$ . Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Fast PWM* načina rada sklopa *Timer/Counter0*.

U projektnom stablu otvorite datoteku `vjezba821.cpp`. Omogućite samo prevođenje datoteke `vjezba821.cpp`. Početni sadržaj datoteke `vjezba821.cpp` prikazan je programskim kodom 8.20.

Programski kod 8.20: Početni sadržaj datoteke `vjezba821.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"

void inicijalizacija() {
    // inicijalizacija za timer 0 - neinvertirajući Fast PWM
    timer0_set_prescaler(TIMER0_PRESCALER_8);
    timer0_set_fast_PWM();
    timer0_OC0A_enable_non_inverted_PWM();

    output_port(DDRB, PB7);
}

int main(void) {

    inicijalizacija();

    while(1) {
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
    }
    return 0;
}
```

U ovoj vježbi sklop *Timer/Counter0* koristit ćemo u *Fast PWM* načinu rada. PWM signal generirat ćemo na kanalu A, odnosno na pinu PB7 (OC0A). Frekvencija PWM signala mora biti veća od 3 kHz, a manja od 10 kHz. Prema relaciji (8.16), frekvencija PWM signala ovisi o frekvenciji radnog takta  $F_{CPU}$  i o djelitelju frekvencije radnog takta. Frekvencija radnog takta jest 16 MHz pa je potrebno odabrati djelitelj frekvencije radnog takta za koji će vrijediti da frekvencija bude između 3 i 10 kHz. Za djelitelj frekvencije radnog takta odabrat ćemo 8. Prema relaciji (8.16), frekvencija PWM signala bit će:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256} = \frac{16000000}{8 \cdot 256} = 7812,5 \text{ Hz} = 7.81 \text{ kHz}. \quad (8.33)$$

Dobivena frekvencija PWM signala u relaciji (8.39) veća je od 3 kHz i manja od 10 kHz. Pokušajte u relaciju (8.16) uvrstiti ostale vrijednosti djelitelj frekvencije radnog takta kako bismo ustanovili sve moguće frekvencije PWM signala pomoću sklopa *Timer/Counter0*. U programskom kodu 8.20 u funkciji `inicijalizacija()` djelitelj frekvencije radnog takta

iznosa 8 konfiguriran je funkcijom `timer0_set_prescaler(TIMERO_PRESCALER_8)`. *Fast* PWM način rada što postavlja se pozivom funkcije `timer0_set_fast_PWM()`. PWM signal mora biti neinvertirajući, a to se postiže pozivom funkcije `timer0_OCOA_enable_non_inverted_PWM()`. Obje prethodne funkcije pozvane su u funkciji `inicijalizacija()`. Sve korištene funkcije za konfiguriranje *Fast* PWM načina rada nalaze se u zaglavlju `timer.h`, stoga je ovo zaglavlje potrebno uključiti u programski kod. PWM signal generira se na pinu PB7 (OC0A) pa ga je potrebno konfigurirati kao izlazni pin. Ukoliko zaboravite pin PB7 konfigurirati kao izlazni, na njemu se neće generirati PWM signal.

Nakon konfiguracije neinvertirajućeg PWM signala potrebno je odrediti vrijednosti registra `OCROA` za koju će širina impulsa PWM signala biti 10 %, 40 % i 90 % perioda  $T$ . Prema relaciji (8.20) vrijednost `OCROA` registra za širinu PWM signala 10 % perioda  $T$  bit će:

$$\text{OCROA} = \frac{T_D}{T} \cdot 255 = 0,1 \cdot 255 = 26. \quad (8.34)$$

Prema relaciji (8.20) vrijednost `OCROA` registra za širinu PWM signala 40 % perioda  $T$  bit će:

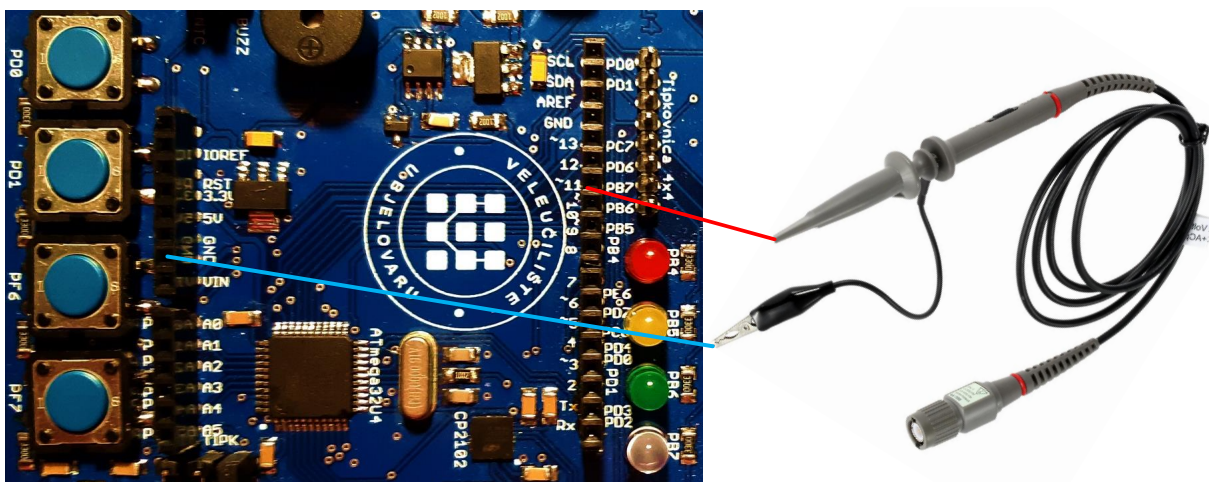
$$\text{OCROA} = \frac{T_D}{T} \cdot 255 = 0,4 \cdot 255 = 102. \quad (8.35)$$

Prema relaciji (8.20) vrijednost `OCROA` registra za širinu PWM signala 90 % perioda  $T$  bit će:

$$\text{OCROA} = \frac{T_D}{T} \cdot 255 = 0,9 \cdot 255 = 230. \quad (8.36)$$

U programski kod 8.20 upišite izračunate vrijednosti registra `OCROA` u `while` petlju ispred svakog kašnjenja iznosa 1000 ms. Ako mikroupravljač koristite samo za generiranje PWM signala, tada u funkciji `main` obavezno morate napraviti beskonačnu `while` petlju. Kada ne biste napravili beskonačnu petlju, mikroupravljač ne bi generirao PWM signal jer bi završio s izvođenjem `main` funkcije.

Prevedite datoteku `vjezba821.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. PWM signal pratit ćemo pomoću osciloskopa. Spajanje sonde osciloskopa na razvojno okruženje s mikroupravljačem ATmega32U4 prikazano je na slici 8.7.



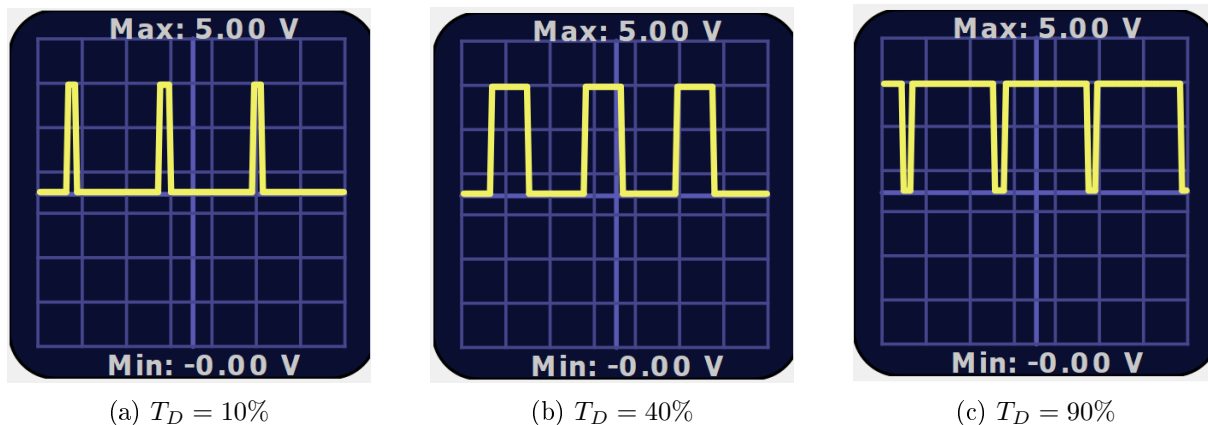
Slika 8.7: Spajanje sonde osciloskopa na razvojno okruženje s mikroupravljačem ATmega32U4

Na razvojnom okruženju s mikroupravljačem ATmega32U4 pinovi su izvučeni na ravne priključnice na kojima su označena imena pinova. Na slici 8.7 prikazane su dvije ravne



priključnice. Sondu osciloskopa potrebno je spojiti između dvaju spojnih mjesta na ravnim priključnicama: GND i PB7 (slika 8.7). Na osciloskopu prikažite tri perioda PWM signala te frekvenciju PWM signala. Prikaz širine impulsa na pinu PB7 (OC0A) pomoću osciloskopa prikazan je na slici 8.8.

Širina impulsa može se definirati korištenjem funkcije `OC0A_set_duty_cycle(float duty)` koja kao argument prima širinu impulsa PWM signala u rasponu od 0 % do 100 %. Na primjer, ukoliko želite na pinu PB7 (OC0A) postaviti širinu impulsa iznosa 40 %, dovoljno je pozvati funkciju `OC0A_set_duty_cycle(40.0)`. Zamijenite sva pridruživanja vrijednosti u registar `OCROA` u `while` petlju s navedenom funkcijom.



Slika 8.8: Prikaz širine impulsa na pinu PB7 (OC0A) pomoću osciloskopa

U programski kod 8.20 funkciju `timer0_OC0A_enable_non_inverted_PWM()` zamijenite s funkcijom `timer0_OC0A_enable_inverted_PWM()`. Prevedite datoteku `vjezba821.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Komentirajte ponašanje LED diode spojene na pin PB7 (OC0A) u odnosu na prethodni programski kod. Pogledajte sada PWM signal generiran na pinu PB7 (OC0A) pomoću osciloskopa.

U programski kod 8.20 tijelo funkcije `inicijalizacija()` zamijenite programskim kodom 8.21. Ovaj pristup složeniji je od korištenja gotovih funkcija iz zaglavlja `timer.h` jer zahtijeva podešenje registara prema tablicama 8.2 i 8.3. Prevedite datoteku `vjezba821.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Programski kod 8.21: Konfiguracija tajmera u *Fast* PWM načinu rada (djelitelj frekvencije radnog takta iznosi 8, PWM signal je neinvertirajući)

```
TCCR0B |= (0 << CS02) | (1 << CS01) | (0 << CS00); //F_CPU/8
TCCR0A |= (1 << WGM01) | (1 << WGM00); // Fast PWM način rada
TCCR0B |= (0 << WGM02); // VRH je 0xFF
TCCR0A |= (1 << COM0A1) | (0 << COM0A0); // Neinvertirajući PWM
DDRB |= (1 << PB7); // PB7 izlazni pin
```

Zatvorite datoteku `vjezba821.cpp` i onemogućite prevođenje ove datoteke.



## Vježba 8.2.2

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Fast PWM* načina rada sklopa *Timer/Counter0*.

U projektnom stablu otvorite datoteku `vjezba822.cpp`. Omogućite samo prevođenje datoteke `vjezba822.cpp`. Početni sadržaj datoteke `vjezba822.cpp` prikazan je programskim kodom 8.22.

Programski kod 8.22: Početni sadržaj datoteke `vjezba822.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija() {

    // inicijalizacija za timer 0 - neinvertirajući Fast PWM

    lcd_init();
    adc_init();
}

int main(void) {

    inicijalizacija();

    float duty; // širina impulsa
    uint16_t ADC_5; // vrijednost AD pretvorbe na pinu ADC5

    while(1) {

        ADC_5 = adc_read(ADC5); // ADC na kanalu ADC5
        duty = 0;
        OC0A_set_duty_cycle(duty);

        lcd_clrscr();
        lcd_home();
        lcd_print("PWM freq:%luHz\n", F_CPU / 8 / 256);
        lcd_print("TD/T:%.2f%%", duty);
        _delay_ms(500);

    }
    return 0;
}
```

U programskom kodu 8.22 u funkciji `inicijalizacija()` napravite konfiguraciju sklopa *Timer/Counter0* u *Fast PWM* načinu rada. Generirani PWM signal mora biti neinvertirajući te frekvencije veće od 3 kHz, a manje od 10 kHz (pogledajte konfiguraciju u prethodnoj vježbi). Širinu PWM signala potrebno je mijenjati pomoću potenciometra koji je spojen na pin ADC5. Zbog toga je u funkciji `inicijalizacija()` pozvana funkcija `adc_init()` kojom se konfigurira

analogno-digitalna pretvorba. Širinu PWM signala mijenjat ćemo pomoću registra `OCROA`, no koristeći funkciju `OC0A_set_duty_cycle`. Vrijednost analogno-digitalne pretvorbe na pinu ADC5 kreće se od  $[0, 1023]$ . Prema tome, 0 će predstavljati 0 % popunjenosti PWM signala, a 1023 će predstavljati 100 % popunjenosti PWM signala. Omjer  $\frac{T_D}{T}$  [%] izračunat ćemo pomoću vrijednosti analogno-digitalne pretvorbe na pinu ADC5 na sljedeći način:

$$duty = \frac{T_D}{T} [\%] = \frac{ADC = 5}{1023} \cdot 100\%. \quad (8.37)$$

Izraz (8.37) potrebno je pridružiti varijabli `duty` u programskom kodu 8.22. Širina impulsa mijenja se u `while` petlji svakih 500 ms. Na LCD displeju prikazuje se frekvencija PWM signala izračunata prema relaciji (8.16) te postotak popunjenosti PWM signala prema relaciji (8.37).

Prevedite datoteku `vjezba822.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Na slici 8.7 prikazane su dvije ravne priključnice. Sondu osciloskopa potrebno je spojiti između dvaju spojnih mjesta na ravnim priključnicama: GND i PB7 (slika 8.7). Na osciloskopu prikažite tri perioda PWM signala te frekvenciju PWM signala. Mijenjajte izlazni otpor potencijometra pomoću odvijača i pratite na osciloskopu popunjenost PWM signala. Što zaključujete? Primijetite da pri širini impulsa PWM signala iznosa 0 % (`duty = 0`) LED dioda sjaji jako malim intezitetom. Zašto?

Zatvorite datoteku `vjezba822.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.2.3

Napravite program koji će:

- na pinu PB5 (OC1A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter1*,
- na pinu PB6 (OC1B) generirati invertirajući PWM signal pomoću sklopa *Timer/Counter1*,
- na pinu PB7 (OC1C) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter1*.

Frekvenciju PWM signala namjestite tako da bude manja od 3 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potencijometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Fast PWM* načina rada sklopa *Timer/Counter1*.

U projektnom stablu otvorite datoteku `vjezba823.cpp`. Omogućite samo prevođenje datoteke `vjezba823.cpp`. Početni sadržaj datoteke `vjezba823.cpp` prikazan je programskim kodom 8.23.

Programski kod 8.23: Početni sadržaj datoteke `vjezba823.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija() {
    // inicijalizacija za timer 1 - Fast PWM
```

```

timer1_set_prescaler(TIMER1_PRESCALER_64);
timer1_set_fast_PWM_8bit();
timer1_OC1A_enable_non_inverted_PWM();
timer1_OC1B_enable_inverted_PWM();
timer1_OC1C_enable_non_inverted_PWM();

DDRB |= (1 << PB5) | (1 << PB6) | (1 << PB7);

adc_init(); // inicijalizacija AD pretvorbe
lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {

    inicijalizacija();

    float duty;

    while(1) {

        duty = adc_read(ADC5)*100.0/1023.0;
        OC1A_set_duty_cycle(duty, PWM_8BIT);
        OC1B_set_duty_cycle(duty, PWM_8BIT);
        OC1C_set_duty_cycle(duty, PWM_8BIT);

        lcd_clrscr();
        lcd_home();
        lcd_print("PWM freq:%luHz\n", F_CPU / 64 / 256);
        lcd_print("TD/T:%.2f%%", duty);
        _delay_ms(500);
    }
    return 0;
}

```

U ovoj vježbi sklop *Timer/Counter1* koristit ćemo u *Fast PWM* načinu rada. Sklop *Timer/Counter1* može generirati PWM signal na trima kanalima: kanalu (PB5 (OC1A)), kanal B (PB6 (OC1B)) i kanal C (PB7 (OC1C)). U ovoj vježbi generirat ćemo PWM signal na svim trima kanalima. Frekvencija PWM signala mora biti manja od 3 kHz. Prema relaciji (8.16), frekvencija PWM signala ovisi o frekvenciji radnog takta  $F_{CPU}$  i o djelitelju frekvencije radnog takta. Frekvencija radnog takta je 16 MHz pa je potrebno odabrati djelitelj frekvencije radnog takta za koji će vrijediti da frekvencija bude manja od 3 kHz. Za djelitelj frekvencije radnog takta odabrat ćemo 64. Prema relaciji (8.16), frekvencija PWM signala bit će:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256} = \frac{16000000}{64 \cdot 256} = 976,56 \text{ Hz.} \quad (8.38)$$

Konfiguracija sklopa *Timer/Counter1* provodi se sukladno tablicama 14-1 do 14-5 u literaturi [1] ili korištenjem brojnih funkcija napisanih u zaglavlju `timer.h`. U programskom kodu 8.23 u funkciji `inicijalizacija()` napravljena je konfiguraciju sklopa *Timer/Counter1* u *Fast PWM* načinu rada funkcijom `timer1_set_fast_PWM_8bit()`. Ovom funkcijom definiran je raspon brojanja registra `TCNT1` ([0, 255]).

Neinvertirajući ili invertirajući PWM signal konfigurira se sljedećim funkcijama u funkciji `inicijalizacija()`:

- `timer1_OC1A_enable_non_inverted_PWM()` - na pinu PB5 (OC1A) generirat će se neinvertirajući PWM signal,
- `timer1_OC1B_enable_inverted_PWM()` - na pinu PB6 (OC1B) generirat će se invertirajući PWM signal,

- `timer1_OC1B_enable_non_inverted_PWM()` - na pinu PB7 (OC1C) generirat će se neinvertirajući PWM signal.

PWM signal generira se na pinovima PB5 (OC1A), PB6 (OC1B) i PB7 (OC1C) pa ih je potrebno konfigurirati kao izlazne pinove. Širinu PWM signala potrebno je mijenjati pomoću potencijometra koji je spojen na pin ADC5, a računat ćemo je pomoću relacije (8.37) iz prethodne vježbe. Izraz (8.37) potrebno je pridružiti varijabli `duty` u programskom kodu 8.23. Promjena širine PWM signala za sva tri kanala provodi se funkcijama:

- `OC1A_set_duty_cycle(duty, PWM_8BIT)`,
- `OC1B_set_duty_cycle(duty, PWM_8BIT)` i
- `OC1C_set_duty_cycle(duty, PWM_8BIT)`.

Prvi argument navedenih funkcija jest širina impulsa `duty` koji je proračunat prema relaciji (8.37). Prema programskom kodu 8.19, konstanta kojom se definira raspon brojanja registra `TCNT1` za 8-bitni *Fast* PWM jest `PWM_8BIT`. Ova konstanta dodjeljuje se kao drugi argument prethodno navedenih funkcija. Širina impulsa mijenja se u `while` petlji svakih 500 ms. Na LCD displeju prikazuje se frekvencija PWM signala izračunata prema relaciji (8.16) te postotak popunjenosti PWM signala.

Prevedite datoteku `vjezba823.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Žuta i plava LED dioda mijenjaju intezitet na drugačiji način od zelene LED diode. Zašto?

Zatvorite datoteku `vjezba823.cpp` i onemogućite prevođenje ove datoteke.



#### Vježba 8.2.4

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Svakih 500 ms u `while` petlji mijenjajte širinu impulsa PWM signala, tako da ona iznosi redom 0 % → 25 % → 50 % → 75 % → 100 % → 0 % perioda  $T$ . Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Phase Correct* PWM načina rada sklopa *Timer/Counter0*.

U projektnom stablu otvorite datoteku `vjezba824.cpp`. Omogućite samo prevođenje datoteke `vjezba824.cpp`. Početni sadržaj datoteke `vjezba824.cpp` prikazan je programskim kodom 8.24.

Programski kod 8.24: Početni sadržaj datoteke `vjezba824.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"

void inicijalizacija() {
    // inicijalizacija za timer 0 - neinvertirajući Phase Correct PWM
    timer0_set_prescaler(TIMER0_PRESCALER_8);
    timer0_set_phase_correct_PWM();
    timer0_OC0A_enable_non_inverted_PWM();

    output_port(DDRB, PB7);
}
```

```

int main(void) {
    inicijalizacija();

    while(1) {
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
        OCROA = 0;
        _delay_ms(1000);
    }
    return 0;
}

```

U ovoj vježbi sklop *Timer/Counter0* koristit ćemo u *Phase Correct* PWM načinu rada. PWM signal generirat ćemo na kanalu A, odnosno na pinu PB7 (OC0A). Frekvencija radnog takta je 16 MHz pa je potrebno odabrati djelitelj frekvencije radnog takta za koji će vrijediti da frekvencija bude između 3 i 10 kHz. Za djelitelj frekvencije radnog takta odabrat ćemo 8. Prema relaciji (8.26), frekvencija PWM signala bit će:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot 510} = \frac{16000000}{8 \cdot 510} = 3921,57 \text{ Hz} = 3.92 \text{ kHz}. \quad (8.39)$$

Dobivena frekvencija PWM signala u relaciji (8.39) veća je od 3 kHz i manja od 10 kHz. U programskom kodu 8.24 u funkciji `inicijalizacija()` djelitelj frekvencije radnog takta iznosa 8 konfiguriran je funkcijom `timer0_set_prescaler(TIMER0_PRESCALER_8)`. *Phase Correct* PWM način rada postavlja se pozivom funkcije `timer0_set_phase_correct_PWM()`.

PWM signal mora biti neinvertirajući, a to se postiže pozivom funkcije `timer0_OCOA_enable_non_inverted_PWM()`. Obje prethodne funkcije pozvane su u funkciji `inicijalizacija()`. Sve korištene funkcije za konfiguriranje *Phase Correct* PWM načina rada nalaze se u zaglavlju `timer.h`, stoga je ovo zaglavlje potrebno uključiti u programski kod. PWM signal generira se na pinu PB7 (OC0A) pa ga je potrebno konfigurirati kao izlazni pin.

Nakon konfiguracije neinvertirajućeg PWM signala potrebno je odrediti vrijednosti registra `OCROA` za koju će širina impulsa PWM signala biti 0 %, 25 %, 50 %, 75 % i 100 % perioda  $T$ .

Prema relaciji (8.20), vrijednost `OCROA` registra za širinu PWM signala 0 % perioda  $T$  bit će:

$$OCROA = \frac{T_D}{T} \cdot 255 = 0 \cdot 255 = 0. \quad (8.40)$$

Prema relaciji (8.20), vrijednost `OCROA` registra za širinu PWM signala 25 % perioda  $T$  bit će:

$$OCROA = \frac{T_D}{T} \cdot 255 = 0,25 \cdot 255 = 64. \quad (8.41)$$

Prema relaciji (8.20), vrijednost `OCROA` registra za širinu PWM signala 50 % perioda  $T$  bit će:

$$OCROA = \frac{T_D}{T} \cdot 255 = 0,5 \cdot 255 = 128. \quad (8.42)$$

Prema relaciji (8.20), vrijednost `OCROA` registra za širinu PWM signala 75 % perioda  $T$  bit će:

$$OCROA = \frac{T_D}{T} \cdot 255 = 0,75 \cdot 255 = 191. \quad (8.43)$$

Prema relaciji (8.20), vrijednost `OCROA` registra za širinu PWM signala 100 % perioda  $T$  bit će:

$$\text{OCROA} = \frac{T_D}{T} \cdot 255 = 1 \cdot 255 = 255. \quad (8.44)$$

U programski kod 8.24 upišite izračunate vrijednosti registra `OCROA` u `while` petlju ispred svakog kašnjenja iznosa 1000 ms. Ako mikroupravljač koristite samo za generiranje PWM signala, tada u funkciji `main` obavezno morate napraviti beskonačnu `while` petlju. Kada ne biste napravili beskonačnu petlju, mikroupravljač ne bi generirao PWM signal jer bi završio s izvođenjem `main` funkcije.

Prevedite datoteku `vjezba824.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. PWM signal pratit ćemo pomoću osciloskopa. Spajanje sonde osciloskopa na razvojno okruženje s mikroupravljačem ATmega32U4 prikazano je na slici 8.7. Na osciloskopu prikažite tri perioda PWM signala te frekvenciju PWM signala prema uputama u prvoj vježbi.

Širina impulsa može se definirati korištenjem funkcije `OCOA_set_duty_cycle(float duty)` koja kao argument prima širinu impulsa PWM signala u rasponu od 0 % do 100 %. Na primjer, ukoliko želite na pinu PB7 (OC0A) postaviti širinu impulsa iznosa 25 %, dovoljno je pozvati funkciju `OCOA_set_duty_cycle(25.0)`. Zamijenite sva pridruživanja vrijednosti u registar `OCROA` u `while` petlju s navedenom funkcijom.

U programskom kodu 8.24 tijelo funkcije `inicijalizacija()` zamijenite programskim kodom 8.25. Ovaj pristup složeniji je od korištenja gotovih funkcija iz zaglavlja `timer.h` jer zahtijeva podešenje registara prema tablicama 8.2 i 8.3. Prevedite datoteku `vjezba824.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Programski kod 8.25: Konfiguracija tajmera u *Phase Correct* PWM načinu rada (djelitelj frekvencije radnog takta iznosi 8, PWM signal je neinvertirajući)

```
//inicijalizacija za timer 0-neinvertirajući Phase Correct PWM (drugi način)
TCCR0B |= (0 << CS02) | (1 << CS01) | (0 << CS00); //F_CPU/8
TCCR0A |= (0 << WGM01) | (1 << WGM00); // Phase Corect PWM način rada
TCCR0B |= (0 << WGM02); // VRH je 0xFF
TCCR0A |= (1 << COM0A1) | (0 << COM0A0); // Neinvertirajući PWM

DDRB |= (1 << PB7); // PB7 izlazni pin
```

Zatvorite datoteku `vjezba824.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.2.5

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0* samo ako nije pritisnuto tipkalo spojeno na pin PD0. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potencijometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Phase Correct* PWM načina rada sklopa *Timer/Counter0*.

U projektnom stablu otvorite datoteku `vjezba825.cpp`. Omogućite samo prevođenje datoteke `vjezba825.cpp`. Početni sadržaj datoteke `vjezba825.cpp` prikazan je programskim kodom 8.26.

Programski kod 8.26: Početni sadržaj datoteke `vjezba825.cpp`

```

#include "AVR VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija() {

    // inicijalizacija za timer 0 - neinvertirajući Phase Correct PWM

    input_port(DDRD,PDO); //PDO ulazni pin
    set_port(PORTD , PDO,1); // pull up otpornik na PDO
    lcd_init();
    adc_init();
}

int main(void) {

    inicijalizacija();

    float duty; // širina impulsa
    uint16_t ADC_5; // vrijednost AD pretvorbe na pinu ADC5

    while(1) {

        if (get_pin(PIND, PDO) == 0) {
            timer0_OCOA_disable();
        }
        else {
            timer0_OCOA_enable_non_inverted_PWM();
        }

        ADC_5 = adc_read(ADC5); // ADC na kanalu ADC5
        duty = ADC_5 *100.0/1023.0;
        OCOA_set_duty_cycle(duty);

        lcd_clrscr();
        lcd_home();
        lcd_print("PWM freq:%luHz\n", F_CPU / 8 / 510);
        lcd_print("TD/T:%.2f%%", duty);
        _delay_ms(500);
    }
    return 0;
}

```

U programskom kodu 8.26 u funkciji `inicijalizacija()` napravite konfiguraciju sklopa *Timer/Counter0* u *Phase Correct PWM* načinu rada. Generirani PWM signal mora biti neinvertirajući te frekvencije veće od 3 kHz, a manje od 10 kHz (pogledajte konfiguraciju u prethodnim vježbama). Širinu PWM signala potrebno je mijenjati pomoću potenciometra koji je spojen na pin ADC5. Zbog toga je u funkciji `inicijalizacija()` pozvana funkcija `adc_init()` kojom se konfigurira analogno-digitalna pretvorba.

PWM signal potrebno je generirati ako tipkalo spojeno na pin PD0 nije pritisnuto. Ova funkcionalnost realizirana je na početku `while` petlje u programskom kodu 8.26. Ako je tipkalo PD0 pritisnuto (vrijedi `get_pin(PIND, PDO)== 0`), tada se poziva funkcija `timer0_OCOA_disable()` kojom se isključuje generiranje PWM signala na pin PB7 (OC0A). Kada tipkalo nije pritisnuto, tada se poziva funkcija `timer0_OCOA_enable_non_inverted_PWM()` kojom se omogućuje generiranje PWM signala na pin PB7 (OC0A).



Širinu PWM signala potrebno je mijenjati pomoću potenciometra koji je spojen na pin ADC5, a računat ćemo ju pomoću relacije (8.37). Izraz (8.37) potrebno je pridružiti varijabli `duty` u programskom kodu 8.26. Širina impulsa mijenja se u `while` petlji svakih 500 ms. Na LCD displeju prikazuje se frekvencija PWM signala izračunata prema relaciji (8.16) te postotak popunjenosti PWM signala prema relaciji (8.37).

Prevedite datoteku `vjezba825.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Mijenjajte izlazni otpor potenciometra pomoću odvijača. Primijetite da je pri širini impulsa PWM signala iznosa 0 % (`duty = 0`) LED dioda isključena. Za razliku od *Fast PWM* načina rada gdje se pri širini impulsa iznosa 0 % pojavljuje šiljak, kod *Phase Correct PWM* načina rada toga šiljka nema, što omogućuje isključenje uređaja u potpunosti.

Zatvorite datoteku `vjezba825.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 8.2.6

Napravite program koji će na pinu PB5 (OC1A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter1*. Frekvenciju PWM signala namjestite tako da djelitelj frekvencije radnog takta iznosi 64. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Phase Correct PWM* načina rada sklopa *Timer/Counter1* u 8-bitnom, 9-bitnom, 10-bitnom te punom rasponu brojanja registra `TCNT1`.

U projektnom stablu otvorite datoteku `vjezba826.cpp`. Omogućite samo prevođenje datoteke `vjezba826.cpp`. Početni sadržaj datoteke `vjezba826.cpp` prikazan je programskim kodom 8.27.

Programski kod 8.27: Početni sadržaj datoteke `vjezba826.cpp`

```
#include "AVR_VUB/avrvub.h"
#include <avr/io.h>
#include <util/delay.h>
#include "Timer/timer.h"
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija() {
    // inicijalizacija za timer 1 - Phase Correct PWM - [0, 255]
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_set_phase_correct_PWM_8bit();
    timer1_OC1A_enable_non_inverted_PWM();

    DDRB |= (1 << PB5); //PB5 izlazni pin
    adc_init(); // inicijalizacija AD pretvorbe
    lcd_init(); // inicijalizacija LCD displeja
}

int main(void) {

    inicijalizacija();
    uint16_t pwm_bit;
    float duty;

    while(1) {

        pwm_bit = PWM_8BIT;
        duty = adc_read(ADC5)*100.0/1023.0;
```

```

    OC1A_set_duty_cycle(duty, pwm_bit);

    lcd_clrscr();
    lcd_home();
    lcd_print("PWM freq:%luHz\n", F_CPU / 64 / (2 * pwm_bit));
    lcd_print("TD/T:%.2f%%", duty);
    _delay_ms(500);
}
return 0;
}

```

U ovoj vježbi sklop *Timer/Counter1* koristit ćemo u *Phase Correct* PWM načinu rada. Program mora na pinu PB5 (OC1A) generirati neinvertirajući PWM, i to za različite raspone brojanja registra `TCNT1`. Djelitelj frekvencije radnog takta zadan je i iznosi 64. U funkciji `inicijalizacija()` djelitelj frekvencije radnog takta namješten je funkcijom `timer1_set_prescaler(TIMER1_PRESCALER_64)` i neće se mijenjati za različite raspone brojanja registra `TCNT1`.

U nastavku ćemo koristiti konstante za vršne vrijednosti registra `TCNT1` pri 8-bitnom, 9-bitnom, 10-bitnom te punom rasponu brojanja registra `TCNT1` koje su definirane programskim kodom 8.28.

Programski kod 8.28: Definiranje konstanti koje predstavljaju vršne vrijednosti raspona brojanja (*TOP* varijabla u izrazu (8.26))

```

//TOP vrijednost za PWM signale
#define PWM_8BIT 255
#define PWM_9BIT 511
#define PWM_10BIT 1023
#define PWM_ICR1 ICR1

```

Za 8-bitni *Phase Correct* PWM način rada frekvencija PWM signala bit će:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot (2 \cdot PWM\_8BIT)} = \frac{16000000}{64 \cdot 510} = 490,2 \text{ Hz.} \quad (8.45)$$

Vrijednost `OCR1A` registra za 8-bitni *Phase Correct* PWM način rada, u ovisnosti o analogno-digitalnoj pretvorbi na pinu ADC5, računa se prema sljedećoj relaciji:

$$OCR1A = \frac{ADC\_5}{1023} \cdot PWM\_8BIT \quad (8.46)$$

U programskom kodu 8.27 provedena je konfiguracija za 8-bitni *Phase Correct* PWM način rada funkcijom `timer1_set_phase_correct_pwm_8bit()`. Za promjenu širine impulsa PWM signala koristi se funkcija `OC1A_set_duty_cycle(duty, pwm_bit)` gdje je argument `duty` širina impulsa u rasponu od 0 do 100 %, a argument `pwm_bit` predstavlja vršnu vrijednost registra `TCNT1`. Za 8-bitni raspon brojanja u registru `TCNT1` vrijedi `pwm_bit = PWM_8BIT`. Prevedite datoteku `vjezba826.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ponovite postupak tako da umjesto funkcije `OC1A_set_duty_cycle(duty, pwm_bit)` koristite relaciju (8.46). Na ovaj način direktno upisujete širinu impulsa PWM signala u registar `OCR1A`.

Za 9-bitni *Phase Correct* PWM način rada frekvencija PWM signala bit će:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot (2 \cdot PWM\_9BIT)} = \frac{16000000}{64 \cdot 1022} = 244,62 \text{ Hz.} \quad (8.47)$$

Vrijednost `OCR1A` registra za 9-bitni *Phase Correct* PWM način rada, u ovisnosti o analogno-digitalnoj pretvorbi na pinu ADC5, računa se prema sljedećoj relaciji:

$$OCR1A = \frac{ADC\_5}{1023} \cdot PWM\_9BIT \quad (8.48)$$

Da biste podesili 9-bitni *Phase Correct* PWM način rada u programskom kodu 8.27, napravite sljedeće korake:

- funkciju `timer1_set_phase_correct_PWM_8bit()` zamijenite funkcijom za 9-bitni način rada `timer1_set_phase_correct_PWM_9bit()`,
- varijabli `pwm_bit` dodijelite vrijednost `PWM_9BIT`.

Prevedite datoteku `vjezba826.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ponovite postupak tako da umjesto funkcije `OC1A_set_duty_cycle(duty, pwm_bit)` koristite relaciju (8.48).

Za 10-bitni *Phase Correct* PWM način rada frekvencija PWM signala bit će:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot (2 \cdot PWM_{10BIT})} = \frac{16000000}{64 \cdot 2046} = 122,19 \text{ Hz.} \quad (8.49)$$

Vrijednost `OCR1A` registra za 10-bitni *Phase Correct* PWM način rada, u ovisnosti o analogno-digitalnoj pretvorbi na pinu ADC5, računa se prema sljedećoj relaciji:

$$OCR1A = \frac{ADC_5}{1023} \cdot PWM_{10BIT} = ADC_5 \quad (8.50)$$

Da biste podesili 10-bitni *Phase Correct* PWM način rada u programskom kodu 8.27, napravite sljedeće korake:

- funkciju `timer1_set_phase_correct_PWM_8bit()` zamijenite funkcijom za 10-bitni način rada `timer1_set_phase_correct_PWM_10bit()`,
- varijabli `pwm_bit` dodijelite vrijednost `PWM_10BIT`.

Prevedite datoteku `vjezba826.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ponovite postupak tako da umjesto funkcije `OC1A_set_duty_cycle(duty, pwm_bit)` koristite relaciju (8.50). S obzirom na to da je analogno-digitalna pretvorba širine 10 bitova, vrijednost analogno-digitalne pretvorbe na pinu ADC5 može se direktno dodijeliti registru `OCR1A`.

Sklop *Timer/Counter1* može generirati PWM signal na temelju cijelog raspona brojanja registra `TCNT1` koji je širine 16 bitova ([0, 65535]). Tada se vršna vrijednost do koje će registar `TCNT1` brojati podešava registrom `ICR1`. Frekvencija će u tom slučaju biti:

$$F_{pcPWM} = \frac{F_{CPU}}{PRESCALER \cdot (2 \cdot PWM_{ICR1})} = \frac{F_{CPU}}{PRESCALER \cdot (2 \cdot ICR1)}. \quad (8.51)$$

Relacija (8.51) daje veću slobodu u odabiru frekvencija PWM signala, za razliku od prethodnih slučajeva u kojima su frekvencije mogle biti diskretno odabrane. Ukoliko želimo generirati PWM signal željene frekvencije, tada je vrijednost registra `ICR1` potrebno izračunati na sljedeći način:

$$ICR1 = \frac{F_{CPU}}{PRESCALER \cdot 2 \cdot F_{pcPWM}}. \quad (8.52)$$

Na primjer, ako želite generirati PWM signal frekvencije 50 Hz, tada će vrijednost registra `ICR1` biti:

$$ICR1 = \frac{F\_CPU}{PRESCALER \cdot 2 \cdot F\_pcPWM} = \frac{16000000}{64 \cdot 2 \cdot 50} = 2500. \quad (8.53)$$

Vrijednost `OCR1A` registra za slučaj kada raspon brojanja registra `TCNT1` definiramo pomoću registra `ICR1` za *Phase Correct* PWM način rada, u ovisnosti o analogno-digitalnoj pretvorbi na pinu ADC5, računa se prema sljedećoj relaciji:

$$OCR1A = \frac{ADC\_5}{1023} \cdot PWM\_ICR1 \quad (8.54)$$

Da biste podesili *Phase Correct* PWM način rada u punom rasponu brojanja registra `TCNT1` u programskom kodu 8.27 napravite sljedeće korake:

- funkciju `timer1_set_phase_correct_PWM_8bit()` zamijenite funkcijom za način rada u cijelom rasponu `[0, 65535]` `timer1_set_phase_correct_PWM_ICR1(2500)` (PWM signal bit će frekvencije 50 Hz),
- varijabli `pwm_bit` dodijelite vrijednost `PWM_ICR1`.

Prevedite datoteku `vjezba826.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Ponovite postupak tako da umjesto funkcije `OC1A_set_duty_cycle(duty, pwm_bit)` koristite relaciju (8.54).

Promijenite djelitelj frekvencije radnog takta na iznos 1024. Prevedite datoteku `vjezba826.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Što se događa s crvenom LED diodom i zašto?

Zatvorite datoteku `vjezba826.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 8.2.4 Zadaci - PWM način rada tajmera

#### Zadatak 8.2.1

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Svakih 500 ms u **while** petlji mijenjajte širinu impulsa PWM signala, tako da ona iznosi redom 0 % → 25 % → 50 % → 75 % → 100 % → 0 % perioda  $T$ . Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Fast* PWM načina rada sklopa *Timer/Counter0*.

---

#### Zadatak 8.2.2

Napravite program koji će na pinu PB7 (OC0A) generirati invertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude manja od 3 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Fast* PWM načina rada sklopa *Timer/Counter0*.

---

#### Zadatak 8.2.3

Napravite program koji će:

- na pinu PB5 (OC1A) generirati invertirajući PWM signal pomoću sklopa *Timer/Counter1*,
- na pinu PB6 (OC1B) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter1*,
- na pinu PB7 (OC1C) generirati invertirajući PWM signal pomoću sklopa *Timer/Counter1*.

Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Fast* PWM načina rada sklopa *Timer/Counter1*.

---

#### Zadatak 8.2.4

Napravite program koji će na pinu PB7 (OC0A) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter0*. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Svakih 500 ms u **while** petlji mijenjajte širinu impulsa PWM signala, tako da ona iznosi redom 10 % → 40 % → 70 % → 95 % → 10 % perioda  $T$ . Po mogućnosti, osciloskopom pratiti PWM signal na pinu PB7 (OC0A). Koristite *Phase Correct* PWM načina rada sklopa *Timer/Counter0*.

---

### Zadatak 8.2.5

Napravite program koji će na pinu PB7 (OC0A) generirati invertirajući PWM signal pomoću sklopa *Timer/Counter0* samo ako je pritisnuto tipkalo spojeno na pin PF7. Frekvenciju PWM signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Phase Correct* PWM načina rada sklopa *Timer/Counter0*.

---

### Zadatak 8.2.6

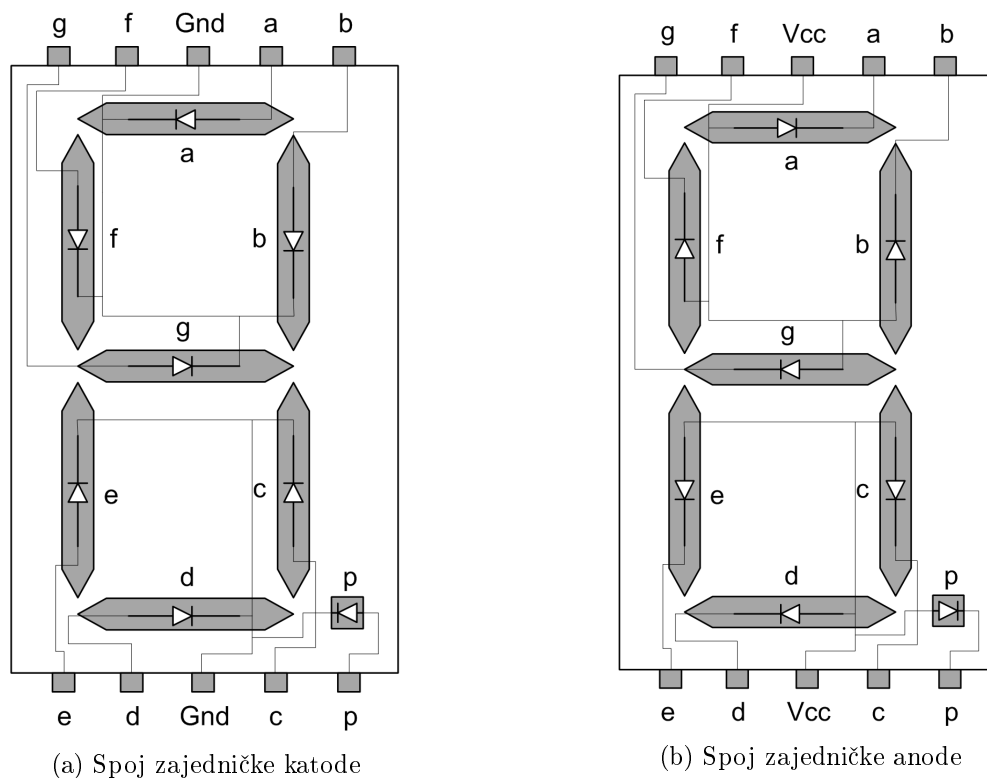
Napravite program koji će na pinu PB6 (OC1B) generirati neinvertirajući PWM signal pomoću sklopa *Timer/Counter1*. Frekvenciju PWM signala namjestite tako da bude jednaka 50 Hz. Omogućite promjenu širine impulsa PWM signala pomoću potenciometra spojenog na pin ADC5. Na LCD displeju ispišite frekvenciju PWM signala i širinu impulsa PWM signala u postocima. Koristite *Phase Correct* PWM načina rada sklopa *Timer/Counter1* punom rasponu brojanja registra **TCNT1**.

---

## Poglavlje 9

# Numerički displej i posmačni registar

Numerički displej najčešće se koristi za prikaz brojeva pomoću segmenata. Svaki je segment LED dioda koja može biti uključena ili isključena. Mnogi numerički displeji imaju decimalnu točku kako bi omogućili ispis realnih brojeva. Numerički displej sa sedam segmenata za prikaz brojeva i jednim segmentom za prikaz decimalne točke prikazan je na slici 9.1.



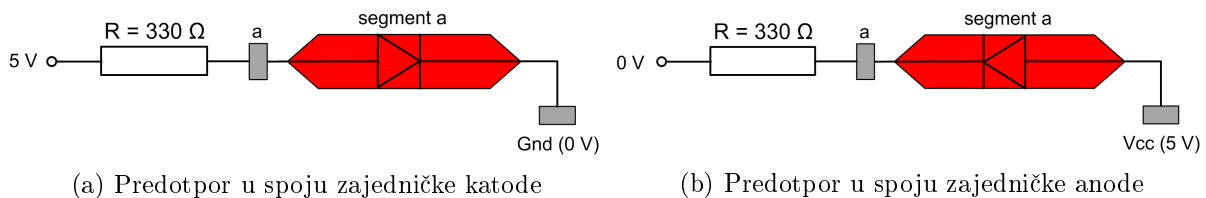
Slika 9.1: Numerički displej sa sedam segmenata za prikaz brojeva i jednim segmentom za prikaz decimalne točke

Imena segmenata za prikaz brojeva jesu a, b, c, d, e, f i g, dok je ime segmenta za prikaz decimalne točke p. Kombinacijom uključenih i isključenih segmenata a, b, c, d, e, f i g može se prikazati bilo koja znamenka dekadskog sustava, uključujući i slova potrebna za prikaz znamenaka heksadekadskog sustava. Postoje dvije izvedbe numeričkih displeja:

- spoj zajedničke katode (slika 9.1a) i
- spoj zajedničke anode (slika 9.1b).

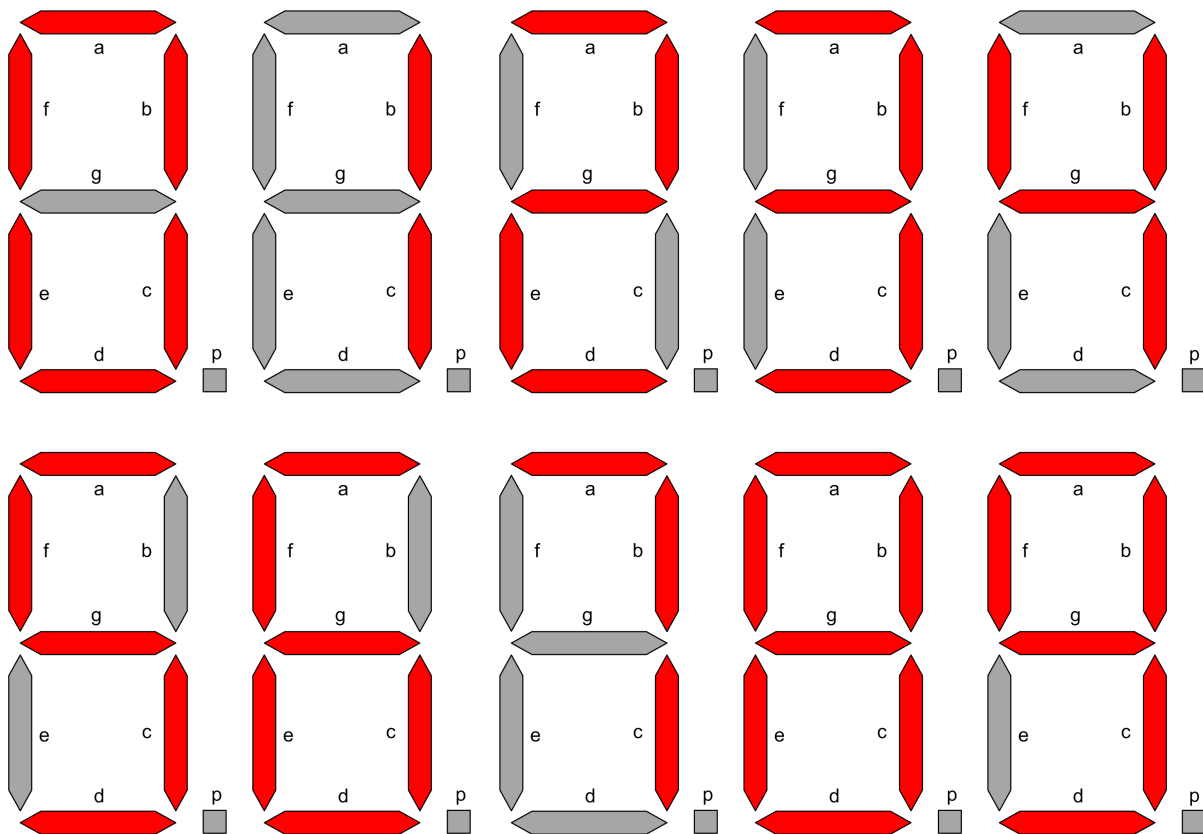
U spoju zajedničke katode sve katode LED dioda koje čine segmente spojene su na zajednički potencijal  $Gnd$  (0 V). U tom slučaju jedan segment uključujemo tako da na pin tog segmenta preko predotpora od  $330 \Omega$  dovedemo potencijal od 5 V (slika 9.2a).

U spoju zajedničke anode sve anode LED dioda koje čine segmente spojene su na zajednički potencijal  $Vcc$  (5 V). U tom slučaju jedan segment uključujemo tako da na pin tog segmenta preko predotpora od  $330 \Omega$  dovedemo potencijal od 0 V (slika 9.2b).



Slika 9.2: Predotpori u spoju zajedničke katode i anode za segment a

Predotpor na slici 9.2 služi kao strujna zaštita LED diode segmenta, a ujedno i kao zaštita digitalnog pina mikroupravljača od prevelike struje. Na slici 9.3 i u tablici 9.1 prikazani su uključeni i isključeni segmenti za prikaz znamenaka iz dekadskog sustava. U tablici 9.1 za segment  $p$  koji se odnosi na decimalnu točku stanje je nedefinirano i označeno je slovom  $x$ . Segment  $p$  uključuje se prema potrebi, odnosno kada prikazujemo realne brojeve. Za prikaz svih znamenaka dekadskog sustava potrebno je osam digitalnih pinova kojima ćemo uključivati ili isključivati pojedini segment. U spoju zajedničke katode visoko stanje na digitalnom pinu mikroupravljača uključuje segment, a nisko stanje isključuje segment. U spoju zajedničke anode nisko stanje na digitalnom pinu mikroupravljača uključuje segment, a visoko stanje isključuje segment.



Slika 9.3: Prikaz uključених и isključених сегмената за знаменке декадског система



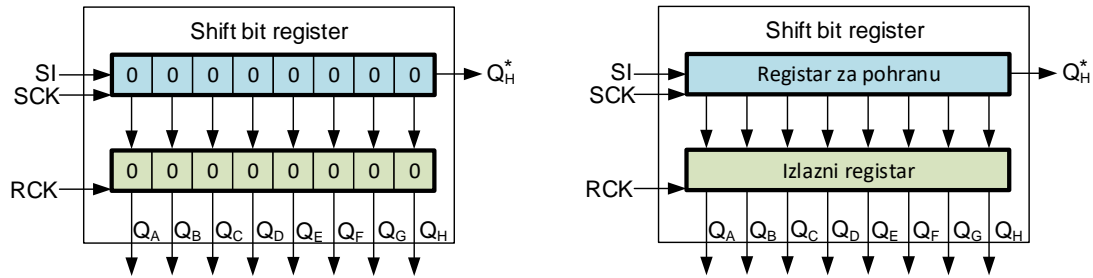
Tablica 9.1: Stanja pojedinih segmenata za prikaz jedne znamenke dekadskog sustava

Znamenka	Segmenti							
	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

U narednim vježbama prezentirat ćemo četveroznamenaste brojeve na četiri numerička displeja. Za prikaz znamenke na jednom numeričkom displeju potrebno je 8 digitalnih izlaza. Ako želimo prikazati četiri znamenke na četiri numerička displeja, potrebna su 32 digitalna izlaza mikroupravljača. Na početku ovog udžbenika naveli smo kako mikroupravljač ATmega32U4 ima 26 digitalnih pinova. Prema tome, pomoću mikroupravljača ATmega32U4 nije moguće prikazati četveroznamenasti broj na način da upravljamo svakim segmentom, već je moguće najviše prikazati troznamenasti broj gdje bi utrošak digitalnih pinova bio 24.

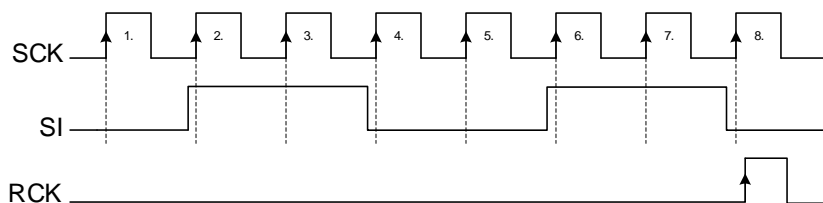
Često je u praksi potrebno proširiti digitalne izlaze nekog mikroupravljača, a naročito kada se koriste numerički displeji, matični displeji i drugi uređaji koji zahtijevaju velik utrošak digitalnih izlaza. Jedan od načina proširenja digitalnih izlaza jest korištenje posmačnog registra sa serijskim ulazom (engl. *Shift Bit Register*). Najčešće su posmačni registri širine 8 bitova, odnosno 8 digitalnih izlaza. Primjer jednog takvog posmačnog registra jest 74HC595D koji ćemo koristiti u vježbi. Blok shema posmačnog registra prikazana je na slici 9.4. Na shemi sa slike 9.4 prikazani su sljedeći osnovni signali i elementi posmačnog registra:

- **Registar za pohranu** (engl. *Storage Register*) - registar širine 8 bitova koji se puni nizom binarnih podataka s ulaza SI,
- SI - (engl. *Serial Input*) ulaz na koji se dovodi serijski niz binarnih podataka (0 ili 1),
- SCK - (engl. *Serial Clock*) ulaz na koji se dovodi signal takta. Na svaki rastući brid signala takta binarni podatak s ulaza SI upisuje se u **Registar za pohranu** (0 ili 1  $\rightarrow Q_A$ ), dok se istovremeno svi bitovi **Registra za pohranu** posmiču desno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ).
- $Q_H^*$  - izlaz iz posmačnog registra na koji se prosljeđuje bit  $Q_H$  nakon rastućeg brida signala SCK,
- **Izlazni registar** (engl. *Output Register*) - registar koji se puni bitovima iz **Registra za pohranu** na svaki rastući brid signala RCK,
- RCK - (engl. *Register Clock*) - ulaz koji na rastući brid signala puni **Izlazni registar** (kopira bitove **Registra za pohranu** u **Izlazni registar**),
- $Q_A, Q_B, \dots, Q_H$  - prošireni izlazi posmačnog registra čije je stanje istovjetno stanju bitova u **Izlaznom registru**.



Slika 9.4: Blok shema posmačnog registra

Postupka punjenja posmačnog registra prikazat ćemo na primjeru binarne kombinacije 01100110. Stanja ulaznih signala SI, SCK, i RCK posmačnog registra za punjenje binarnom kombinacijom 01100110 prikazana su na slici 9.5.



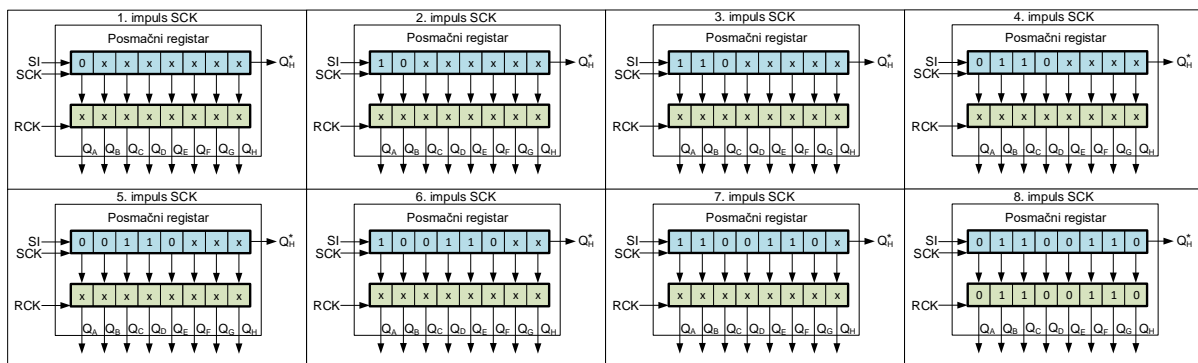
Slika 9.5: Stanja ulaznih signala posmačnog registra za punjenje binarnom kombinacijom 01100110

Postupak punjenja posmačnog registra binarnom kombinacijom 01100110 prikazan je na slici 9.6, a provodi se na sljedeći način (pretpostavimo da ne znamo prethodna stanja Registra za punjenje i Izlaznog registra te ta stanja označimo s x):

- na 1. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 0. Vrijednost bita 0 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 0xxxxxxx) (slika 9.6 - 1. impuls SCK),
- na 2. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 1. Vrijednost bita 1 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 10xxxxxx) (slika 9.6 - 2. impuls SCK),
- na 3. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 1. Vrijednost bita 1 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 110xxxxx) (slika 9.6 - 3. impuls SCK),
- na 4. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 0. Vrijednost bita 0 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 0110xxxx) (slika 9.6 - 4. impuls SCK),
- na 5. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 0. Vrijednost bita 0 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 00110xxx) (slika 9.6 - 5. impuls SCK),

- na 6. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 1. Vrijednost bita 1 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 100110xx) (slika 9.6 - 6. impuls SCK),
- na 7. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 1. Vrijednost bita 1 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 1100110x) (slika 9.6 - 7. impuls SCK),
- na 8. rastući brid signala SCK (slika 9.5), stanje ulaza SI je 0. Vrijednost bita 0 ulazi u Registar za pohranu na poziciju  $Q_A$  uz posmicanje Registra za pohranu udesno ( $Q_A \rightarrow Q_B, Q_B \rightarrow Q_C, \dots, Q_G \rightarrow Q_H, Q_H \rightarrow Q_H^*$ ; stanje Registra za pohranu je 01100110) (slika 9.6 - 8. impuls SCK),
- na rastući brid signala RCK (slika 9.5), Izlazni registar puni se kombinacijom 01100110 iz Registra za punjenje,
- stanja izlaza posmačnog registra jesu:  $Q_A = 0, Q_B = 1, Q_C = 1, Q_D = 0, Q_E = 0, Q_F = 1, Q_G = 1, Q_H = 0$ .

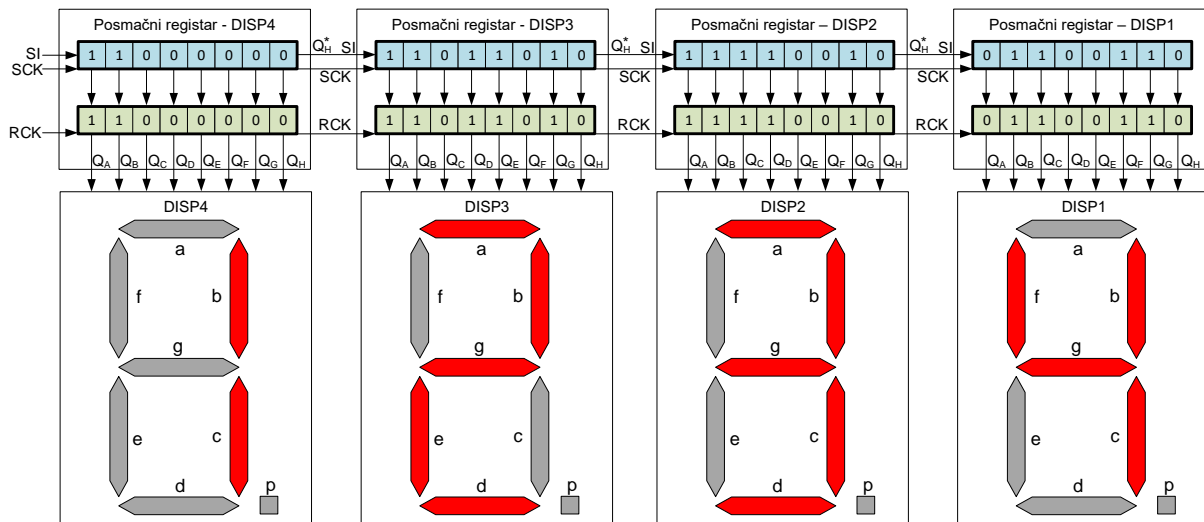
Izlaz  $Q_A$  predstavlja stanje najznačajnijeg bita (engl. *Most Significant Bit* - MSB), a izlaz  $Q_H$  predstavlja stanje najmanje značajnog bita (engl. *Least Significant Bit* - LSB).



Slika 9.6: Postupak punjenja posmačnog registra binarnom kombinacijom 01100110

Da bismo poslali 8 bitova podataka na posmačni registar, potrebno je koristiti tri digitalna izlaza mikroupravljača. Jedan posmačni registar može upravljati jednim numeričkim displejom tako da se izlaz  $Q_A$  spoji na segment a,  $Q_B$  spoji na segment b,  $Q_C$  spoji na segment c, ...,  $Q_H$  spoji na segment p. Za četiri numerička displeja potrebno je koristiti četiri posmačna registra koji se mogu povezati serijski na sljedeći način (slika 9.7):

- ulaz SI prvog posmačnog registra spajamo na digitalni izlaz mikroupravljača,
- ulaz SI drugog posmačnog registra spajamo na izlaz  $Q_H^*$  prvog posmačnog registra,
- ulaz SI trećeg posmačnog registra spajamo na izlaz  $Q_H^*$  drugog posmačnog registra,
- ulaz SI četvrtog posmačnog registra spajamo na izlaz  $Q_H^*$  trećeg posmačnog registra,
- SCK ulazi svih posmačnih registara spajaju se zajedno na digitalni izlaz mikroupravljača,
- RCK ulazi svih posmačnih registara spajaju se zajedno na digitalni izlaz mikroupravljača.



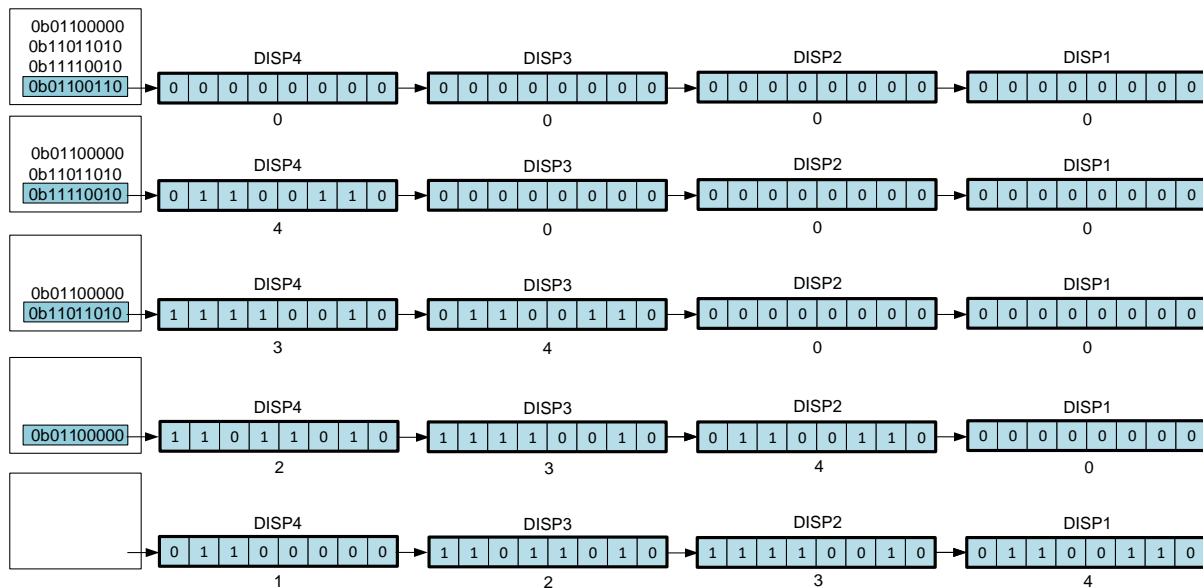
Slika 9.7: Shema spajanja četiriju posmačnih registara u seriju s numeričkim displejima

Ako spajamo posmačne registre na ovaj način, utrošak digitalnih izlaza mikroupravljača za upravljanje s četiri numerička displeja i dalje je tri. Vrijeme punjenja četiriju posmačnih registara bit će četiri puta veće. Pretpostavimo da na četiri numerička displeja želimo prikazati sljedeće znamenke: DISP4 = 1, DISP3 = 2, DISP2 = 3 i DISP1 = 4. Stanja pojedinih segmenata za prikaz jedne znamenke dekadskog sustava na numeričkom displeju prikazana su u tablici 9.2. Prema tablici 9.2, znamenka 1 odgovara binarnoj kombinaciji segmenata 01100000, znamenka 2 odgovara binarnoj kombinaciji segmenata 11011010, znamenka 3 odgovara binarnoj kombinaciji segmenata 11110010, a znamenka 4 odgovara binarnoj kombinaciji segmenata 01100110. Pretpostavimo da je numerički displej DISP4 spojen na prvi posmačni registar čiji je ulaz SI spojen na digitalni izlaz mikroupravljača. Numerički displej DISP3 spojen je na drugi u seriji posmačni registar (slika 9.7) itd.

Tablica 9.2: Stanja pojedinih segmenata za prikaz jedne znamenke dekadskog sustava na numeričkom displeju

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

Postupak punjenja četiriju bajtova podataka u četiri posmačna registra prikazan je na slici 9.8. S obzirom na način kako su posmačni registri međusobno spojeni, binarna kombinacija za numerički displej DISP1 koji je zadnji u nizu mora ići prva u serijski niz posmačnih registara.



Slika 9.8: Postupak punjenja četiriju bajtova podataka u četiri posmačna registra

Pretpostavimo da su segmenti svih četiriju numeričkih displeja bili isključeni (slika 9.8 prvi redak). Postupak punjenja četiriju bajtova podataka provodi se na sljedeći način:

1. na ulaz SI šalje se binarna kombinacija za numerički displej DISP1, odnosno znamenka 4 čija je binarna kombinacija jednaka 01100110. Svi posmačni registri posmiču se udesno za 8 mjesta. Binarna kombinacija za numerički displej DISP1 nalazi se na prvom posmačnom registru (slika 9.8 drugi redak).
2. na ulaz SI šalje se binarna kombinacija za numerički displej DISP2, odnosno znamenka 3 čija je binarna kombinacija jednaka 11110010. Svi posmačni registri posmiču se udesno za 8 mjesta. Binarna kombinacija za numerički displej DISP1 nalazi se na drugom posmačnom registru, a za numerički displej DISP2 nalazi se na prvom posmačnom registru (slika 9.8 treći redak).
3. na ulaz SI šalje se binarna kombinacija za numerički displej DISP3, odnosno znamenka 2 čija je binarna kombinacija jednaka 11011010. Svi posmačni registri posmiču se udesno za 8 mjesta. Binarna kombinacija za numerički displej DISP1 nalazi se na trećem posmačnom registru, za numerički displej DISP2 nalazi se na drugom posmačnom registru, a za numerički displej DISP3 nalazi se na prvom posmačnom registru (slika 9.8 četvrti redak).
4. na ulaz SI šalje se binarna kombinacija za numerički displej DISP4, odnosno znamenka 1 čija je binarna kombinacija jednaka 01100000. Svi posmačni registri posmiču se udesno za 8 mjesta. Binarna kombinacija za numerički displej DISP1 nalazi se na četvrtom posmačnom registru, za numerički displej DISP2 nalazi se na trećem posmačnom registru, za numerički displej DISP3 nalazi se na drugom posmačnom registru, a za numerički displej DISP4 nalazi se na prvom posmačnom registru (slika 9.8 peti redak).

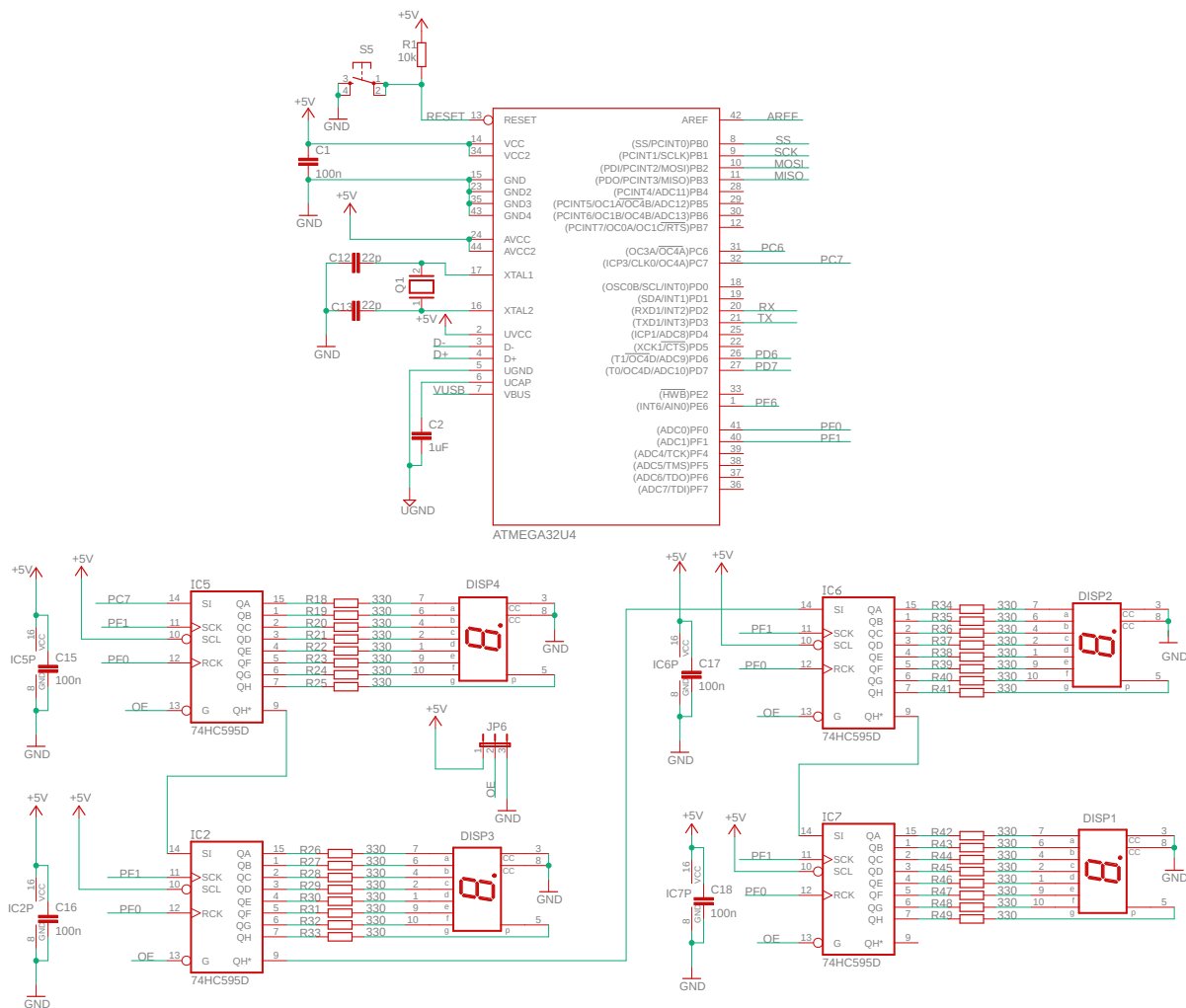
Kada sva četiri posmačna registra napunimo s 4 bajta podataka, potrebno je poslati rastući brid na ulaz RCK kako bi se poslana kombinacija u četiri posmačna registra iz **Registara za pohranu** prosljedila na **Izlazne registre**, odnosno na numeričke displeje. Prikaz brojeva 1, 2, 3 i 4 na numeričkim displejima DISP1, DISP2, DISP3 i DISP4 sa sadržajem posmačnih registara prikazan je na slici 9.7. Treba imati na umu da četveroznamenkasti broj koji će se prikazati na nizu od četiri numerička displeja ovisi o njihovom razmještanju na tiskanoj pločici. Ako je razmještaj

takav da s lijeva na desno imamo niz numeričkih displeja DISP4, DISP3, DISP2 i DISP1, tada će numerički displeji prikazivati broj 1234 (razmještaj na slici 9.7).

Ako je razmještaj takav da s lijeva na desno imamo niz numeričkih displeja DISP1, DISP2, DISP3 i DISP4, tada će numerički displeji prikazivati broj 4321. Na ovaj način spojeni su numerički displeji na razvojnom okruženju sa slike 3.1.

## 9.1 Vježbe - numerički displej i posmačni registar

Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.



Slika 9.9: Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4

Na shemi se nalaze četiri numerička displeja u spoju zajedničke katode sa sedam segmenata (a, b, c, d, e, f, g) za prikaz znamenaka i jednim segmentom (p) za prikaz decimalne točke. Numerički displeji imaju oznake DISP1, DISP2, DISP3 i DISP4, a povezani su na izlaze posmačnih registara  $Q_A$ ,  $Q_B$ ,  $Q_C$ ,  $Q_D$ ,  $Q_E$ ,  $Q_F$ ,  $Q_G$ ,  $Q_H$ . Način spajanja segmenata numeričkog displeja prikazan je na slici 9.9 ( $Q_A \rightarrow a$ ,  $Q_B \rightarrow b$ ,  $Q_C \rightarrow c$ ,  $Q_D \rightarrow d$ ,  $Q_E \rightarrow e$ ,  $Q_F \rightarrow f$ ,  $Q_G \rightarrow g$ ,  $Q_H \rightarrow p$ ). Ulaz SI prvog posmačnog registra spojen je na pin PC7. Ulazi SCK svih posmačnih registara spojeni su na pin PF1, dok su ulazi RCK svih posmačnih registara spojeni na pin PF0.

Ulaz SI prvog posmačnog registra spojen je na mikroupravljač. Izlaz  $Q_H^*$  prvog posmičnog registra spojen je na ulaz SI drugog posmičnog registra. Nadalje, izlaz  $Q_H^*$  drugog posmičnog registra spojen je na ulaz SI trećeg posmičnog registra te je izlaz  $Q_H^*$  trećeg posmičnog registra spojen na ulaz SI četvrtog posmičnog registra. U prikazu četveroznamenkastog broja vrijedi sljedeće:

- numerički displej s oznakom DISP1 prikazuje znamenke tisućica i spojen je na četvrti posmačni registar,
- numerički displej s oznakom DISP2 prikazuje znamenke stotica i spojen je na treći posmačni registar,
- numerički displej s oznakom DISP3 prikazuje znamenke desetice i spojen je na drugi posmačni registar,
- numerički displej s oznakom DISP4 prikazuje znamenke jedinica i spojen je na prvi posmačni registar,

Prema tome, kada se posmačni registri pune četveroznamenkastim brojevima, pune se redoslijedom: znamenka tisućice, znamenka stotice, znamenka desetice i na kraju znamenka jedinice.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku SSD-SBR.zip. Na radnoj površini stvorite praznu datoteku koju ćete nazvati **Vaše Ime i Prezime** ne koristeći pritom diakritičke znakove. Na primjer, ako je Vaše ime Ivica Ivić, datoteka koju ćete stvoriti zvat će se Ivica Ivic. Datoteku SSD-SBR.zip raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite VUB mikroracunala.atsln u datoteci \\SSD-SBR\\vjezbe. U otvorenom projektu nalaze se sve vježbe koja ćemo obraditi u poglavlju **Numerički displej i posmačni registar**. 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. Na razvojnom okruženju sa slike 3.1 odspojite LCD displej.



### Vježba 9.1.1

Napravite program pomoću kojeg ćete na sva četiri posmačna registra postaviti 8-bitni podatak 0x66. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

U projektnom stablu otvorite datoteku vjezba911.cpp. Omogućite samo prevođenje datoteke vjezba911.cpp. Početni sadržaj datoteke vjezba911.cpp prikazan je programskim kodom 9.1.

Programski kod 9.1: Početni sadržaj datoteke vjezba911.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>

void inicijalizacija() {

    output_port(DDRC, PC7); // digitalni izlaz spojen na SI
    output_port(DDRF, PF1); // digitalni izlaz spojen na SCK
    output_port(DDRF, PF0); // digitalni izlaz spojen na RCK
}
```

```

int main(void) {

    inicijalizacija();

    uint8_t number = 0x66;

    for (int i = 0; i < 8; i++) {
        if(number & 0x01) { // ako je number LSB == 1
            set_port(PORTC, PC7, 1); // na SI pošalji 1
        }
        else { // ako je number LSB == 0
            set_port(PORTC, PC7, 0); // na SI pošalji 0
        }
        // rastući brid signala SCK
        set_port(PORTF, PF1, 1); //SCK = 1
        _delay_us(1);
        set_port(PORTF, PF1, 0); //SCK = 0
        _delay_us(1);

        number = number >> 1; // posmakni broj za jedan u desno
    }
    // rastući brid signala RCK
    set_port(PORTF, PF0, 1); //RCK = 1
    _delay_us(1);
    set_port(PORTF, PF0, 0); //RCK = 0
    _delay_us(1);

    while(1) {

        _delay_ms(100);
    }
    return 0;
}

```

Rezultat slanja 8-bitnog podatka 0x66 na sva četiri posmačna registra bit će uključenje jednakih segmenata na sva četiri numerička displeja koji su spojeni na posmačne registre. Koji će znak biti prikazan na numeričkim displejima zasad ne znamo.

U programskom kodu 9.1 nalazi se programsko rješenje koje postavlja jedan 8-bitni podatak na prvi posmačni registar<sup>1</sup>. U funkciji `inicijalizacija()` konfigurirani su sljedeći digitalni izlazi za potrebe slanja podataka na posmačni registar:

- pin PC7 spojen je na ulaz SI prvog posmačnog registra,
- pin PF1 spojen je na ulaz SCK svih četiri posmačna registra,
- pin PF0 spojen je na ulaz RCK svih četiri posmačna registra.

Binarna kombinacija za broj 0x66 koji se šalje na prvi posmačni registar jest 01100110. U `main()` funkciji za slanje 8 bitova koristi se `for` petlja koja se izvršava osam puta. U jednom koraku `for` petlje provode se sljedeće radnje:

- naredbom `if(number & 0x01)` ispitujemo stanje LSB bita, odnosno bita 0. Ako je bit 0 u stanju 1, tada se na izlazni pin PC7 koji je spojen na ulaz SI postavlja visoko stanje (binarna 1). Inače, se na izlazni pin PC7 koji je spojen na ulaz SI postavlja nisko stanje (binarna 0). U prvom koraku `for` petlje SI = 0.

<sup>1</sup>Ako se mikroupravljač resetira više puta, a razvojno okruženje ne izgubi napajanje, svi posmačni registri popunit će se istim 8-bitnim podatkom.



- nakon što je pripremljen bit na ulazu SI, na izlaznom pinu PF1 koji je spojen na ulaz SCK generira se impuls trajanja  $1 \mu s^2$  kako bi se na rastući brid signala SCK napravio posmak bitova u posmačnim registrima.
- naredbom `number = number >> 1`; binarna kombinacija posmiče se udesno za jedno mjesto kako bi u sljedećem koraku na poziciju bita 0 došao sljedeći bit koji se šalje na SI ulaz. U prvom koraku `for` petlje varijabla `number` nakon posmicanja udesno za jedan korak poprima vrijednost 00110011.

Nakon osam koraka `for` petlje 8-bitni podatak postavljen je u prvi posmačni registar. Ako taj podatak želimo postaviti na izlaze posmačnog registra, tada se na izlaznom pinu PF0 koji je spojen na ulaz RCK generira impuls trajanja  $1 \mu s$  kako bi se na rastući brid sadržaj `Registra za punjenje` prosljedio na `Izlazni registar`. Nakon slanja jednog 8-bitnog podatka na prvi posmačni registar `while` petlja se izvršava s kašnjenjem iznosa od 100 ms.

Prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Odspojite napajanje na razvojnom okruženju i spojite ga nazad. Na numeričkom displeju `DISP4` prikazana je znamenka 4. Pritisnite tipku `RESET` kako biste resetirali mikroupravljač barem tri puta. Primijetite kako se sada na svima numeričkim displejima pojavila znamenka 4. Zašto?

Da biste odmah na sva četiri numerička displeja ispisali kombinaciju 4444, `for` petlju u programskom kodu 9.1 potrebno je izvršiti četiri puta. To ćete napraviti tako da `for` petlju kojom punite posmačni registar ugnjezdite u `for` petlju koja će se izvršiti četiri puta. Vanjska `for` petlja prikazana je u programskom kodu 9.2. S obzirom na to da varijabla `number` nakon osam posmicanja udesno poprima vrijednost 0, varijabli `number` u svakoj je iteraciji vanjske petlje ponovno potrebno dodijeliti vrijednost `0x66`. Nakon što se napune sva četiri posmična registra, impulsom na ulaz RCK osvježavaju se izlazi posmičnih registara.

Programski kod 9.2: Vanjska `for` petlja koju je potrebno dodati u programski kod 9.1

```
for(int n = 0; n < 4; n++) {
    number = 0x66;

    // for petlja za punjenje posmačnog registra
}

// rastući brid signala RCK
set_port(PORTF, PF0, 1); //RCK = 1
_delay_us(1);
set_port(PORTF, PF0, 0); //RCK = 0
_delay_us(1);
```

Ponovno prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

U programskom kodu 9.3 prikazan je novi sadržaj kojim je potrebno zamijeniti programski kod 9.1 u datoteci `vjezba911.cpp`. U ovom slučaju stvoreni su sljedeći objekti tipa `DigitalOutput`:

- `DigitalOutput ShiftReg_SI(C7)` - objekt koji upravlja pinom PC7 koji je spojen na ulaz SI prvog posmačnog registra,
- `DigitalOutput ShiftReg_SCK(F1)` - objekt koji upravlja pinom PF1 koji je spojen na ulaz SCK prvog posmačnog registra,

<sup>2</sup>Prema tehničkim specifikacijama sklopa 74HC595D, ovo vrijeme mora biti veće od 15 ns pa se kašnjenje iznosa  $1 \mu s$  može izbrisati iz programskog koda, što će dati impuls širine 62.5 ns ( $1/16 \text{ MHz} = \text{jedna instrukcija}$ ).

- `DigitalOutput ShiftReg_RCK(F0)` - objekt koji upravlja pinom PF0 koji je spojen na ulaz RCK prvog posmačnog registra.

Programski kod 9.3: Novi sadržaj datoteke `vjezba911.cpp` - korištenje klase `DigitalOutput`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"

int main(void) {

    DigitalOutput ShiftReg_SI(C7);
    DigitalOutput ShiftReg_SCK(F1);
    DigitalOutput ShiftReg_RCK(F0);

    uint8_t number;

    for(int n = 0; n < 4; n++) {
        number = 0x66;
        for(int i = 0; i < 8; i++) {
            if(number & 0x01) { // ako je bit0 == 1
                ShiftReg_SI.set(); // na SI pošalji 1
            }
            else { // ako je bit0 == 0
                ShiftReg_SI.reset(); // na SI pošalji 0
            }
            // rastući brid signala SCK
            ShiftReg_SCK.set(); //SCK = 1
            _delay_us(1);
            ShiftReg_SCK.reset(); //SCK = 0
            _delay_us(1);

            number = number >> 1; // posmakni broj za jedan u desno
        }
        // rastući brid signala RCK
        ShiftReg_RCK.set(); //RCK = 1
        _delay_us(1);
        ShiftReg_RCK.reset(); //RCK = 0
        _delay_us(1);
    }

    while(1) {

        _delay_ms(100);
    }
    return 0;
}
```

Primijetite da su makronaredbe `set_port` zamijenjene pozivima metoda nad objektima (`ShiftReg_SCK.set()` umjesto `set_port(PORTF, PF0, 1)` itd.). Ostali dio koda isti je kao u programskom kodu 9.1. Ponovno prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

U programskom kodu 9.4 prikazan je novi sadržaj koji je potrebno napisati u datoteku `vjezba911.cpp` koji je temeljen na sljedećim funkcijama koje su napisane u zaglavlju `ssd_sbr.h`:

- `shift_bit_init()` - funkcija koja inicijalizira posmačni registar, odnosno konfigurira digitalne izlaze mikroupravljača spojene na ulaze posmačnog registra SI, SCK i RCK.
- `shift_bit_clock_sck()` - funkcija za generiranje impulsa širine 1  $\mu$ s na ulazu SCK,

- `shift_bit_clock_rck()` - funkcija za generiranje impulsa širine  $1 \mu\text{s}$  na ulazu RCK,
- `shift_bit_send_1byte(uint8_t number)` - funkcija za slanje jednog bajta podataka (`number`) na posmačni registar,
- `shift_bit_send_4byte(uint8_t num1, uint8_t num2, uint8_t num3, uint8_t num4)` - funkcija za slanje četiriju bajtova (`num1`, `num2`, `num3`, `num4`) na posmačni registar redoslijedom argumenata funkcije.

Funkcije za slanje podataka na posmačne registre zasnovane su na programskom kodu 9.1.

Programski kod 9.4: Novi sadržaj datoteke `vjezba911.cpp` - korištenje funkcija zaglavlja `ssd_sbr.h`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "SSD i SBR/ssd_sbr.h"

void inicijalizacija() {
    // konfiguracija pinova mikroupravljača koji su spojeni
    // na Shift Bit Register (definirati konstante u ssd_sbr.h)
    shift_bit_init();
}

int main(void) {

    inicijalizacija();

    uint8_t number = 0x66;

    // petlja koja pojedinačno šalje četiri bajta
    for(int n = 0; n < 4; n++) {
        shift_bit_send_1byte(number);
    }
    // generiranje impulsa na RCK
    shift_bit_clock_rck();

    while(1) {

        _delay_ms(100);
    }
    return 0;
}
```

Neposredno prije korištenja posmačnog registra spojenog na mikroupravljač u zaglavlju `ssd_sbr.h` konstantama je potrebno dodijeliti informacije o pinovima mikroupravljača koji su spojeni na ulaze posmačnog registra SI, SCK i RCK. Definirane konstante u zaglavlju `ssd_sbr.h` prikazane su u programskom kodu 9.5. S obzirom na to da je pin PC7 spojen na ulaz SI, konstanti `SI_DDR` dodijeljen je registar `DDRC`, konstanti `SI_PORT` dodijeljen je registar `PORTC`, a konstanti `SI_PIN` dodijeljena je pozicija pina PC7. Isto vrijedi i za ostale ulaze posmačnog registra.

Programski kod 9.5: Definiranje konstanti za konfiguriranje pinova mikroupravljača spojenih na ulaze posmačnog registra

```
//DDR registri za ulaze SI, SCK i RCK
#define SI_DDR DDRC
#define SCK_DDR DDRF
#define RCK_DDR DDRF

//PORT registri za ulaze SI, SCK i RCK
#define SI_PORT PORTC
#define SCK_PORT PORTF
#define RCK_PORT PORTF

//pinovi za ulaze SI, SCK i RCK
#define SI_PIN PC7
#define SCK_PIN PF1
#define RCK_PIN PF0
```

Prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Zamijenite sada `for` petlju u programskom kodu 9.4 koja pojedinačno šalje 4 bajta podatka na posmačne registre pomoću funkcije `shift_bit_send_1byte(uint8_t number)` funkcijom koja odjednom šalje 4 bajta podataka na posmačne registra. Ponovno prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

U zaglavlje `DigitalIO.h` dodana je nova klasa koja se zove `Shift_Bit_Register`. U klasi `Shift_Bit_Register` definirana su dva preopterećena konstruktora, od kojih je jedan tipa `default`. Stvaranje objekata koji su tipa `Shift_Bit_Register` moguće je na sljedeći način:

- `Shift_Bit_Register SBR(C7, F1, F0);` - objekt se inicijalizira trima konstantama oblika  $Xi$  ( $X = B, C, D, E, F; i = 0, 1, 2, 3, 4, 5, 6, 7$ ) gdje  $X$  označava port, a  $i$  pin na portu. Npr. za pin 7 na portu C koristit ćemo konstantu `C7`, a za pin 1 na portu F koristit ćemo konstantu `F1`. Prvi argument konstruktora jest pin na koji je spojen ulaz SI (PC7), drugi argument konstruktora jest pin na koji je spojen ulaz SCK (PF1), a treći argument konstruktora jest pin na koji je spojen ulaz RCK (PF0).

Korištenje klase `Shift_Bit_Register` prikazano je u programskom kodu 9.6. Biblioteka `DigitalIO.h` u kojoj se nalazi klasa `Shift_Bit_Register` u programski kod 9.6 uključena je pomoću naredbe `#include "DigitalIO/DigitalIO.h"`. Primijetite kako u ovom pristupu nema inicijalizacijske funkcije `inicijalizacija()`. Razlog tome je taj što se objekti trebaju tretirati kao i varijable jer egzistiraju samo u bloku u kojem su definirani. Dakle, objekt `SBR` dohvatljiv je samo u `main()` funkciji.

Klasa `Shift_Bit_Register` ima brojne metode (funkcije klase) koje se mogu pozvati nad objektima:

- `clock_sck()` - metoda generira impuls širine  $1 \mu s$  na ulazu SCK,
- `clock_rck()` - metoda generira impuls širine  $1 \mu s$  na ulazu RCK,
- `send_1byte(uint8_t number)` - metoda služi za slanje jednog bajta podataka (`number`) na posmačni registar,
- `send_4byte(uint8_t num1, uint8_t num2, uint8_t num3, uint8_t num4)` - metoda služi za slanje četiriju bajtova (`num1, num2, num3, num4`) na posmačni registar redosljedom argumenata metode.

Programski kod 9.6: Novi sadržaj datoteke vjezba911.cpp - korištenje klase `Shift_Bit_Register` zaglavlja `DigitalIO.h`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"

int main(void) {
    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);

    uint8_t number = 0x66;
    // petlja koja pojedinačno šalje četiri bajta
    for(int n = 0; n < 4; n++) {
        SBR.send_1byte(number);
    }
    // generiranje impulsa na RCK
    SBR.clock_rck();

    while(1) {
        _delay_ms(100);
    }
    return 0;
}
```

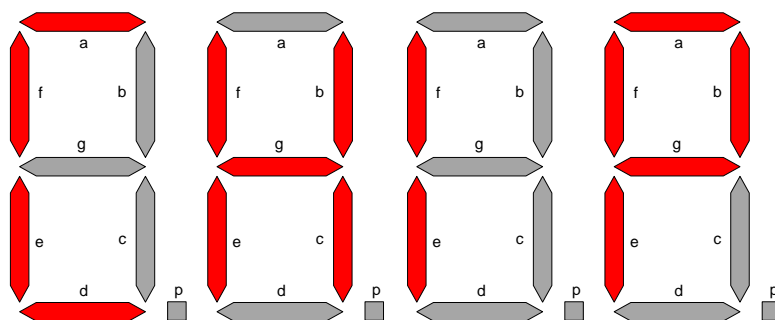
Prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Zamijenite sada `for` petlju u programskom kodu 9.6 koja pojedinačno šalje 4 bajta podatka na posmačne registre pomoću poziva metode nad objektom `SBR.send_1byte(number)` pozivom metode koja odjednom šalje 4 bajta podatka na posmačne registra. Ponovno prevedite datoteku `vjezba911.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba911.cpp` i onemogućite prevođenje ove datoteke.

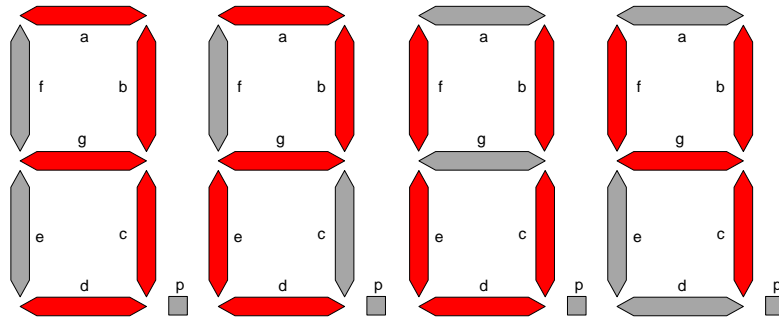


### Vježba 9.1.2

Napravite program u kojem ćete na numeričkim displejima izmjenjivati znakove sa slika 9.10 i 9.11 svakih 1000 ms.



Slika 9.10: Znakovi za prikaz na numeričkim displejima (1)



Slika 9.11: Znakovi za prikaz na numeričkim displejima (2)

Schema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

U projektnom stablu otvorite datoteku `vjezba912.cpp`. Omogućite samo prevođenje datoteke `vjezba912.cpp`. Početni sadržaj datoteke `vjezba912.cpp` prikazan je programskim kodom 9.7.

Programski kod 9.7: Početni sadržaj datoteke `vjezba912.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"

int main(void) {

    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);
    // heksadecimalni kodovi znakova sa slika 9.10 i 9.11
    // inicijalizirajte polja: npr. signs1[4] = {0x66, 0x77, 0x88, 0x99};
    uint8_t signs1[4] = {};
    uint8_t signs2[4] = {};









    while(1) {
        // pošalji znakove sa slike 9.10
        for(int n = 0; n < 4; n++) {
            SBR.send_1byte(signs1[n]);
        }
        SBR.clock_rck(); // generiranje impulsa na RCK
        _delay_ms(1000);

        // pošalji znakove sa slike 9.11
        SBR.send_4byte(signs2[0], signs2[1], signs2[2], signs2[3]);
        _delay_ms(1000);
    }
    return 0;
}
```

U programskom kodu 9.7 prikazana je `main()` funkcija u kojoj je stvoren objekt `SBR` tipa `Shift_Bit_Register`, sukladno uputama u prethodnoj vježbi. Najvažniji zadatak u ovoj vježbi jest kreirati tražene znakove prikazane na slikama 9.10 i 9.11. Znakovi se kreiraju tako da se segmentima koji moraju biti uključeni dodijeli logička vrijednost 1, a segmentima koji moraju biti isključeni dodijeli logička vrijednost 0. Na primjer, znak `C` bit će prezentiran na numeričkom displeju ako su uključeni segmenti `a`, `d`, `e`, `f`, a isključeni segmenti `b`, `c`, `g`, `h`. Temeljem informacije o uključenosti i isključenosti segmenata kreira se binarna kombinacija oblika `abcdefgh`, odnosno za znak `C` `10011100`. Definiranje preostalih znakova sa slika 9.10 i 9.11

prikazano je u tablici 9.3. Definirane znakove potrebno je u programskom kodu 9.7 dodijeliti deklariranim cjelobrojnim poljima<sup>3</sup> `signs1` i `signs2`.

Tablica 9.3: Definiranje znakova sa slika 9.10 i 9.11

Znak	Segmenti								(hex)
	$Q_A$	$Q_B$	$Q_C$	$Q_D$	$Q_E$	$Q_F$	$Q_G$	$Q_H$	
	a	b	c	d	e	f	g	p	
	1	0	0	1	1	1	0	0	0x9C
	0	1	1	0	1	1	1	0	0x6E
	0	0	0	0	1	1	0	0	0x0C
	1	1	0	0	1	1	1	0	0xCE
	1	1	1	1	0	0	1	0	0xF2
	1	1	0	1	1	0	1	0	0xDA
	0	1	1	1	1	1	0	0	0x7C
	0	1	1	0	0	1	1	0	0x66

Izmjena znakova sa slika 9.10 i 9.11 implementirana je u `while` petlji u programskom kodu 9.7 korištenjem metode za slanje jednog bajta (`send_1byte(uint8_t)`) i korištenjem metode za slanje četiriju bajtova odjednom (`send_4byte(uint8_t, uint8_t, uint8_t, uint8_t)`).

Prevedite datoteku `vjezba912.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Promijenite programski kod 9.7 tako da za prikaz znakova ne koristite klasu `Shift_Bit_Register`, već funkcije definirane u zaglavlju `ssd_sbr.h`. Prevedite datoteku `vjezba912.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba912.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 9.1.3

Napravite program u kojim ćete realizirati štopericu na preciznost desetinke sekunde. Štoperica se pokreće i zaustavlja tipkalom spojenim na pin PD0. Resetiranje štoperice na početno stanje 000.0 potrebno je omogućiti pomoću tipkala spojenog na pin PD1. Maksimalno vrijeme koje štoperica može prikazati jest 999.9. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

U projektnom stablu otvorite datoteku `vjezba913.cpp`. Omogućite samo prevođenje datoteke `vjezba913.cpp`. Početni sadržaj datoteke `vjezba913.cpp` prikazan je programskim kodom 9.8.

Programski kod 9.8: Početni sadržaj datoteke `vjezba913.cpp`

```
#include <avr/io.h>
```

<sup>3</sup>Polja je potrebno inicijalizirati: npr `uint8_t signs1[4] = {0x9C, 0x6E, 0x0C, 0xCE};`.

```

#include "AVR VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"
#include "SSD i SBR/ssd_sbr.h"

int main(void) {
    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);
    DigitalInput StartStop(D0); // PD0 ulazni pin
    DigitalInput Clear(D1); // PD1 ulazni pin

    StartStop.pullup_on(); //pull up na PD0
    Clear.pullup_on(); //pull up na PD1

    uint16_t desetinke = 0;
    uint8_t d1 = 0, d2 = 0, d3 = 0, d4 = 0;
    // slanje četiri bajta na posmačne registre
    SBR.send_4byte(digits[d1], digits[d2], digits[d3] | DOT, digits[d4]);

    bool counting = false;

    while(1) {
        // na padajući brid tipkala StartStop
        if(StartStop.isFalling_edge()) {
            if (counting) {
                counting = false;
            }
            else {
                counting = true;
            }
        }
        // na padajući brid tipkala Clear
        if(Clear.isFalling_edge()) {
            desetinke = 0;
            d1 = d2 = d3 = d4 = 0;
            SBR.send_4byte(digits[d1], digits[d2], digits[d3]
                | DOT, digits[d4]);
        }

        if (counting) {
            // računanje znamenaka
            d1 = desetinke / 1000;
            d2 = (desetinke / 100) % 10;
            d3 = (desetinke / 10) % 10;
            d4 = desetinke % 10;

            SBR.send_4byte(digits[d1], digits[d2], digits[d3]
                | DOT, digits[d4]);

            // brojanje desetinki sekunde
            _delay_ms(100);
            if (++desetinke > 9999) {
                desetinke = 0;
            }
        }
    }
    return 0;
}

```

Na početku `main()` funkcije u programskom kodu 9.8 stvoreni su objekt tipa `Shift_Bit_Register` koji se koristi za punjenje posmačnog registra i dva objekta tipa `DigitalInput` koji se koriste za očitavanje stanja tipkala koji su spojeni na digitalne pinove PD0 i PD1. Na digitalnim ulazima PD0 i PD1 uključeni su pritezni otpornici.



Varijable `desetinke` služi za brojanje desetinki sekundi, dok se varijable `d1`, `d2`, `d3` i `d4` koriste za znamenke tisućica, stotica, desetica i jedinica. U zaglavlju `ssd_sbr.h` definirali smo polje s deset elemenata koje prema tablici 9.2 sadrži binarne kombinacije uključenih i isključenih segmenata za prezentaciju brojeva. Naziv tog polja jest `digits`, a znamenke koje želimo prikazati na numeričkim displejima adresiraju se indeksom polja. Na primjer, ako želimo prikazati broj 4 na numeričkom displeju, indeksirat ćemo element `digits[4]` itd. U programskom kodu 9.8 naredbom `#include "SSD i SBR/ssd_sbr.h"` uključili smo zaglavlje `ssd_sbr.h` u kojem je definirano polje `digits`. Početno su sve znamenke postavljene na 0 i poslone na posmačne registre, odnosno prezentirane na numeričkim displejima.

U naredbi `SBR.send_4byte(digits[d1], digits[d2], digits[d3] | DOT, digits[d4]);` na poziciji znamenke `d3` nalazi se konstanta `DOT` u bitovnoj ILI operaciji s `digits[d3]`. Konstanta `DOT` ima vrijednost `0x01`, a služi za uključenje točke, odnosno segmenta `h` na numeričkom displeju. S obzirom na to da nam je cilj prezentirati sekunde na numeričkim displejima, a vrijeme mjerimo na preciznost jedne desetinke sekunde, točka ide iza treće znamenke.

Varijable `counting` je tipa `bool` i koristi se kao indikator na sljedeći način:

- ako je varijable `counting == false`, tada štoperica ne mjeri vrijeme,
- ako je varijable `counting == true`, tada štoperica mjeri vrijeme.

Ova varijable mijenja svoje stanje na svaki padajući brid tipkala `StartStop` pomoću `if` naredbe koja provjerava stanje varijable `counting` i mijenja ga u negirano stanje. Uvjetovani `if` blok naredaba može se zamijeniti jednom naredbom `counting ^= true;` koja mijenja stanje varijable `counting`. Tipkalom `Clear` e varijable `desetinke` i sve znamenke postavljaju se na 0.

Vrijeme koje se nalazi u varijabli `desetinke` potrebno je rastaviti na znamenke tisućica, stotica, desetica i jedinica. Znamenke smo u programskom kodu 9.8 redom izračunali na sljedeći način:

- `d1` - znamenka tisućica dobije se cjelobrojnim dijeljenjem varijable `desetinke` s 1000 (npr.  $1204/1000 = 1 \Rightarrow d1 = 1$ ),
- `d2` - znamenka stotica dobije se tako da se varijable `desetinke` podijeli sa 100, a zatim se izračuna ostatak pri cjelobrojnom dijeljenju s 10 (npr.  $1204/100 = 12 \Rightarrow 12\%10 = 2 \Rightarrow d2 = 2$ )
- `d3` - znamenka desetica dobije se tako da se varijable `desetinke` podijeli s 10, a zatim se izračuna ostatak pri cjelobrojnom dijeljenju s 10 (npr.  $1204/10 = 120 \Rightarrow 120\%10 = 0 \Rightarrow d3 = 0$ ),
- `d4` - znamenka jedinica dobije se tako da se izračuna ostatak cjelobrojnog djeljenja varijable `desetinke` s 10 (npr.  $1204\%10 = 4 \Rightarrow d4 = 4$ ).

Varijable `desetinke` uvećava se u `while` petlji s kašnjenjem od 100 ms i ograničena je na maksimalnu vrijednost 9999, nakon čega se postavlja u 0. Ovo rješenje zapravo nije najzgodnije jer je, osim kašnjenja iznosa od 100 ms, u `while` petlji veliki broj naredaba koje za svoje izvođenje troše mikroprocesorsko vrijeme.

Prevedite datoteku `vjezba913.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Preciznije mjerenje vremena može se ostvariti pomoću tajmera. Dodatno, zanimljivo bi bilo implementirati i mogućnost mjerenja vremena prema gore i prema dolje (odbrojavanje). Navedene značajke implementirane su u programskom kodu 9.9. Za mjerenje vremena korišten je sklop `Timer/Counter1` koji je konfiguriran tako da prekidnu rutinu poziva točno svakih 100 ms. Proučite razlike u programskim kodovima 9.8 i 9.9. Ključne su sljedeće razlike:

- varijable `desetinke` i `count_up` postale su globalne jer im doseg mora biti globalan,
- koristi se varijabla `new_time_ready` koja poprima vrijednost `true` kad tajmer generira novu desetinku sekunde te omogućuje ispis na numeričke displeje. Kada se novo vrijeme ispiše na numeričke displeje, varijabla `new_time_ready` poprima vrijednost `false`,
- ovisno o tome mjeri li štoperica vrijeme ili ne, sklop *Timer/Counter1* omogućuje se ili ne omogućuje,
- stvoren je novi objekt tipa `DigitalInput` naziva `Direction` koji na padajući brid tipkala spojenog na pin PF6 mijenja smjer mjerenja vremena,
- uključena su sva potrebna zaglavlja.

Prepravite datoteku `vjezba913.cpp` u skladu s programskim kodom 9.9 te prevedite datoteku `vjezba913.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Programski kod 9.9: Novi sadržaj datoteke `vjezba913.cpp` - napredniji pristup rješenju

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"
#include "SSD i SBR/ssd_sbr.h"
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

uint16_t desetinke = 0;
bool count_up = true;
bool new_time_ready = false;

ISR(TIMER1_OVF_vect) {
    TCNT1 = 40536;
    new_time_ready = true;
    if(count_up) {
        if(++desetinke > 9999) {
            desetinke = 0;
        }
    }
    else {
        if(--desetinke == 65535) {
            desetinke = 9999;
        }
    }
}

void inicijalizacija(){
    // normalna način rada - timer 1
    timer1_set_normal_mode();
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    TCNT1 = 40536;
    interrupt_enable();
}

int main(void) {

    inicijalizacija();

    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);
    DigitalInput StartStop(D0); // PDO ulazni pin
```

```
    DigitalInput Clear(D1); // PD1 ulazni pin
    DigitalInput Direction(F6); // PF6 ulazni pin

    StartStop.pullup_on(); //pull up na P0
    Clear.pullup_on(); //pull up na PD1
    Direction.pullup_on(); //pull up na PF6

    bool counting = false;
    uint8_t d1 = 0, d2 = 0, d3 = 0, d4 = 0;
    // slanje četiri bajta na posmačne registre
    SBR.send_4byte(digits[d1], digits[d2], digits[d3] | DOT, digits[d4]);

    while(1) {
        // na padajući brid tipkala StartStop
        if(StartStop.isFalling_edge()) {
            if (counting) {
                counting = false;
                timer1_interrupt_OVF_disable();
            }
            else {
                counting = true;
                timer1_interrupt_OVF_enable();
            }
        }
        // na padajući brid tipkala Direction
        if(Direction.isFalling_edge()) {
            if (count_up) {
                count_up = false;
            }
            else {
                count_up = true;
            }
        }
        // na padajući brid tipkala Clear
        if(Clear.isFalling_edge()) {
            desetinke = 0;
            d1 = d2 = d3 = d4 = 0;
            SBR.send_4byte(digits[d1], digits[d2], digits[d3]
                | DOT, digits[d4]);
        }
        // ako je tajmer generirao novih 100 ms, prikaži ih
        if (new_time_ready) {
            // računanje znamenaka
            d1 = desetinke / 1000;
            d2 = (desetinke / 100) % 10;
            d3 = (desetinke / 10) % 10;
            d4 = desetinke % 10;

            SBR.send_4byte(digits[d1], digits[d2], digits[d3]
                | DOT, digits[d4]);
            new_time_ready = false;
        }
    }
    return 0;
}
```

Zatvorite datoteku vjezba913.cpp i onemogućite prevođenje ove datoteke.



### Vježba 9.1.4

Napravite program koji će na numeričkim displejima ispisati napon na potenciometru spojenom na pin ADC5. Napon ispišite na tri decimalna mjesta. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

U projektnom stablu otvorite datoteku `vjezba914.cpp`. Omogućite samo prevođenje datoteke `vjezba914.cpp`. Početni sadržaj datoteke `vjezba914.cpp` prikazan je programskim kodom 9.10.

Programski kod 9.10: Početni sadržaj datoteke `vjezba914.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"
#include "SSD i SBR/ssd_sbr.h"
#include "ADC/adc.h"

void inicijalizacija(){
    adc_init(); // inicijalizacija AD pretvorbe
}

int main(void) {

    inicijalizacija();
    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);

    uint8_t d1, d2, d3, d4;
    uint16_t napon_mV;

    while(1) {

        napon_mV = adc_read(ADC5) * 5000.0 / 1023.0;
        // računanje znamenaka za prikaz
        d1 = napon_mV / 1000;
        d2 = (napon_mV / 100) % 10;
        d3 = (napon_mV / 10) % 10;
        d4 = napon_mV % 10;
        // slanje četiri bajta na posmačne registre
        SBR.send_4byte(digits[d1] | DOT, digits[d2], digits[d3], digits[d4]);
        _delay_ms(200);
    }
    return 0;
}
```

U ovoj vježbi potrebno je prezentirati napon potenciometra na pinu ADC5 pomoću numeričkih displeja. U programski kod 9.10 u funkciji `inicijalizacija()` pozovite funkciju `adc_init()` kojom ćete inicijalizirati analogno-digitalnu pretvorbu.

Napon koji ćemo prikazivati na numeričkim displejima mora biti prezentiran na tri decimalna mjesta. Iz realnog broja moguće je izvući znamenke prije i poslije decimalne točke. U tom smislu, napon u V možemo pomnožiti s 1000 i dobiti četveroznamenkasti cijeli broj, što će predstavljati napon u mV. Taj cijeli broj lako je rastaviti na znamenke tisućica, stotica, desetica i jedinica. Budući da znamo da smo izračunali napon u mV, decimalnu ćemo točku postaviti nakon prve znamenke kako bi prezentirani napon bio u V.

Analogno-digitalnom pretvorbom na pinu ADC5 dobit ćemo broj iz cjelobrojnog intervala [0, 1023]. Taj broj treba skalirati na realni interval [0, 5,0] V. Ako prethodni realni interval

pomnožimo s 1000, dobit ćemo cjelobrojni interval [0, 5000] mV. Relacija za izračun intervala [0, 5000] mV jest:

$$U_{ADC5} = \frac{ADC_5}{1023} \cdot 5000. \quad (9.1)$$

U relaciji (9.1) dobili smo napon u mV. Želimo li prezentirati napon u V na numeričkim displejima, na rezultatu pretvorbe prema relaciji (9.1) decimalnu točku pomičemo za tri mjesta ulijevo. Prema tome, kako smo već spomenuli, decimalna točka dolazi iza prve znamenke (`digits[d1] | DOT`). Relacija (9.1) implementirana je u `while` petlji, nakon čega su izračunate znamenke napona i poslane na numeričke displeje posredno preko posmačnih registara.

Prevedite datoteku `vjezba914.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba914.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 9.1.5

Napravite program u kojem ćete na numeričke displeje ispisivati “25°C”, “ AH ” i “-12.3” naizmjenice svakih 1000 ms. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

U projektnom stablu otvorite datoteku `vjezba915.cpp`. Omogućite samo prevođenje datoteke `vjezba915.cpp`. Početni sadržaj datoteke `vjezba915.cpp` prikazan je programskim kodom 9.11.

Programski kod 9.11: Početni sadržaj datoteke `vjezba915.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"
#include "SSD i SBR/ssd_sbr.h"

int main(void) {
    // stvaranje objekta za posmačni registar
    Shift_Bit_Register SBR(C7, F1, F0);

    while(1) {
        // slanje definiranih znakova na numeričke displeje
        SBR.send_4byte(DIGIT_2, DIGIT_5, BIG_DEEGRE, CHAR_C);
        _delay_ms(1000);
        SBR.send_4byte(EMPTY_SPACE, CHAR_A, CHAR_H, EMPTY_SPACE);
        _delay_ms(1000);
        SBR.send_4byte(MINUS, DIGIT_1, DIGIT_2 | DOT, DIGIT_3);
        _delay_ms(1000);
    }
    return 0;
}
```

Numerički displeji osim brojeva mogu prezentirati i druge znakove, sukladno segmentima koji su uključeni ili isključeni. U zaglavlju `ssd_sbr.h` definirane su konstante s korisnim znakovima koji se mogu prezentirati na numeričkom displeju. U ovoj vježbi korišteni su znakovi:

- `DIGIT_2` - prikaz broja 2, što je ekvivalent `digits[2]`,
- `DIGIT_5` - prikaz broja 5, što je ekvivalent `digits[5]`,
- `BIG_DEEGRE` - prikaz stupnjeva Celzijusevih,
- `CHAR_C` - prikaz slova C,

- `EMPTY_SPACE` - prikaz praznog mjesta,
- `CHAR_A` - prikaz slova A,
- `CHAR_H` - prikaz slova H,
- `MINUS` - prikaz predznaka "-".

Navedeni se znakovi šalju na posmačne registre kako je to prikazano u programskom kodu 9.10. Prevedite datoteku `vjezba915.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Proizvoljno u programski kod 9.10 dodajte prikaz novih znakova koji su definirani u zaglavlju `ssd_sbr.h`. Prevedite datoteku `vjezba915.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba915.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

## 9.2 Zadaci - numerički displej

### Zadatak 9.2.1

Napravite program pomoću kojeg ćete na sva četiri posmačna registra postaviti 8-bitni podatak 0x9C. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

---

### Zadatak 9.2.2

Napravite program u kojem ćete na numeričkim displejima izmjenjivati znakove `AudI` i `A3` svakih 1000 ms. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

---

### Zadatak 9.2.3

Napravite program u kojim ćete realizirati štopericu na preciznost stotinke sekunde. Štoperica se pokreće i zaustavlja tipkalom spojenim na pin PD0. Resetiranje štoperice na početno stanje 00.00 potrebno je omogućiti pomoću tipkala spojenog na pin PD1. Maksimalno vrijeme koje štoperica može prikazati jest 49.99. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

---

### Zadatak 9.2.4

Napravite program koji će prezentirati otpor potencijometra spojenog na pin ADC5 u k $\Omega$  na dva decimalna mjesta (nazivni otpor potencijometra jest 10 k $\Omega$ ). Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

---

** Zadatak 9.2.5**

Napravite program u kojem ćete na numeričke displeje ispisivati “18°C”, “B B” i “-25.9” naizmjenice svakih 1000 ms. Shema spajanja četiriju posmačnih registara i numeričkih displeja na mikroupravljač ATmega32U4 prikazana je na slici 9.9.

---





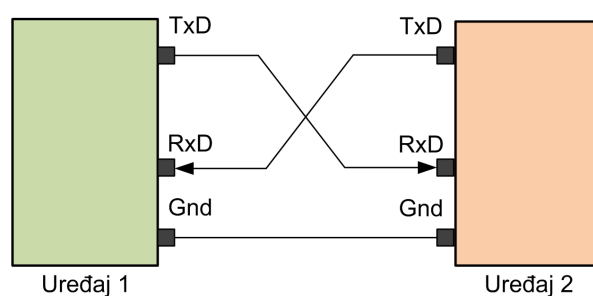
## Poglavlje 10

# Univerzalna asinkrona serijska komunikacija

Univerzalnu asinkronu serijsku komunikaciju omogućuje sklopovlje UART (engl. *Universal Asynchronous Receiver-Transmitter*). Ovo sklopovlje koristi dva pina:

- RxD - pin za primanje podataka koji je povezan s primateljom (engl. *receiver*),
- TxD - pin za slanje podataka koji je povezan s pošiljateljom (engl. *transmitter*).

Pomoću RxD i TxD pinova omogućena je dvosmjerna (engl. *full-duplex*) komunikacija, što znači da dva uređaja mogu istovremeno i slati i primiti podatke pomoću dviju žice. Razlog tome jest što sklopovlje UART ima odvojene registre za primanje i slanje podataka. Shema povezivanja dvaju uređaja sa sklopovljem UART prikazana je na slici 10.1. Princip spajanja uvijek je isti. RxD pin prvog uređaja spaja se na TxD pin drugog uređaja i obratno (slika 10.1). Mase (Gnd) dvaju uređaja koji komuniciraju moraju biti spojene zajedno kako bi uređaji imali isti referentni potencijal.



Slika 10.1: Shema povezivanja dvaju uređaja sa sklopovljem UART

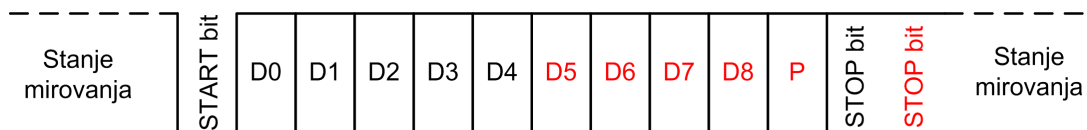
Komunikacija između dvaju sklopovlja UART asinkrona je, što znači da ne postoji izvor takta koji sinkronizira prijenos podataka. Zbog asinkrone serijske komunikacije dva uređaja sa sklopovljem UART koja razmjenjuju podatke moraju imati jednaku konfiguraciju parametara. Parametri sklopovlja UART jesu [4]:

- 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 10.2.



Slika 10.2: Podatkovni okvir kod univerzalne asinkrone serijske komunikacije

Kada se podaci ne šalju između dvaju uređaja, oba pina (RxD i TxD) u stanju su logičke jedinice. To stanje je 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 10.2 ovisi o brzini prijenosa podataka. Najčešće postavke univerzalne asinkrone serijske komunikacije jesu:

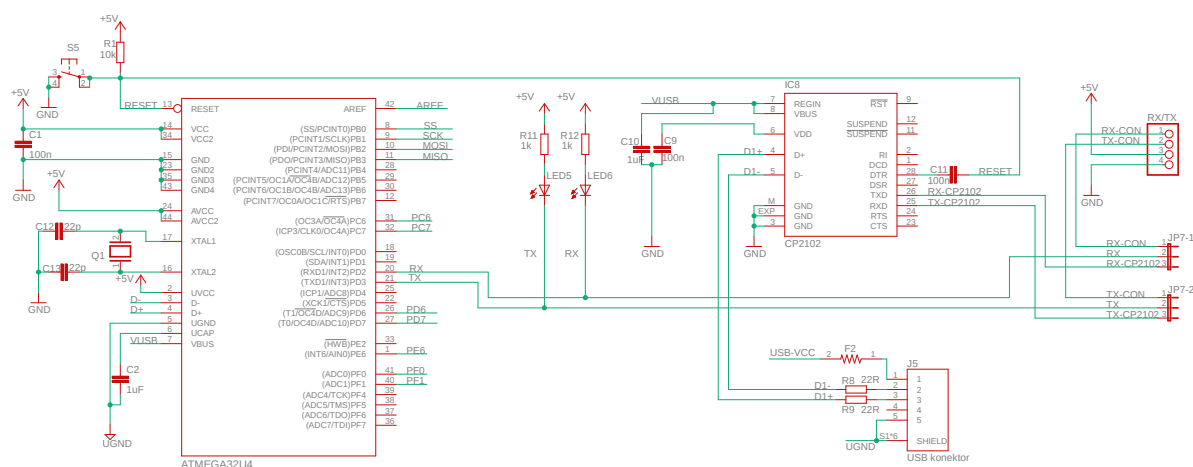
- brzina prijenosa podataka: prema potrebi i mogućnostima uređaja (npr. 19200),
- broj podatkovnih bitova: 8,
- paritetni bit: onemogućen,
- *Stop* bit: 1.

Kako bismo poslali 1 B (8 bitova) podataka, potrebno je poslati ukupno 10 bitova (1 *Start* bit + 8 bitova podataka + 1 *Stop* bit = 10 bitova). Ako je brzina prijenosa 19200 b/s, tada se u jednoj sekundi može poslati 1920 B podataka ( $(19200 \text{ b/s}) / (10 \text{ b/B}) = 1920 \text{ B}$ ). Zbog asinkronog načina prijenosa, dva uređaja koja razmjenjuju podatke moraju biti jednako konfigurirana. U suprotnom tumačenje podatka neće biti ispravno.

Sklopovlje UART definira samo asinkroni serijski komunikacijski protokol, ali ne i fizičke karakteristike uređaja koji komuniciraju. Na primjer, kod mikroupravljača se koristi TTL logika u kojoj je naponska razina logičke jedinice 5 V, a naponska razina logičke nule je 0 V. Kod računala se koristi standard RS232 u kojem je naponska razina logičke jedinice -12 V, a naponska razina logičke nule je 12 V. Kako bismo ostvarili komunikaciju između mikroupravljača i računala, potrebno je koristiti prilagodni međusklop koji će uskladiti naponske nivoe. Sklop koji se koristi u tu svrhu zove se MAX232. Danas računala nemaju integrirani RS232 standard i COM port pa se sklop MAX232 gotovo i ne koristi u praktičnoj primjeni. Češće se danas koriste integrirani krugovi (prilagodni međusklopovi) koji rade pretvorbu između USB ↔ TTL. Neki od poznatijih integriranih krugova koji omogućuju ovu pretvorbu jesu CP2102 i FT232. Naime, ove uređaje računalo promatra kao virtualni COM port koji radi na RS232 standardu.

## 10.1 Vježbe - univerzalna asinkrona serijska komunikacija

Mikroupravljač ATmega32U4 posjeduje sklopovlje USART (engl. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*). Ovo sklopovlje služi i za sinkronu i za asinkronu komunikaciju. U vježbama ćemo koristiti samo asinkronu komunikaciju. U mikroupravljaču ATmega32U4 sklopovlje USART kompatibilno je sa sklopovljem UART koje smo prethodno opisali. Podaci se na mikroupravljač ATmega32U4 asinkronom serijskom komunikacijom primaju pomoću pina RXD1 (PD2), a šalju pomoću pina TXD1 (PD3). Ukoliko na mikroupravljaču ATmega32U4 koristite asinkronu serijsku komunikaciju, tada digitalne pinove PD2 i PD3 ne možete koristiti u druge svrhe. U svrhu vježbi, mikroupravljač ATmega32U4 spojit ćemo s računalom koristeći ili prilagodni međusklop CP2102 na razvojnom okruženju ili direktnim pristupom pinovima RXD1 i TXD1. Shema spajanja prilagodnog međusklopa CP2102 na mikroupravljač ATmega32U4 prikazana je na slici 10.3.



Slika 10.3: Shema spajanja prilagodnog međusklopa CP2102 na mikroupravljač ATmega32U4

Na shemi sa slike 10.3 UART komunikacija može se koristiti na dva načina:

1. način: posredno preko prilagodnog međusklopa CP2102. U tom slučaju kratkospojnici JP7-1 i JP7-2 moraju biti spojeni između trnova 2 i 3.
2. način: direktnim pristupom UART pinovima mikroupravljača RXD1 i TXD1. U tom slučaju kratkospojnici JP7-1 i JP7-2 moraju biti spojeni između trnova 1 i 2.

Kada se koristi prilagodni međusklop CP2102, tada se razvojno okruženje sa slike 3.1 na računalo povezuje mikro USB konektorom tipa B.

Podatak primljen preko RXD1 pina sprema se u međuspremnik primljenih podataka. Kada se podatak šalje preko TXD1 pina, on se neposredno prije slanja sprema u međuspremnik podataka za slanje. Sklopovlje USART u mikroupravljaču ATmega32U4 koristi istu memorijsku lokaciju registra za slanje i za primanje podataka. Ime registra je **UDR1**. Kada čitamo podatak iz registra **UDR1**, tada kao povratnu vrijednost dobijemo podatak iz međuspremnika primljenih podataka. Kada podatak zapisujemo u registar **UDR1**, tada se on prosljeđuje u međuspremnik podataka za slanje. Sklopovlje USART automatski šalje i prima podatke. Primljene podatke potrebno je na vrijeme pročitati iz registra **UDR1** jer postoji opasnost da se novi podatak prepíše preko staroga. Kada se na pinu RXD1 pojavi *Start* bit, mikroupravljač generira prekid. Na taj način novi podatak odmah možemo pročitati iz registra **UDR1**, bez obzira na to što se u glavnom programu izvodi dio programskog koda.

Rad sklopovlja USART konfigurira se pomoću registara **UCSR1A**, **UCSR1B** i **UCSR1C**. Konfiguracija se odnosi na broj podatkovnih bitova, paritetni bit, broj *Stop* bitova, omogućavanje prekida kada pristigne podatak i drugo. Podatkovni okvir koji podržava mikroupravljač ATmega32U4 isti je kao podatkovni okvir prikazan na slici 10.2. Više detalja o konfiguraciji registara **UCSR1A**, **UCSR1B** i **UCSR1C** pogledajte u literaturi [1] na stranicama 209 - 213.

Brzina prijenosa podataka konfigurira se pomoću registra **UBRR1**, a računa se prema relaciji [1]:

$$BAUD = \frac{F\_CPU}{16(UBRR1 + 1)}. \quad (10.1)$$

Vrijednost registra **UBRR1** za frekvenciju radnog takta  $F\_CPU$  i brzinu prijenosa  $BAUD$  može se izračunati iz relacije (10.1) na sljedeći način:

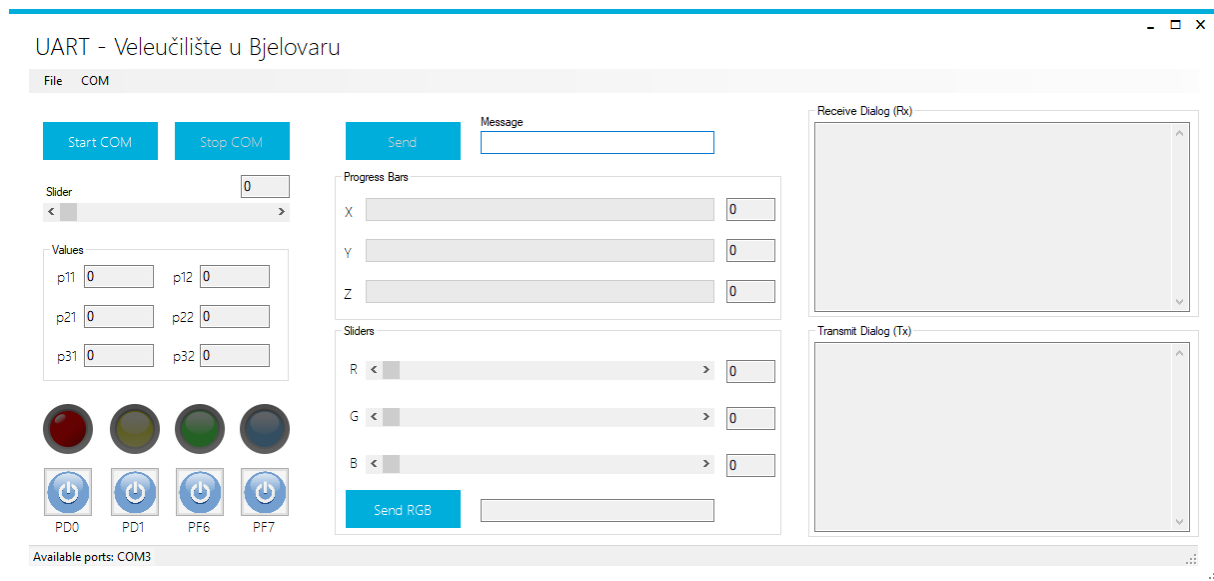
$$UBRR1 = \frac{F\_CPU}{16 \cdot BAUD} - 1. \quad (10.2)$$

Rezultat dobiven pomoću relacije (10.2) mora biti cijeli broj jer će se u suprotnom javljati pogreška pri prijenosu podataka. U tablicama 18-4 do 18-6 u literaturi [1] nalaze se vrijednosti registra **UBRR1** za standardne brzine prijenosa podataka<sup>1</sup> i standardne frekvencije radnog takta te postotna pogreška u prijenosu podataka. Frekvencija radnog takta koju mi koristimo u vježbama jest 16 MHz. Ako odaberemo brzinu prijenosa podataka od 19200 b/s, javit će se pogreška u prijenosu podataka od 0,2 %<sup>2</sup>. Za pogrešku u prijenosu podataka od 0,0 % za standardne brzine prijenosa preporučuje se korištenje vanjskog izvora radnog takta od 7,3728 MHz, 11,0592 MHz, 14,7456 MHz i 18,4320 MHz.

Rad asinkrone serijske komunikacije testirat ćemo pomoću aplikacija za upravljanje i nadzor razvojnog okruženja s mikroupravljačem ATmega32U4. S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku *Serijska komunikacija - aplikacija.zip* u kojoj se nalazi instalacija aplikacije. Datoteku *Serijska komunikacija - aplikacija.zip* raspakirajte na radnu površinu u datoteku datoteke *UART VUB*. Aplikaciju pokrenite tako da pokrenete \*.exe datoteku *UART*. Početni prozor aplikacije prikazan je na slici 10.4.

<sup>1</sup>Standardne brzine prijenosa: 2400 b/s, 4800 b/s, 9600 b/s, 19200 b/s, 14,4 kb/s, 19,2 kb/s, 28,8 kb/s, 38,4kb/s, 57,6 kb/s, 76,8 kb/s, 115,2 kb/s, ...

<sup>2</sup>Ova je greška zanemariva kada se šalju mali paketi poruka (male tekstualne poruke), što je uobičajeno kod mikroupravljača.



Slika 10.4: Aplikacija za upravljanje i nadzor razvojnog okruženja s mikroupravljačem ATmega32U4 pomoću serijske komunikacije

Aplikacija omogućuje interakciju s razvojnim okruženjem sa slike 3.1. Može slati i primati tekstualne poruke. Aplikacija će na razvojno okruženje slati poruke zadanog sadržaja i za sljedeće akcije:

- Prvim pritiskom na tipku PD0 aplikacija pomoću serijskog porta šalje poruku "B41\r", sljedećim pritiskom šalje poruku "B40\r" itd. Poruka "B41\r" uključuje crveno svjetlo u aplikaciji, a poruka "B40\r" isključuje crveno svjetlo u aplikaciji.
- Prvim pritiskom na tipku PD1 aplikacija pomoću serijskog porta šalje poruku "B51\r", sljedećim pritiskom šalje poruku "B50\r" itd. Poruka "B51\r" uključuje žuto svjetlo u aplikaciji, a poruka "B50\r" isključuje žuto svjetlo u aplikaciji.
- Prvim pritiskom na tipku PF6 aplikacija pomoću serijskog porta šalje poruku "B61\r", sljedećim pritiskom šalje poruku "B60\r" itd. Poruka "B61\r" uključuje zeleno svjetlo u aplikaciji, a poruka "B60\r" isključuje zeleno svjetlo u aplikaciji.
- Prvim pritiskom na tipku PF7 aplikacija pomoću serijskog porta šalje poruku "B71\r", sljedećim pritiskom šalje poruku "B70\r" itd. Poruka "B71\r" uključuje plavo svjetlo u aplikaciji, a poruka "B70\r" isključuje plavo svjetlo u aplikaciji.
- Pritiskom na tipku Send aplikacija će pomoću serijskog porta poslati poruku koja se nalazi u tekst okviru Message. Na primjer, ako u tekst okviru Message piše "VUB MEH i RAC", aplikacija će pomoću serijskog porta poslati poruku "VUB MEH i RAC\r".
- Promjenom stanja klizača Slider čiji je raspon vrijednosti cijelih brojeva [0, 255], aplikacija će pomoću serijskog porta poslati poruku formata "S%u\*\r". U navedenoj poruci:
  - znak 'S' označava da se šalje cjelobrojni podatak s klizača.
  - kvalifikator "%u" je cjelobrojna vrijednost klizača Slider.
  - znakovi "\*\r" su zaključni znakovi kako bismo mogli detektirati kraj broja i kraj poruke.

Na primjer, ako stanje klizača Slider iznosi 155, aplikacija će pomoću serijskog porta poslati poruku "S155\*\r".

- Pritiskom na tipku **Send RGB** aplikacija će pomoću serijskog porta poslati poruku formata "**R%u;%u;%u\*\r**". U navedenoj poruci:
  - znak '**R**' označava da se šalju tri cjelobrojne vrijednosti namijenjene za RGB diodu.
  - prvi kvalifikator "**%u**" cjelobrojna je vrijednost klizača **R** čiji je raspon vrijednosti cijelih brojeva [0, 100], a koristi se za upravljanjem intenziteta crvene diode na RGB diodi.
  - prvi znak '**;**' separator je prvog i drugog broja.
  - drugi kvalifikator "**%u**" cjelobrojna je vrijednost klizača **G** čiji je raspon vrijednosti cijelih brojeva [0, 100], a koristi se za upravljanjem intenziteta zelene diode na RGB diodi.
  - drugi znak '**;**' separator je drugog i trećeg broja.
  - treći kvalifikator "**%u**" cjelobrojna je vrijednost klizača **B** čiji je raspon vrijednosti cijelih brojeva [0, 100], a koristi se za upravljanjem intenziteta plave diode na RGB diodi.
  - znakovi "**\*\r**" zaključni su znakovi kako bismo mogli detektirati kraj trećeg broja i kraj poruke.

Na primjer, ako stanje klizača **R** iznosi 23, stanje klizača **G** iznosi 172, a stanje klizača **B** iznosi 250, aplikacija će pomoću serijskog porta poslati poruku "**R23;172;250\*\r**".

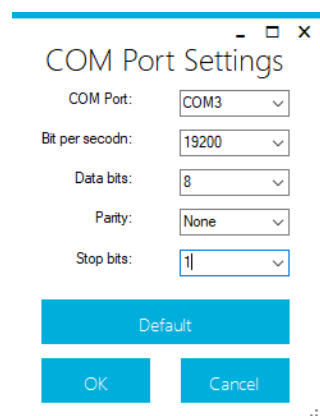
Sigurno ste primijetili da svaka poruka završava specijalnim znakom "**\r**". Radi se o znaku *Carriage Return* (ASCII kod 0x0D), a koristi se kao zaključni znak svake tekstualne poruke kako bi mikroupravljač mogao detektirati kraj poruke. U programu mikroupravljača znak *Carriage Return* bit će potrebno navesti kao zaključni znak.

Ukupan promet podataka serijskim portom prikazuje se u tekstualnim okvirima **Receive Dialog (Rx)** za dolazne poruke i **Transmit Dialog (Tx)** za odlazne poruke. Aplikacija će obraditi primljene poruke zadanog sadržaja na sljedeći način:

- ako na aplikaciju dođe poruka "**B41\*\r**", uključuje se crveno svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B40\*\r**", isključuje se crveno svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B51\*\r**", uključuje se žuto svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B50\*\r**", isključuje se žuto svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B61\*\r**", uključuje se zeleno svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B60\*\r**", isključuje se zeleno svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B71\*\r**", uključuje se plavo svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka "**B80\*\r**", isključuje se plavo svjetlo u aplikaciji,
- ako na aplikaciju dođe poruka formata "**A;%u;%u;%u\*\r**", tada u navedenoj poruci:
  - znak '**A**' označava da se primaju tri cjelobrojne vrijednosti namijenjene za trake za prikaz napretka **X**, **Y**, **Z**.
  - prvi kvalifikator "**%u**" cjelobrojna je vrijednost trake za prikaz napretka **X** čiji je raspon vrijednosti cijelih brojeva [0, 100].
  - prvi znak '**;**' separator je prvog i drugog broja.
  - drugi kvalifikator "**%u**" cjelobrojna je vrijednost trake za prikaz napretka **Y** čiji je raspon vrijednosti cijelih brojeva [0, 100].

- drugi znak ';' separator je drugog i trećeg broja.
- treći kvalifikator "%u" cjelobrojna je vrijednost trake za prikaz napretka Z čiji je raspon vrijednosti cijelih brojeva [0, 100].
- znakovi "\*\r" zaključni su znakovi kako bismo mogli detektirati kraj poruke.
- ako na aplikaciju dođe poruka formata "X;%u\*\r", tada se cjelobrojna vrijednost "%u" postavlja na traku za prikaz napretka X.
- ako na aplikaciju dođe poruka formata "Y;%u\*\r", tada se cjelobrojna vrijednost "%u" postavlja na traku za prikaz napretka Y.
- ako na aplikaciju dođe poruka formata "Z;%u\*\r", tada se cjelobrojna vrijednost "%u" postavlja na traku za prikaz napretka Z.
- ako na aplikaciju dođe poruka formata "P;%u;%u;%u;%u;%u;%u\*\r", tada u navedenoj poruci:
  - znak 'P' označava da se prima šest cjelobrojnih vrijednosti namijenjenih za prikaz u polju podataka Values.
  - prvi kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{11}$ .
  - prvi znak ';' separator je prvog i drugog broja.
  - drugi kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{21}$ .
  - drugi znak ';' separator je drugog i trećeg broja.
  - treći kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{31}$ .
  - treći znak ';' separator je trećeg i četvrtog broja.
  - četvrti kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{12}$ .
  - četvrti znak ';' separator je četvrtog i petog broja.
  - peti kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{22}$ .
  - peti znak ';' separator je petog i šestog broja.
  - šesti kvalifikator "%u" cjelobrojna je vrijednost koja se prikazuje u polju  $p_{32}$ .
  - znakovi "\*\r" zaključni su znakovi kako bismo mogli detektirati kraj poruke.

Sve navedene poruke aplikacija sa slike 10.4 obrađuje svakih 50 ms. Preporuka je da se dvije poruke prema aplikaciji šalju u vremenskom razmaku od najmanje 50 ms. Konfiguracija serijske komunikacije na računalo može se postaviti pomoću prozora prikazanog na slici 10.5. Ovaj prozor otvorit će se tako da u aplikaciji odaberete COM → Edit.



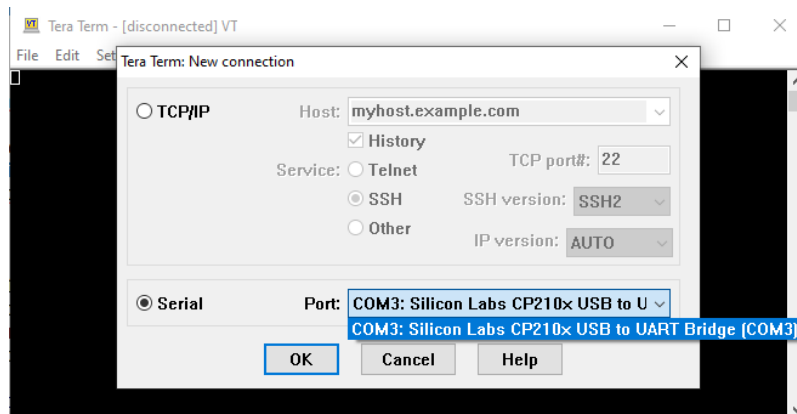
Slika 10.5: Konfiguracija serijske komunikacije na računalo

Postavke mikroupravljača u sljedećim vježbama bit će sljedeće:

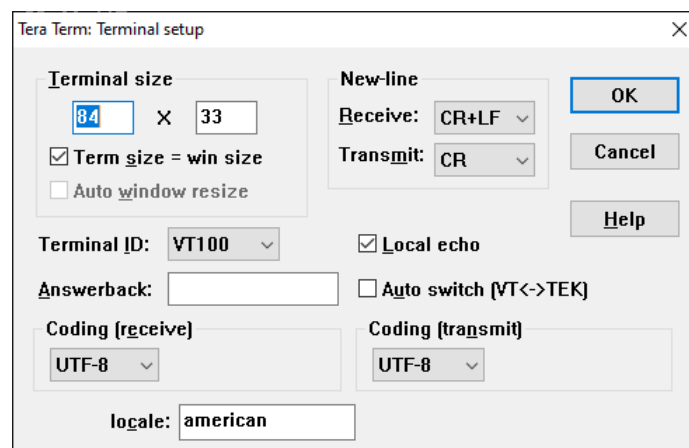
- brzina prijenosa podataka: prema potrebi,
- broj podatkovnih bitova: 8,
- paritetni bit: onemogućen,
- *Stop* bit: 1.

U prozoru sa slike 10.5 bit će potrebno mijenjati samo brzinu prijenosa podataka i COM Port koji se koristi za serijsku komunikaciju. Dostupni COM portovi prikazuju se u podnožju aplikacije na slici 10.4. Nakon što se konfigurira serijska komunikacija na računalu, potrebno je otvoriti serijski port za komunikaciju klikom miša na tipku **Start COM**. Ako je serijski port uspješno otvoren, aplikacija je spremna za komunikaciju s vanjskim uređajem.

U inženjerskoj praksi često se koriste terminali za testiranje serijske komunikacije. Jedan takav terminal jest *Tera Term* koji možete preuzeti sa stranice <https://ttssh2.osdn.jp/>. Pri pokretanju aplikacije *Tera Term* pojavit će se prozor sa slike 10.6. Kao vrstu nove komunikacije potrebno je odabrati serijsku komunikaciju (**Serial**). Za serijsku komunikaciju padajući izbornik prikazuje sve dostupne COM portove na računalu. Odaberite onaj port koji u svom opisu ima oznaku prilagodnog međusklopa CP2102. Nakon što je pokrenuta nova komunikacija (otvoren je COM port), potrebno je namjestiti postavke terminala tako da budu jednake postavkama na slici 10.7. Prozor sa slike 10.7 otvara se tako da u izborniku **Setup** odaberete podizbornik **Terminal**....



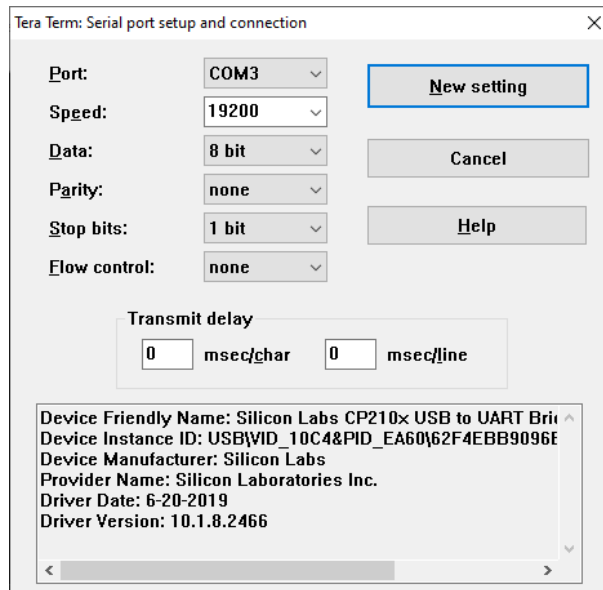
Slika 10.6: Pokretanje aplikacije (terminala) *Tera Term*



Slika 10.7: Podešavanje terminala *Tera Term*

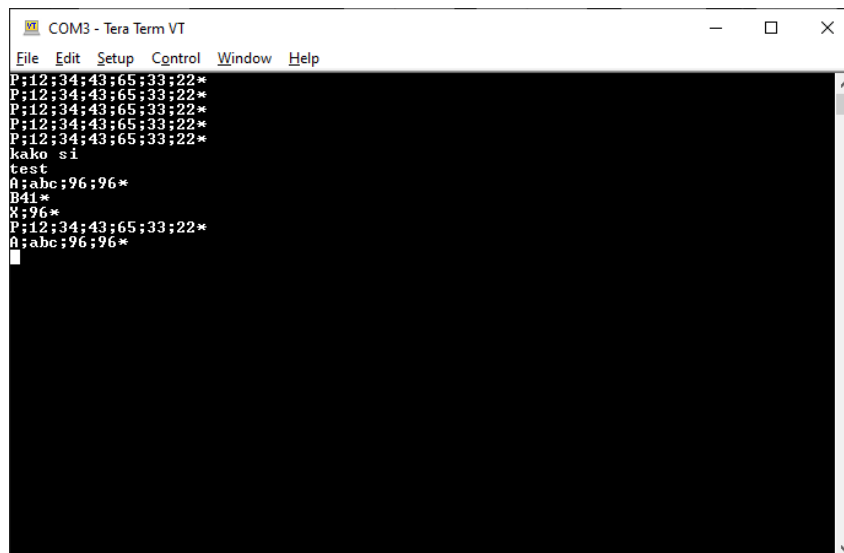


Kako smo već spomenuli, postavke serijske komunikacije moraju biti jednake na uređajima koji međusobno komuniciraju. Podešavanje postavki serijskog porta COM3 u terminalu *Tera Term* prikazano je na slici 10.9.



Slika 10.8: Podešavanje postavki serijskog porta COM3 u terminalu *Tera Term*

Prozor sa slike 10.8 otvara se tako da u izborniku **Setup** odaberete podizbornik **Serial port...** Konačno, praćenje komunikacije u terminalu *Tera Term* prikazano je na slici 10.9. Korištenje terminala vrlo je korisno u razvoju programskih rješenja mikroupravljača zasnovanih na serijskoj komunikaciji zbog bržeg i jednostavnijeg testiranja.



Slika 10.9: Praćenje komunikacije u terminalu *Tera Term*

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) 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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se **Ivica Ivic**. Datoteku **UART.zip** raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite **VUB**

mikroracunala.atsln u datoteci \\UART\vjezbe. U otvorenom projektu nalaze se sve 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 10.1.1

Napravite program kojim ćete slati i primiti proizvoljne tekstualne poruke UART komunikacijom mikroupravljača. U programu mikroupravljača definirajte dvije proizvoljne tekstualne poruke koje ćete slati pritiskom na tipkala spojena na pinove PD0 i PD1. Na računalu je za slanje i primanje tekstualnih poruka potrebno koristiti terminal *Tera Term*. Poslane poruke iz terminala potrebno je prikazivati na LCD displeju. Znakovni niz koji šaljemo na serijski port putem terminala zaključen je *Carriage Return* znakom ('`\r`') (vidi postavke na slici 10.7) kako bi se u mikroupravljaču detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku vjezba1011.cpp. Omogućite samo prevođenje datoteke vjezba1011.cpp. Početni sadržaj datoteke vjezba1011.cpp prikazan je programskim kodom 10.1.

Programski kod 10.1: Početni sadržaj datoteke vjezba1011.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"

void inicijalizacija() {
    uart_init(19200); //inicijalizacija serijske komunikacije
    lcd_init(); // konfiguriranje LCD displeja
    interrupt_enable(); //omogući globalni prekid
}

int main(void) {

    inicijalizacija();
    // stvaranje objekata za tipkala
    DigitalInput Tipkalo_D0(D0); // pin PD0
    DigitalInput Tipkalo_D1(D1); // pin PD1
    Tipkalo_D0.pullup_on(); // pritezni otpornik na pinu PD0
    Tipkalo_D1.pullup_on(); // pritezni otpornik na pinu PD1

    while(1) {
        // provjera primljenih poruka
        if(uart_read_all() == true) {
            lcd_clrscr();
            lcd_home();
            lcd_print("%s", uart_buffer);
        }
        // provjera padajućeg brida tipkala na PD0
        if(Tipkalo_D0.isFalling_edge()) {
            uart_print("Serijska komunikacija je uspostavljena\r");
        }
        // provjera padajućeg brida tipkala na PD1
        if(Tipkalo_D1.isFalling_edge()) {
```

```

        uart_print("Baud rate je %u\r", 19200);
    }
}
return 0;
}

```

Funkcije koje se koriste za asinkronu serijsku komunikaciju definirane su u zaglavlju `uart.h`. Naredba kojom uključujemo zaglavlje `uart.h` u datoteku koja se prevodi jest `#include "UART/uart.h"`.

U programskom kodu 10.1 u funkciji `inicijalizacija()` nalazi se funkcija `uart_init(uint32_t baud_rate)` za konfiguriranje asinkrone serijske komunikacije mikroupravljača. Ova funkcija kao argument prima brzinu prijenosa podataka. Osim brzine prijenosa podataka, funkcija `uart_init(uint32_t)` postavlja sljedeće parametre:

- broj podatkovnih bitova: 8,
- paritetni bit: onemogućen,
- *Stop* bit: 1,
- prekid za pristigle poruke: omogućen,
- pinovi RXD1 i TXD1: omogućeni.

Brzina prijenosa podataka u programskom kodu 10.1 postavljena je na 19200 b/s. Poruke ćemo na mikroupravljač slati iz terminala *Tera Term*, a šalju se znak po znak. Na kraju poruke postavlja se zaključni znak koji označava kraj pristigle poruke. Registar `UDR1` širine je 8 bitova i u njega stane samo jedan znak (podatak od 8 bitova). Postavlja se pitanje kako primiti cijelu tekstualnu poruku koja može biti proizvoljno dugačka. U zaglavlju `uart.h` deklariran je niz znakova `uart_buffer` koji predstavlja međuspremnik znakova (engl. *buffer*) koji se puni pomoću registra `UDR1`. Parametri za međuspremnik znakova `uart_buffer` i deklaracija niza znakova `uart_buffer` prikazani su u programskom kodu 10.2.

Programski kod 10.2: Parametri za međuspremnik znakova `uart_buffer`

```

#define end_char '\r' // zaključni znak -> znak '\r' je
                       // Carriage Return (CR) ASCII kod 0x0D
#define MESSAGE_LENGTH 50 // maksimalna duljina poruke
char uart_buffer[MESSAGE_LENGTH + 1]; // spremnik za dolazne poruke

```

Parametri u programskom kodu 10.2 redom su:

- `end_char` - definirana konstanta koja predstavlja zaključni znak. Aplikacija sa slike 10.4 te terminal *Tera Term* s postavkama na slici 10.7 kao zaključni znak šalje *Carriage Return* znak s ASCII kodom 0x0D (specijalni znak u programskom jeziku C/C++ `'\r'`). Zaključni znak najčešće je znak koji se ne koristi u kreiranju tekstualne poruke (npr. `'\r'`, `'\n'`, `'*'`, `','`, `':'`).
- `MESSAGE_LENGTH` - definirana konstanta koja predstavlja maksimalnu duljinu tekstualne poruke koju ćete poslati na mikroupravljač i spremiti u međuspremnik znakova `uart_buffer`. Vrijednost se mijenja prema potrebi.
- `uart_buffer` - međuspremnik znakova koji se puni pomoću niza znakova koji pristižu u registar `UDR1`. Kada registar `UDR1` primi zaključni znak (u ovom slučaju znak `'\r'` s ASCII kodom 0x0D), međuspremnik znakova zaključuje se tzv. *null* znakom<sup>3</sup>.

<sup>3</sup>U programskom jeziku C/C++ niz znakova mora se zaključiti se *null* znakom kako bi prevoditelj znao gdje je kraj znakovnog niza.

Ako serijskim portom pristiže niz znakova "A;123;435\*\rB41\*\r", on će predstavljati dvije poruke. Prva poruka koja će biti zapisana u `uart_buffer` jest "A;123;435\*", a druga je "B41\*". Primijetite kako zaključni znak '\r' nije dio poruke koja se koristi u mikroupravljaču.

Prekidna rutina koja se poziva kada novi znak preko pina RXD1 dolazi na mikroupravljač zove se `USART1_RX_vect`. Međuspremnik znakova uvijek čuva zadnju pristiglu poruku. Kada u registar `UDR1` dođe znak koji predstavlja početak nove poruke, on se u prekidnoj rutini `USART1_RX_vect` sprema na memorijsku lokaciju `uart_buffer[0]`. Sljedeći se znak sprema na memorijsku lokaciju `uart_buffer[1]` i tako redom dok ne dođe zaključni znak. Prekidna rutina `USART1_RX_vect` napisana je u datoteci `uart.h` i preporučuje se njezino korištenje bez izmjene programskog koda prekidne rutine. Programer mikroupravljača mijenja samo parametre prikazane programskim kodom 10.2.

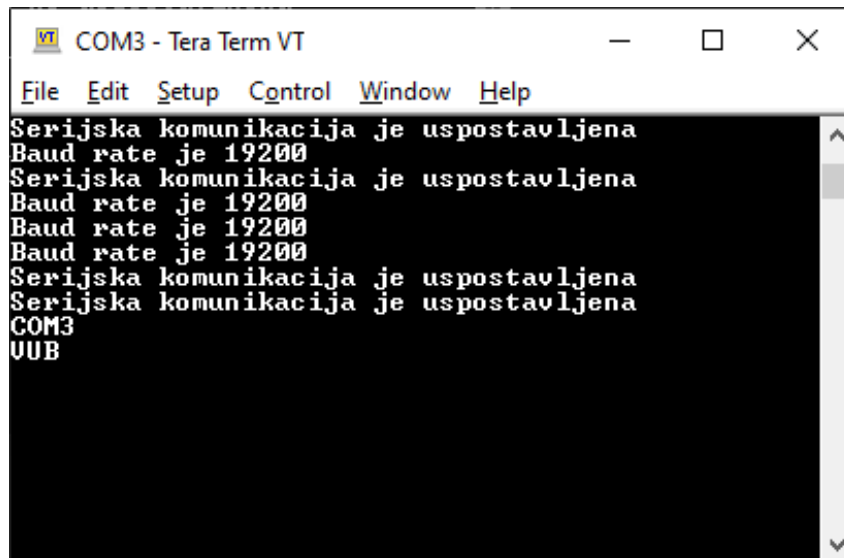
U programskom kodu 10.1 u funkciji `inicijalizacija()` konfiguriran je LCD displej i globalno su omogućeni prekidi funkcijom `interrupt_enable()`. U `main()` funkciji stvorena su dva objekta koja predstavljaju tipkala spojena na pinove PD0 i PD1.

U `while` petlji koristi se funkcija `uart_read_all()`. Ova funkcija vraća vrijednost `true` ako je dostupna nova poruka u međuspremniku znakova `uart_buffer`, a inače vraća vrijednost `false`. Funkcija `uart_read_all()` detektira novu poruku kada je pristigao zaključni znak definiran konstantom `end_char` (u našem slučaju to je specijalni znak '\r'). Kada funkcija `uart_read_all()` vrati vrijednost `true`, pristigla poruka u međuspremniku `uart_buffer` ispisuje se na LCD displej. Jasno je da poruke duže od 16 znakova neće biti potpuno prikazane na LCD displeju.

Na padajući brid signala tipkala spojenih na pinove PD0 i PD1 mikroupravljač na serijski port (odnosno na pin TXD1) šalje tekstualne poruke pomoću funkcije `uart_print`. Argumenti funkcije `uart_print` jednaki su argumentima funkcije `printf` (također i argumentima funkcije `lcd_print`). Dakle, funkcija `uart_print` koristi se na jednak način kao i funkcije `printf` i `lcd_print`. Ako pritisnemo tipkalo spojeno na pin PD0, tada će se izvršiti naredba `uart_print("Serijska komunikacija je uspostavljena\r")`. Ako pritisnemo tipkalo spojeno na pin PD1, tada će se izvršiti naredba `uart_print("Baud rate je %u\r", 19200)`, a poruka koju će mikroupravljač poslati UART komunikacijom bit će "Baud rate je 19200\r". U ovom slučaju specijalni znak '\r' služi da prijemna strana (terminal ili aplikacija) zna da je stigao kraj poruke.

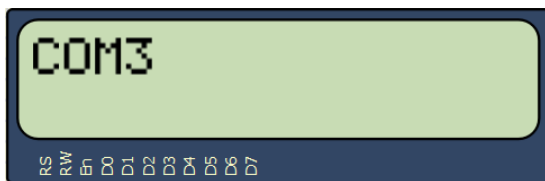
Prevedite datoteku `vjezba1011.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću terminala *Tera Term*. Namjestite parametre terminala i serijske komunikacije za COM port pomoću prozora sa slika 10.7 i 10.8. Na slici 10.10 prikazan je rezultat serijske komunikacije, pri čemu je redom:

- jednom pritisnuto tipkalo spojeno na pin PD0,
- jednom pritisnuto tipkalo spojeno na pin PD1,
- jednom pritisnuto tipkalo spojeno na pin PD0,
- tri puta pritisnuto tipkalo spojeno na pin PD1,
- dva puta pritisnuto tipkalo spojeno na pin PD0,
- pomoću terminala na mikroupravljač poslana poruka "COM3" i pritisnut *Enter* (terminal šalje poruku "COM3\r"),
- pomoću terminala na mikroupravljač poslana poruka "VUB" i pritisnut *Enter* (terminal šalje poruku "VUB\r").

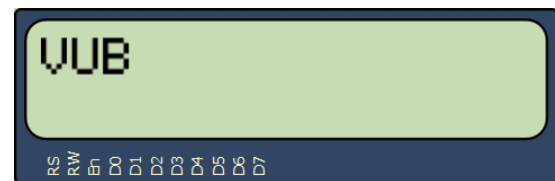


Slika 10.10: Praćenje komunikacije u terminalu *Tera Term* za programski kod 10.1

Poruke "COM3" i "VUB" prikazuju se na LCD displeju kako je prikazano na slici 10.11.



(a) Prva poruka ("COM3")



(b) Druga poruka ("VUB")

Slika 10.11: Prikaz pristiglih poruka iz terminala *Tera Term*

Dio programskog koda kojim ćete uvijek provjeravati nalazi li se u međuspremniku znakova `uart_buffer` nova poruka prikazan je programskim kodom 10.3. Ovaj dio koda uvijek mora biti u `while` petlji kako bi se neprestano provjeravalo je li dostupna nova poruka.

Programski kod 10.3: Izvođenje programskog koda na temelju pristigle poruke putem asinkrone serijske komunikacije

```
if(uart_read_all() == true){
// izvođenje programskog koda na temelju pristigle poruke
}
```

Pomoću asinkrone komunikacije poruke mogu razmjenjivati svi uređaji koji podržavaju sklopovlje UART. Na primjer, mikroupravljač može komunicirati s računalom, s GSM modemom, s drugim mikroupravljačem, s GPS modulom i drugim uređajima koji podržavaju sklopovlje UART.

Pokušajte kreirati druge proizvoljne poruke koje šaljete pomoću mikroupravljača. Prevedite datoteku `vjezba1011.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću terminala *Tera Term*.

Zatvorite datoteku `vjezba1011.cpp` i onemogućite prevođenje ove datoteke.



## Vježba 10.1.2

Napravite program kojim će se pomoću tipkala PD0, PD1, PF6 i PF7 u aplikaciji na slici 10.4 uključivati redom crvena LED dioda, žuta LED dioda, zelena LED dioda i plava LED dioda na razvojnom okruženju. Klikom miša na tipkalo u aplikaciji, putem serijske komunikacije, šalje se niz znakova u obliku "Bxy\r" (korisni dio poruke jest "Bxy"), gdje je:

- 'B' - niz znakova koji označuje port B,
- 'x' - znak koji određuje poziciju pina na portu B ('x' = '4', '5', '6', '7'),
- 'y' - znak koji određuje digitalno stanje pina ('y' = '0', '1'),
- '\r' - specijalan znak (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke.

Na primjer, ako je pristigla poruka "B41\r" (korisni dio poruke jest "B41"), crvenu LED diodu treba uključiti, a ako je pristigla poruka "B40\r" (korisni dio poruke jest "B40"), crvenu LED diodu treba isključiti. Brzinu prijenosa podataka postavite na 19200 b/s. Poslane poruke iz aplikacije potrebno je prikazivati na LCD displeju.

U projektnom stablu otvorite datoteku vjezba1012.cpp. Omogućite samo prevođenje datoteke vjezba1012.cpp. Početni sadržaj datoteke vjezba1012.cpp prikazan je programskim kodom 10.4.

Programski kod 10.4: Početni sadržaj datoteke vjezba1012.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"

void inicijalizacija() {
    uart_init(19200); //inicijalizacija serijske komunikacije
    lcd_init(); // konfiguriranje LCD displeja
    interrupt_enable(); //omogući globalni prekid
}

int main(void) {

    inicijalizacija();
    // stvaranje objekata za LED diode
    DigitalOutput LED_crvena(B4); // pin PB4 - crvena dioda
    DigitalOutput LED_zuta(B5); // pin PB5 - žuta dioda
    DigitalOutput LED_zelena(B6); // pin PB6 - zelena dioda
    DigitalOutput LED_plava(B7); // pin PB7 - plava dioda
    char B, x, y;

    while(1) {
        // provjera primljenih poruka
        if(uart_read_all() == true) {
            lcd_clrscr();
            lcd_home();
            lcd_print("%s", uart_buffer);
            B = uart_buffer[0];
            x = uart_buffer[1];
            y = uart_buffer[2];
        }
    }
}
```

```

// provjera prvog znaka
if (B == 'B') {
    // automat stanja koji obrađuje ostatak poruke tipa "xy"
    switch(x) {
        case '4':
            if (y == '1') LED_crvena.on();
            if (y == '0') LED_crvena.off();
            break;
        case '5':
            if (y == '1') LED_zuta.on();
            if (y == '0') LED_zuta.off();
            break;
        case '6':
            if (y == '1') LED_zelena.on();
            if (y == '0') LED_zelena.off();
            break;
        case '7':
            if (y == '1') LED_plava.on();
            if (y == '0') LED_plava.off();
            break;
        default:
            break;
    }
}
}
}
return 0;
}

```

Cilj vježbe jest uključiti digitalni izlaz temeljem pristigle poruke putem serijske komunikacije. Brzina prijenosa podataka u programskom kodu 10.4 postavljena je na 19200 b/s. Konfiguriran je LCD displej i globalno su omogućeni prekidi funkcijom `interrupt_enable()`. Aplikacija će serijskom komunikacijom slati tekstualne poruke oblika `"Bxy\r"`. Ova poruka šalje se na način da se svaki znak šalje zasebno i dodatno se na kraju poruke postavlja zaključni znak `"\r"` koji označava kraj pristigle poruke.

U programskom kodu 10.4 u `main()` funkciji stvoreni su objekti tipa `DigitalOutput` koji su povezani s pinovima PB4, PB5, PB6 i PB7 na kojima su spojene redom crvena, žuta, zelena i plava LED dioda.

U `while` petlji koristi se funkcija `uart_read_all()` za provjeru pristiglih poruka putem serijske komunikacije. Kada je pristigla nova poruka, ona se ispisuje na LCD displeju na način da ispisujemo sadržaj međuspremnika `uart_buffer`. Poruka koja se iz aplikacije šalje na mikroupravljač ima oblik `"Bxy\r"`. Znak `"\r"` specijalni je znak koji se koristi za detekciju kraja poruke, što znači da korisni dio poruke koja se nalazi u međuspremniku `uart_buffer` ima oblik `"Bxy"`. Pristiglu poruku iz međuspremnika `uart_buffer` raspakirali smo na pojedinačne znakove:

- `B = uart_buffer[0]` - znak koji definira kojem se portu pristupa (očekujemo znak `'B'`),
- `x = uart_buffer[1]` - znak koji određuje poziciju pina na portu (očekujemo znakove `'4'`, `'5'`, `'6'`, `'7'`),
- `y = uart_buffer[2]` - znak koji određuje digitalno stanje pina (očekujemo znakove `'0'`, `'1'`).

Ako se u varijabli `B` nalazi znak `'B'`, tada se izvodi takozvani automat stanja (standardni `switch case`) koji provjerava varijablu `x` na sljedeći način:

- ako je varijabla `x` jednaka znaku `'4'`, tada će se:

- ako je varijabla y jednaka znaku '1', uključiti crvena LED dioda,
- ako je varijabla y jednaka znaku '0', isključiti crvena LED dioda.
- ako je varijabla x jednaka znaku '5', tada će se:
  - ako je varijabla y jednaka znaku '1', uključiti žuta LED dioda,
  - ako je varijabla y jednaka znaku '0', isključiti žuta LED dioda.
- ako je varijabla x jednaka znaku '6', tada će se:
  - ako je varijabla y jednaka znaku '1', uključiti zelena LED dioda,
  - ako je varijabla y jednaka znaku '0', isključiti zelena LED dioda.
- ako je varijabla x jednaka znaku '7', tada će se:
  - ako je varijabla y jednaka znaku '1', uključiti plava LED dioda,
  - ako je varijabla y jednaka znaku '0', isključiti plava LED dioda.

Ako se niti jedan uvjet ne zadovolji, to znači da je poruka koja je pristigla neispravna. Prevedite datoteku `vjezba1012.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za onaj COM Port na kojem se nalazi CP2101 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku **Start COM**.

Pokušajte pritiskati tipke PD0, PD1, PF6 i PF7 u aplikaciji sa slike 10.4 te pratite što se događa. Pristigle poruke mogu se obraditi i na način koji je prikazan u programskom kodu 10.5.

Programski kod 10.5: Novi sadržaj datoteke `vjezba1012.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"

void inicijalizacija() {
    uart_init(19200); //inicijalizacija serijske komunikacije
    lcd_init(); // konfiguriranje LCD displeja
    interrupt_enable(); //omoguci globalni prekid
    // PB4, PB5, PB6, PB7 konfigurirani kao izlazni pinovi
    DDRB |= (1 << PB4) | (1 << PB5) | (1 << PB6) | (1 << PB7);
}

int main(void) {

    inicijalizacija();
    char B, x, y;

    while(1) {
        // provjera primljenih poruka
        if(uart_read_all() == true) {
            lcd_clrscr();
            lcd_home();
            lcd_print("%s", uart_buffer);
            B = uart_buffer[0];
            x = uart_buffer[1];
            y = uart_buffer[2];
        }
    }
}
```



```

        if (B == 'B' && (x >= '4' && x <= '7')) {
            if (y == '0' || y == '1') {
                set_port(PORTB, x - 48, y - 48);
            }
        }
    }
}
return 0;
}

```

U programskom kodu 10.5 ne koristimo objekte za digitalne izlaze, već ih konfiguriramo direktno pomoću registra **DDRB**. Automat stanja iz programskog koda 10.4 zamijenjen je dvama uvjetovano izvedenim blokovima. Prvi uvjetovani blok ispituje je li znak u varijabli **B** jednak znaku **'B'** i je li znak u varijabli **x** veći od znaka **'4'**, a manji od znaka **'7'**. S obzirom na to da su znakovi kodirani prema ASCII tablici, ovi znakovi mogu se uspoređivati operatorima usporedbe. Drugi uvjetovani blok ispituje je li znak u varijabli **y** jednak **'0'** ili jednak **'1'**. Kada su oba uvjeta zadovoljena, sigurni smo da je pristigla ispravna poruka.

U varijablama **x** i **y** nalaze znakovi s ASCII kodom brojeva 0 - 9. ASCII kod znaka **'0'** je 48 te je od varijable **x** potrebno oduzeti broj 48 kako bismo dobili poziciju pina kojem želimo promijeniti stanje. Isto vrijedi i za varijablu **y**. Ovo je standardna pretvorba znakova dekadskog sustava u brojeve dekadskog sustava. Prema navedenom, naredbom `set_port(PORTB, x - 48, y - 48)` mijenjamo stanje na pinu PBx.

Poruku oblika **"Bxy\r"** možete poslati pomoću tekstualnog okvira za slanje poruka. Na primjer, upišite u tekstualni okvir **Message** poruka **"B41"** te pritisnite tipku **Send**. Kada se pritisne tipka **Send**, poruci **"B41"** na kraj će se dodati specijalni znak **"\r"**. Pomoću tekstualnog okvira pošaljite proizvoljnu poruku. Ako poruka nije oblika **"Bxy\r"**, ona će se ispisati na LCD displeju, dok se stanja LED dioda neće mijenjati.

Promijenite u programskom kodu 10.5 brzinu prijenosa podataka s 19200 b/s na 57600 b/s. Prevedite datoteku `vjezba1012.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću aplikacije sa slike 10.4 bez mijenjanja postavki COM porta u aplikaciji.

Primijetiti ćete da različito podešene brzine na dvama uređajima koji komuniciraju serijskom komunikacijom rezultiraju pogrešno poslanim porukama. Promijenite brzinu prijenosa podataka s 19200 b/s na 57600 b/s u aplikaciji sa slike 10.4 te ponovno testirajte program.

Zatvorite datoteku `vjezba1012.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.3

Napravite program kojim će se pomoću tipkala PD0, PD1 i PF6 u aplikaciji sa slike 10.4 uključivati redom crvena LED dioda, žuta LED dioda i zelena LED dioda na razvojnom okruženju. Klikom miša na tipkalo u aplikaciji, putem serijske komunikacije, šalje se niz znakova u obliku **"Bxy\r"** (korisni dio poruke jest **"Bxy"**) po istom principu kao u prošloj vježbi. Plava LED dioda mora mijenjati intenzitet u ovisnosti o vrijednosti klizača **Slider** u aplikaciji sa slike 10.4 koristeći *Phase Correct* PWM način rada sklopa *Timer/Counter0*. Pomoću tipkala PF7 u aplikaciji sa slike 10.4 omogućuje se i onemogućuje generiranje PWM signala na pinu PB7 (pin na kojem je spojena plava LED dioda). Na primjer, ako je pristigla poruka **"B71\r"** (korisni dio poruke jest **"B71"**), na pinu PB7 generirat će se PWM signal i plava LED dioda mijenjat će intenzitet sukladno vrijednosti klizača, a ako je pristigla poruka **"B70\r"** (korisni dio poruke jest **"B70"**), plava LED dioda bit će isključena te će biti onemogućeno generiranje PWM signala na pinu PB7. Promjenom pozicije klizača u aplikaciji sa slike 10.4 serijskom komunikacijom šalje se poruka oblika **"S%u\*\r"**. Na primjer, ako je pozicija klizača 125, tada će poslana poruka biti **"S125\*\r"** (korisni dio poruke jest **"S125\*"**). Brzinu prijenosa podataka postavite na 38400 b/s.



```

        break;
    case '7':
        if (y == '1') LED_plava.enable();
        if (y == '0') LED_plava.disable();
        break;
    default:
        break;
    }
}

if (key_char == 'S') {
    for(uint8_t i = 1; uart_buffer[i] != '*'; i++) {
        slider = slider * 10 + (uart_buffer[i] - 48);
    }
    OCOA_set_duty_cycle(slider/2.55);
    slider = 0;
}
}
return 0;
}

```

U ovoj vježbi potrebno je napraviti program koji će imati funkcionalnost programa iz prethodne vježbe, izuzev upravljanja intenzitetom plave LED diode. U programskom kodu 10.6 u funkciji `inicijalizacija()` napravite konfiguraciju asinkrone serijske komunikacije s brzinom prijenosa od 38400 b/s. S obzirom na to da će se mijenjati intenzitet plave LED diode koja je spojena na pin PB7, koristit ćemo sklop *Timer/Counter0* u *Phase Correct* načinu rada. Ovaj način rada konfiguriran je u funkciji `inicijalizacija()`. U beskonačnoj `while` petlji ponovno se nalazi automat stanja koji čeka primljene poruke iz aplikacije sa slike 10.4. Razlika u automatu stanja koji se brine o uključanju LED dioda u programskom kodu 10.4 i programskom kodu 10.6 jest u slučaju kada je drugi pristigli znak u poruci oblika "`Bxy\r`" jednak `'7'`. U tom slučaju omogućujemo, odnosno onemogućujemo objekt `LED_plava` koji je povezan s izlaznim pinom PB7. Na taj način generiramo, odnosno ne generiramo PWM signal na pinu PB7. Omogućavanje, odnosno onemogućavanje PWM signala može se provesti i funkcijama `timer0_OCOA_enable_non_inverted_PWM()`, odnosno `timer0_OCOA_disable()`.

Vrijednost klizača iz aplikacije šalje se kao niz znakova oblika "`S%u*\r`". Da se pristigla poruka tiče promjene intenziteta plave LED diode pomoću klizača iz aplikacije sa slike 10.4, detektirat ćemo tako što će prvi znak u međuspremniku znakova `usart_buffer` biti jednak `'S'`. Nakon znaka `'S'` slijedi broj nepoznatog broja znamenaka (jedna, dvije ili tri) te znak `'*'` koji u korisnom dijelu poruke definira kraj poslanih vrijednosti s klizača.

Niz znakova koji predstavlja vrijednost klizača u aplikaciji kreće se u intervalu `[0, 255]`. Taj niz znakova potrebno je pretvoriti u cijeli broj koji će biti u intervalu `[0, 255]`. Pretvorba se provodi u `for` petlji izrazom `slider = slider * 10 + (uart_buffer[i] - 48)`. Pokažimo pretvorbu niza znakova u cijeli broj na poruci "`S125*`":

- `key_char`, odnosno `usart_buffer[0]` jednak je znaku `'S'`, stoga iza ovoga znaka očekujemo cijeli broj nepoznatog broja znamenki.
- Varijabla `slider` ima početnu vrijednost 0. U prvoj iteraciji `for` petlje na poziciji `usart_buffer[1]` nalazi se znak `'1'`. Od ovog znaka oduzima se 48 te će se dobiti broj 1. Vrijednost varijable `slider` sada je 1. Da je sljedeći znak u međuspremniku (`usart_buffer[2]`) jednak `'*'`, pretvorba bi završila.
- U drugoj iteraciji `for` petlje na poziciji `usart_buffer[2]` nalazi se znak `'2'`. Od ovog znaka oduzima se 48 te će se dobiti broj 2. Stara vrijednost varijable `slider` množi se s 10 i dodaje se 2 te je nova vrijednost varijable `slider` jednaka 12. Da je sljedeći znak u međuspremniku (`usart_buffer[3]`) jednak `'*'`, pretvorba bi završila.

- U trećoj iteraciji `for` petlje na poziciji `usart_buffer[3]` nalazi se znak `'5'`. Od ovog znaka oduzima se 48 te će se dobiti broj 5. Stara vrijednost varijable `slider` množi se s 10 i dodaje se 5 te je nova vrijednost varijable `slider` jednaka 125. S obzirom na to da je sljedeći znak u međuspremniku (`usart_buffer[4]`) jednak `'*`', pretvorba završava.
- Vrijednost varijable `slider` skalira se na interval `[0.0, 100.0]` te se skalirana vrijednost prosljeđuje u funkciju `OCOA_set_duty_cycle(slider/2.55)` koja mijenja širinu impulsa PWM signala, odnosno mijenja intenzitet plave LED diode.
- Nakon postavljanja nove širine impulsa, varijabla `slider` postavlja se na vrijednost 0 kako bi bila spremna za novi izračun vrijednosti klizača iz aplikacije sa slike 10.4.

Prevedite datoteku `vjezba1013.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću tipki i klizača u aplikaciji sa slike 10.4. Namjestite parametre serijske komunikacije za dostupan COM Port na kojem se nalazi CP2102 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`. Mijenjajte intenzitet plave LED diode pomoću klizača u aplikaciji sa slike 10.4.

Zatvorite datoteku `vjezba1013.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.4

Napravite program kojim ćete osigurati sljedeće funkcionalnosti:

- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'P'`, mikroupravljač aplikaciji mora poslati poruku `"ATmega32U4"`,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'C'`, mikroupravljač aplikaciji mora poslati informaciju o stanju crvene LED diode spojene na pin PB4,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'D'`, mikroupravljač aplikaciji mora poslati informaciju o stanju tipkala spojenog na pin PD0,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'A'`, mikroupravljač aplikaciji mora poslati rezultat analogno-digitalne pretvorbe na pinu ADC5,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'N'`, mikroupravljač aplikaciji mora poslati napon potencijometra (na tri decimalna mjesta) koji je spojen na pin ADC5,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'T'`, mikroupravljač aplikaciji mora poslati temperaturu koju mjeri NTC otpornik (na tri decimalna mjesta) koji je spojen na pin ADC4,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak `'V'`, mikroupravljač aplikaciji mora poslati vrijeme u sekundama od uključivanja mikroupravljača,
- za sve druge znakove poslane pomoću aplikacije sa slike 10.4 na mikroupravljač, mikroupravljač aplikaciji mora poslati `"Neispravan unos!"`.

Crvena LED dioda mora mijenjati stanje svakih 1000 ms pomoću sklopa `Timer/Counter1` u normalnom načinu rada. Brzinu prijenosa podataka postavite na 38400 b/s. Sve poruke aplikacije i mikroupravljača zaključane su specijalnim znakom `'\r'` (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke. Poruke poslane iz aplikacije potrebno je prikazivati



```

        // napravite AD pretvorbu na ADC5
        // pošaljite ADC5
        break;
    case 'N':
        // napravite AD pretvorbu na ADC5
        // izračunajte i pošaljite napon U_ADC5
        break;
    case 'T':
        // napravite AD pretvorbu na ADC4
        // izračunajte i pošaljite temperaturu
        break;
    case 'V':
        // pošaljite vrijeme rada mikroupravljača
        break;
    default:
        uart_print("Neispravan unos!\r");
        break;
    }
}
}
return 0;
}

```

Često je u praksi potrebno prezentirati neku varijablu sustava na zahtjev aplikacije. U ovoj vježbi zahtjevi su znakovi koje šaljemo na mikroupravljač koji povratno šalje informacije iz okoline sustava u kojoj se mikroupravljač nalazi.

U programskom kodu 10.7 potrebno je uključiti sva zaglavlja koja se koriste u vježbi ("LCD/lcd.h", "UART/uart.h", "Interrupt/interrupt.h", "DigitalIO/DigitalIO.h", "Timer/timer.h", "ADC/adc.h"). U funkciji `inicijalizacija()` konfigurirajte UART komunikaciju s brzinom prijenosa 19200 b/s, LCD displej, analogno-digitalnu pretvorbu, omogućite globalne prekide te konfigurirajte sklop *Timer/Counter1* u normalnom načinu rada pri kojem se prekidna rutina poziva svakih 1000 ms (jednu sekundu).

Prema funkcijama za inicijalizaciju sklopa *Timer/Counter1* koje su pozvane u funkciji `inicijalizacija()`, sklop *Timer/Counter1* konfiguriran je kao tajmer u normalnom načinu rada s djeljiteljem frekvencije radnog takta 1024 te je omogućen prekid sklopa *Timer/Counter1*. Vrijeme između dvaju poziva prekidne rutine mora biti 1000 ms ( $t_{T1} = 1$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 1 \cdot \frac{16000000}{1024} = 49911. \quad (10.3)$$

Početnu vrijednost iznosa 49911 potrebno je dodijeliti registru `TCNT1` u funkciji `inicijalizacija()` i u prekidnoj rutini sklopa *Timer/Counter1*. U `main()` funkciji prikazanoj u programskom kodu 10.7 nakon poziva funkcije `inicijalizacija()`, stvoren je objekt za tipkalo spojeno na pin PD0 i deklarirane su varijable koje će se koristiti u nastavku.

U beskonačnoj `while` petlji provjera primljenih poruka provodi se pozivom funkcije `uart_read_all()`. Ako je poruka pristigla, tada se ona najprije ispisuje na LCD displej. Nakon toga slijedi provjera je li na poziciji `uart_buffer[1]` znak `'\0'`, s obzirom na to da su svi upiti koji dolaze iz aplikacije sa slike 10.4 u obliku jednog slova. Ako je u međuspremnik `uart_buffer` stigao samo jedan znak, on se obrađuje automatom stanja (`switch case` blok). U ovisnosti o tome koji smo znak poslali na mikroupravljač, on mora nazad poslati poruku sukladno specifikacijama vježbe. Na primjer, ako na mikroupravljač stigne znak `'P'`, mikroupravljač će poslati poruku `"ATmega32U4\r"` pomoću funkcije `uart_print("ATmega32U4\r");`. Primijetite da je poruka zaključana znakom `'\r'`, što je bitno za svaku poslanu poruku jer aplikacija sa slike 10.4 očekuje poruku zaključanu znakom `'\r'`.

Zamijenite komentare u `switch case` bloku odgovarajućim pozivima funkcija, odnosno naredbama:

- za `case 'C'`: dodajte sljedeći poziv funkcije:
  - `uart_print("Stanje crvene diode je %d.\r", LED_crvena.state());`
- za `case 'D'`: dodajte sljedeći poziv funkcije:
  - `uart_print("Stanje tipkala D0 je %d.\r", Tipkalo_D0.state());`
- za `case 'A'`: dodajte sljedeće dvije naredbe:
  - `ADC_5 = adc_read(ADC5);`
  - `uart_print("ADC5 = %u.\r", ADC_5);`
- za `case 'N'`: dodajte sljedeće tri naredbe:
  - `ADC_5 = adc_read(ADC5);`
  - `U_ADC5 = ADC_5/1023.0*5.0;`
  - `uart_print("Napon na pinu ADC5 je %.3f V.\r", U_ADC5);`
- za `case 'T'`: dodajte sljedeće tri naredbe:
  - `ADC_4 = adc_read(ADC4);`
  - `T = 3435 / (log(ADC_4 / (1023.0 - ADC_4))+ 10.861)- 273.15;`
  - `uart_print("Temperatura je %.3f C.\r", T);`
- za `case 'V'`: dodajte sljedeći poziv funkcije:
  - `uart_print("Vrijeme rada uC je %lu s.\r", vrijeme);`

Ukoliko pošaljete krivi znak pomoću aplikacije na mikroupravljač, on će vratiti poruku "Neispravan unos!". Varijabla `vrijeme` uvećava se za jedan u prekidnoj rutini sklopa *Timer/Counter1* svaku sekundu. Na taj se način mjeri vrijeme rada mikroupravljača u sekundama. U prekidnoj se rutini isto tako mijenja stanje crvene LED diode.

Prevedite datoteku `vjezba1014.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port na kojem se nalazi CP2102 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku **Start COM**. Pomoću tekstualnog okvira **Message** unesite željene poruke te pošaljite poruku klikom miša na tipku **Send**. Poruke mikroupravljača možete provjeriti u dijalogu **Receive Dialog (Rx)**.

Zatvorite datoteku `vjezba1014.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.5

Potrebno je napraviti program kojim će se proširiti zahtjevi vježbe 10.1.2. sljedećim funkcionalnostima:

- jednom u sekundi mikroupravljač serijskom komunikacijom na aplikaciju sa slike 10.4 šalje poruku oblika `"A;%u;%u;%u*\r"`, gdje tri cijela broja predstavljaju rezultat analogno-digitalne pretvorbe na ADC4 pinu skaliran na cjelobrojni interval [0, 100], rezultat analogno-digitalne pretvorbe na ADC5 pinu skaliran na cjelobrojni interval [0, 100] i vrijeme rada mikroupravljača u sekundama kao modul broja 101.

- jednom u sekundi mikroupravljač serijskom komunikacijom na aplikaciju sa slike 10.4 šalje poruku oblika "P;%u;%u;%u;%u;%u;%u\*\r", gdje su prva tri cijela broja jednaka 1, 4 i 5, a druga tri cijela broja predstavljaju vrijeme rada mikroupravljača u sekundama, rezultat analogno-digitalne pretvorbe na ADC4 pinu i rezultat analogno-digitalne pretvorbe na ADC5 pinu.
- na padajući brid tipkala spojenog na pin PD0 mijenjati stanje crvene LED diode spojene na pinu PB4 te istovremeno mijenjati stanje crvene LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B40\*\r" i "B41\*\r".
- na padajući brid tipkala spojenog na pin PD1 mijenjati stanje žute LED diode spojene na pinu PB5 te istovremeno mijenjati stanje žute LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B50\*\r" i "B51\*\r".
- na padajući brid tipkala spojenog na pin PF6 mijenjati stanje zelene LED diode spojene na pinu PB6 te istovremeno mijenjati stanje zelene LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B60\*\r" i "B61\*\r".
- na padajući brid tipkala spojenog na pin PF7 mijenjati stanje plave LED diode spojene na pinu PB7 te istovremeno mijenjati stanje plave LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B70\*\r" i "B71\*\r".

Dakle, stanja fizičkih i virtualnih LED dioda mogu se mijenjati i pomoću fizičkih tipkala i pomoću virtualnih tipkala neometano. Brzinu prijenosa podataka postavite na 19200 b/s. Sve poruke aplikacije i mikroupravljača zaključane su specijalnim znakom '\r' (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke. Poruke poslane iz aplikacije potrebno je prikazivati na LCD displeju.

U projektnom stablu otvorite datoteku vjezba1015.cpp. Omogućite samo prevođenje datoteke vjezba1015.cpp. Početni sadržaj datoteke vjezba1015.cpp prikazan je programskim kodom 10.8.

Programski kod 10.8: Početni sadržaj datoteke vjezba1015.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "UART/uart.h"
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"
#include "Timer/timer.h"
#include "ADC/adc.h"

bool send_uart = false;
uint8_t send_uart_state = 0;
uint32_t vrijeme_16 = 0;

ISR(TIMER1_OVF_vect){
    TCNT1 = 49911;
    vrijeme_16++; // uvećaj svakih 1/16 s
    send_uart_state++; // korak [0,1,2,3,...]
    send_uart = true; // zahtjev za obradom svakih 1/16 s
}

void inicijalizacija() {
    uart_init(19200); //inicijalizacija serijske komunikacije
    lcd_init(); // konfiguriranje LCD displeja
```



```

    adc_init(); // konfiguriranje AD pretvorbe
    interrupt_enable(); //omogući globalni prekid
    // konfiguriranje tajmera 1
    timer1_set_normal_mode();
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_interrupt_OVF_enable();
    TCNT1 = 49911;
}

int main(void) {

    inicijalizacija();
    // stvaranje objekata za LED diode
    DigitalOutput LED_crvena(B4); // pin PB4 - crvena dioda
    DigitalOutput LED_zuta(B5); // pin PB5 - žuta dioda
    DigitalOutput LED_zelena(B6); // pin PB6 - zelena dioda
    DigitalOutput LED_plava(B7); // pin PB7 - plava dioda
    DigitalInput Tipkalo_D0(D0); // pin PD0 - tipkalo
    Tipkalo_D0.pullup_on(); // uključi pull up na PD0
    DigitalInput Tipkalo_D1(D1); // pin PD1 - tipkalo
    Tipkalo_D1.pullup_on(); // uključi pull up na PD1
    DigitalInput Tipkalo_F6(F6); // pin PF6 - tipkalo
    Tipkalo_F6.pullup_on(); // uključi pull up na PF6
    DigitalInput Tipkalo_F7(F7); // pin PF7 - tipkalo
    Tipkalo_F7.pullup_on(); // uključi pull up na PF6
    char B, x, y; // za provjeru primljenih poruka
    uint16_t ADC_4, ADC_5; // vrijednost AD pretvorbe na pinu ADC4 i ADC5
    uint16_t ADC_4_s, ADC_5_s; // skalirani ADC4 i ADC5 na [0, 100]
    uint16_t vrijeme = 0;
    uint16_t vrijeme_m = 0;

    while(1) {
        // provjera primljenih poruka
        if(uart_read_all() == true) {
            lcd_clrscr();
            lcd_home();
            lcd_print("%s", uart_buffer);
            B = uart_buffer[0];
            x = uart_buffer[1];
            y = uart_buffer[2];
            // provjera prvog znaka
            if (B == 'B') {
                // automat stanja koji obrađuje ostatak poruke tipa "xy"
                switch(x) {
                    case '4':
                        if (y == '1') LED_crvena.on();
                        if (y == '0') LED_crvena.off();
                        break;
                    case '5':
                        if (y == '1') LED_zuta.on();
                        if (y == '0') LED_zuta.off();
                        break;
                    case '6':
                        if (y == '1') LED_zelena.on();
                        if (y == '0') LED_zelena.off();
                        break;
                    case '7':
                        if (y == '1') LED_plava.on();
                        if (y == '0') LED_plava.off();
                        break;
                    default:
                        break;
                }
            }
        }
    }
}

```

```

    }
}

ADC_4 = adc_read(ADC4);
ADC_4_s = ADC_4 / 10.23; // ADC_4 skaliran na [0, 100]
ADC_5 = adc_read(ADC5);
ADC_5_s = ADC_5 / 10.23; // ADC_5 skaliran na [0, 100]
vrijeme = vrijeme_16 / 16; // vrijeme u [s]
vrijeme_m = vrijeme % 101; // vrijeme modul na [0,100]

// if koji se obrađuje svakih 1000/16 = 62.5 ms
if (send_uart) {

    // t1 = 62.5 ms
    if (send_uart_state == 1) {
        uart_print("A;%u;%u;%u*\r", ADC_4_s, ADC_5_s, vrijeme_m);
    }
    // t3 = 187.5 ms
    if (send_uart_state == 3) {
        uart_print("P;%u;%u;%u;%u;%u;%u*\r", 1, 4, 5, vrijeme,
            ADC_4, ADC_5);
    }
    // if koji se obrađuje na svaki paran poziv prekidne rutine
    // (svakih 125 ms)
    if ((send_uart_state % 2) == 0) {
        // na padajućem bridu tipkala PD0 (kada se tipkalo pritisne)
        if (Tipkalo_D0.isFalling_edge() == true) {
            if (LED_crvena.state()) {
                uart_print("B40*\r"); // pošalji niz za crvena off
                LED_crvena.off();
            }
            else {
                uart_print("B41*\r"); // pošalji niz za crvena on
                LED_crvena.on();
            }
        }
    }

    // nastaviti za ostala tipkala prema programskom kodu 10.9
}

// kada prođe 16 * 1000/16 = 1000 ms, brojač
//koraka se postavlja na 0
if (send_uart_state == 16) {
    send_uart_state = 0;
}
// obrađen je if koji se obrađuje svakih 1000/16 = 62.5 ms
send_uart = false;
}
}
return 0;
}
}

```

Dio programskog koda 10.8 koji obrađuje primljene poruke oblika "Bxy\r" isti je kao programski kod 10.4 iz vježbe 10.1.2. U ovoj su vježbi naspram vježbe 10.1.2. dodane brojne funkcionalnosti. U funkciji `inicijalizacija()` konfigurirani su serijska komunikacija, LCD displej, analogno-digitalna pretvorba i sklop *Timer/Counter1* u normalnom načinu rada. Također, globalno su omogućeni prekidi.

U `main()` funkciji stvoreni su objekti koji obrađuju četiri LED diode (objekti tipa `DigitalOutput`) i četiri tipkala (objekti tipa `DigitalInput`). Svim objektima koji obrađuju ulaze (tipkala) uključen je pritezni otpornik. Deklarirane su i brojne cjelobrojne varijable koje

ćemo opisati u nastavku.

U beskonačnoj `while` petlji prvi dio koda obrađuje primljene poruke oblika "`Bxy\r`", što smo objasnili u vježbi 10.1.2. S obzirom na zahtjeve definirane ovom vježbom, pripremljene su vrijednosti u sljedećim cjelobrojnim varijablama:

- `ADC_4` - varijabla u kojoj se nalazi rezultat analogno-digitalne pretvorbe na pinu ADC4. Ova varijabla poprima vrijednosti u intervalu  $[0, 1023]$ .
- `ADC_4_s` - varijabla koja skalira vrijednosti varijable `ADC_4` u cjelobrojni interval  $[0, 100]$  ( $[0, 1023] \div 10.23 \rightarrow [0, 100]$ ).
- `ADC_5` - varijabla u kojoj se nalazi rezultat analogno-digitalne pretvorbe na pinu ADC5. Ova varijabla poprima vrijednosti u intervalu  $[0, 1023]$ .
- `ADC_5_s` - varijabla koja skalira vrijednosti varijable `ADC_5` u cjelobrojni interval  $[0, 100]$  ( $[0, 1023] \div 10.23 \rightarrow [0, 100]$ ).
- `vrijeme` - varijabla u kojoj se nalazi vrijeme rada mikroupravljača u sekundama.
- `vrijeme_m` - varijabla u kojoj se nalazi modul varijable `vrijeme` sa 101, što varijablu `vrijeme`, koja može poprimiti vrijednosti veće od 100, smješta u cjelobrojni interval  $[0, 100]$ .

Poruke oblika "`A;%u;%u;%u*\r`" i "`P;%u;%u;%u;%u;%u;%u*\r`" na mikroupravljač se šalju jednom u sekundi. Svakako je važno da razmak između slanja dviju poruka istog oblika (npr. "`A;%u;%u;%u*\r`") uvijek bude jedna sekunda. Istovremeno je potrebno u stvarnom vremenu obrađivati padajuće bridove na tipkalima. Slanje poruka na mikroupravljač bit će sinkronizirano pomoću tajmera. Koristit ćemo konfiguraciju sklopa *Timer/Counte1* iz prethodne vježbe, ali uz 16 puta manji djelitelj frekvencije radnoga takta. Takva konfiguracija tajmera prekidnu će rutinu `ISR(TIMER1_OVF_vect)` pozvati 16 puta u jednoj sekundi. U prekidnoj rutini `ISR(TIMER1_OVF_vect)` trima varijablama dodjeljuju se vrijednosti:

- `vrijeme_16` - varijabla koja svoju vrijednost stalno uvećava za jedan pri svakom pozivu prekidne rutine `ISR(TIMER1_OVF_vect)`. Ova varijabla uveća se za 16 svaku sekundu, stoga se varijabla `vrijeme` (vrijeme u sekundama) dobije tako da se varijabla `vrijeme_16` podijeli sa 16.
- `send_uart_state` - varijabla koja prolazi kroz 16 stanja (0,1,2,...,14,15,(16) 0,1,...) u jednoj sekundi.
- `send_uart` - varijabla koja se postavlja u vrijednost `true` pri svakom pozivu prekidne rutine `ISR(TIMER1_OVF_vect)`, a služi za indikaciju da je obrađena prekidna rutina, odnosno da je promijenjeno stanje `send_uart_state`.

Varijable `send_uart` i `send_uart_state` koriste se u funkciji tzv. automata stanja u `while` petlji. Naime, kod svakog poziva prekidne rutine `ISR(TIMER1_OVF_vect)` varijabla `send_uart` postavlja se u vrijednost `true`. Slanje poruka s mikroupravljača uvjetovano je upravo varijablom `send_uart`, što se može vidjeti u `while` petlji. Kada je varijabla `send_uart` jednaka `true`, tada u ovisnosti o stanju `send_uart_state` mikroupravljač šalje određenu poruku. Kada se slanje poruka obradi, varijabla `send_uart` poprima vrijednost `false`. Slanje poruka obrađuje se svakih 62.5 ms (1000/16 ms).

Kada je stanje `send_uart_state` jednako 1, tada mikroupravljač šalje poruku oblika "`A;%u;%u;%u*\r`". Ova poruka sadrži rezultat analogno-digitalne pretvorbe na ADC4 pinu skaliran na cjelobrojni interval  $[0, 100]$ , rezultat analogno-digitalne pretvorbe na ADC5 pinu

skaliran na cjelobrojni interval [0, 100] i vrijeme rada mikroupravljača u sekundama koje je modul broja 101 (vidjeti programski kod 10.8). Navedeni podaci prikazivat će na trakama za prikaz napretka X, Y, Z na aplikaciji sa slike 10.4. Kada je stanje `send_uart_state` jednako 3, tada mikroupravljač šalje poruku oblika "`P;%u;%u;%u;%u;%u;%u*\r`". Ova poruka sadrži prva tri cijela broja s vrijednostima 1, 4 i 5 te druga tri cijela broja koji predstavljaju vrijeme rada mikroupravljača u sekundama, rezultat analogno-digitalne pretvorbe na ADC4 pinu i rezultat analogno-digitalne pretvorbe na ADC5 pinu (vidjeti programski kod 10.8). Navedeni podaci prikazivat će se u polju podataka `Values` na aplikaciji sa slike 10.4. Obje poruke neprestano se šalju svaku sekundu jer varijabla `send_uart_state` svaku sekundu neprestano prolazi kroz stanja 1 i 3.

Kada varijabla `send_uart_state` poprimi parne vrijednosti (stanje 2,4,6,...,16), tada se obrađuju padajući bridovi tipkala. Dakle, padajući bridovi tipkala obrađuju se svakih 125 ms. U programski kod 10.8 potrebno je dodati programski kod 10.9 kako bi se obradila tipkala spojena na pinove PD1, PF6 i PF7. Kada varijabla `send_uart_state` postigne vrijednost 16, tada se ona odmah postavlja u vrijednost 0, što omogućuje stalnu izmjenu 16 stanja.

Programski kod 10.9: Niz naredaba koje obrađuju pritisnuta tipkala na pinovima PD1, PF6 i PF7 te serijskom komunikacijom šalju poruke kojima se uključuju LED diode u aplikaciji sa slike 10.4

```
// na padajući brid tipkala PD1 (kada se tipkalo pritisne)
if (Tipkalo_D1.isFalling_edge() == true) {
    if (LED_zuta.state()) {
        uart_print("B50*\r"); // pošalji niz za zuta off
        LED_zuta.off();
    }
    else {
        uart_print("B51*\r"); // pošalji niz za zuta on
        LED_zuta.on();
    }
}
// na padajući brid tipkala PF6 (kada se tipkalo pritisne)
if (Tipkalo_F6.isFalling_edge() == true) {
    if (LED_zelena.state()) {
        uart_print("B60*\r"); // pošalji niz za zelena off
        LED_zelena.off();
    }
    else {
        uart_print("B61*\r"); // pošalji niz za zelena on
        LED_zelena.on();
    }
}
// na padajući brid tipkala PF7 (kada se tipkalo pritisne)
if (Tipkalo_F7.isFalling_edge() == true) {
    if (LED_plava.state()) {
        uart_print("B70*\r"); // pošalji niz za plava off
        LED_plava.off();
    }
    else {
        uart_print("B71*\r"); // pošalji niz za plava on
        LED_plava.on();
    }
}
```

Prevedite datoteku `vjezba1015.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port na kojem se nalazi CP2102 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`. Pritisnite tipkala, promijenite zakret potencijometra, zagrijete NTC otpornik

te pratite na aplikaciji sa slike 10.4 što se događa.

Teoretski, slanje podataka s mikroupravljača svaku sekundu mogli smo realizirati koristeći funkciju `_delay_ms()`. Zašto je korištenje funkcije `_delay_ms()` uvijek loše u programskim rješenjima mikroupravljača? Nikako nije dobro zaustaviti `while` petlju. Ako bismo ju zaustavili na jednu sekundu, tada poruke koje mikroupravljač primi unutar izvođenja funkcije kašnjenja ne bi bile obrađene iz razloga što se neće izvršiti dio koda u kojem provjeravamo je li stigla nova poruka (`if(uart_read_all()== true)`). Nadalje, ni tipkala se neće obrađivati ispravno. Uvijek će slanje informacije o padajućem bridu kasniti jednu sekundu. U inženjerskoj praksi jedini ispravan način jest formirati `while` petlju koja neće imati vremenska kašnjenja uzrokovana funkcijom `_delay_ms()`. U tom slučaju vremenski uvjetovane akcije morate provesti koristeći tajmere u normalnom načinu rada, kako je prikazano u ovoj vježbi.

Zatvorite datoteku `vjezba1015.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 10.1.1 Zadaci - univerzalna asinkrona serijska komunikacija

#### Zadatak 10.1.1

Napravite program kojim ćete slati i primiti proizvoljne tekstualne poruke UART komunikacijom mikroupravljača. U programu mikroupravljača definirajte dvije proizvoljne tekstualne poruke koje ćete slati pritiskom na tipkala spojena na pinove PF6 i PF7. Na računalu je za slanje i primanje tekstualnih poruka potrebno koristiti terminal *Tera Term*. Poslane poruke iz terminala potrebno je prikazivati na LCD displeju. Znakovni niz koji šaljemo na serijski port putem terminala zaključen je *Carriage Return* znakom (ASCII kod 0x0D) (vidi postavke na slici 10.7) kako bi se u mikroupravljaču detektirao kraj poruke koja pristiže serijskom komunikacijom.

---

#### Zadatak 10.1.2

Napravite program kojim će se pomoću tipkala PD0, PD1, PF6 i PF7 u aplikaciji na slici 10.4 uključivati redom crvena ('R') LED dioda, žuta ('Y') LED dioda, zelena ('G') LED dioda i plava ('B') LED dioda na razvojnom okruženju. Klikom miša na tipkalo u aplikaciji, putem serijske komunikacije, šalje se niz znakova u obliku "xy\r" (korisni dio poruke jest "xy"), gdje je:

- 'x' - znak koji određuje boju LED diode ('x' = 'R', 'Y', 'G', 'B'),
- 'y' - znak koji određuje digitalno stanje LED diode ('y' = '0', '1'),
- '\r' - specijalan znak (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke.

Na primjer, ako je pristigla poruka "R1\r" (korisni dio poruke jest "R1"), crvenu LED diodu treba uključiti, a ako je pristigla poruka "R0\r" (korisni dio poruke jest "R0"), crvenu LED diodu treba isključiti. Brzinu prijenosa podataka postavite na 38400 b/s. Poruke poslane iz aplikacije potrebno je prikazivati na LCD displeju.

---

#### Zadatak 10.1.3

Napravite program kojim će se pomoću tipkala PD0, PD1 i PF6 u aplikaciji na slici 10.4 uključivati redom crvena LED dioda, žuta LED dioda i zelena LED dioda na razvojnom okruženju. Klikom miša na tipkalo u aplikaciji, putem serijske komunikacije, šalje se niz znakova u obliku "xy\r" (korisni dio poruke je "xy") po istom principu kao u prošlom zadatku. Plava LED dioda mora mijenjati intenzitet u ovisnosti o vrijednosti klizača *Slider* u aplikaciji sa slike 10.4 koristeći *Fast PWM* način rada sklopa *Timer/Counter0*. Pomoću tipkala PF7 u aplikaciji na slici 10.4 omogućuje se i onemogućuje generiranje PWM signala na pinu PB7 (pin na kojem je spojena plava LED dioda). Na primjer, ako je pristigla poruka "B1\r" (korisni dio poruke jest "B1"), na pinu PB7 generirat će se PWM signal i plava LED dioda mijenjat će intenzitet sukladno vrijednosti klizača, a ako je pristigla poruka "B0\r" (korisni dio poruke jest "B0"), plava LED dioda bit će isključena te onemogućeno generiranje PWM signala na pinu PB7. Promjenom pozicije klizača u aplikaciji sa slike 10.4 serijskom komunikacijom šalje se poruka oblika "S%u\r". Na primjer, ako je pozicija klizača 125, tada će poslana poruka biti "S125\*\r" (korisni dio poruke jest "S125\*"). Brzinu prijenosa podataka postavite na 19200 b/s. Sve poruke zaključane su specijalnim znakom '\r' (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke. Poruke poslane iz aplikacije potrebno je prikazivati na LCD displeju.

### Zadatak 10.1.4

Napravite program kojim ćete osigurati sljedeće:

- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'F', mikroupravljač aplikaciji mora poslati poruku "AVR porodica mikroupravljača",
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'Z', mikroupravljač aplikaciji mora poslati informaciju o stanju zelene LED diode spojene na pin PB6,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'D', mikroupravljač aplikaciji mora poslati informaciju o stanju tipkala spojenog na pin PD1,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'A', mikroupravljač aplikaciji mora poslati rezultat analogno-digitalne pretvorbe na pinu ADC4,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'N', mikroupravljač aplikaciji mora poslati napon na tri decimalna mjesta potenciometra koji je spojen na pin ADC4,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'O', mikroupravljač aplikaciji mora poslati otpor potenciometra na tri decimalna mjesta koji je spojen na pin ADC5,
- ako pomoću aplikacije sa slike 10.4 na mikroupravljač pošaljemo znak 'M', mikroupravljač aplikaciji mora poslati vrijeme u minutama od uključenja mikroupravljača,
- za sve druge znakove poslane pomoću aplikacije sa slike 10.4 na mikroupravljač, mikroupravljač aplikaciji mora poslati "Neispravan unos!".

Crvena LED dioda mora mijenjati stanje svakih 500 ms pomoću sklopa *Timer/Counter3* u normalnom načinu rada. Brzinu prijenosa podataka postavite na 19200 b/s. Sve poruke aplikacije i mikroupravljača zaključane su specijalnim znakom '\r' (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke. Poruke poslane iz aplikacije potrebno je prikazivati na LCD displeju.

### Zadatak 10.1.5

Potrebno je napraviti program kojim će se proširiti zahtjevi zadatka 10.1.2 sljedećim funkcionalnostima:

- svakih 500 ms mikroupravljač serijskom komunikacijom na aplikaciju sa slike 10.4 šalje poruku oblika "A;%u;%u;%u\*\r", gdje tri cijela broja predstavljaju rezultat analogno-digitalne pretvorbe na ADC5 pinu skaliran na cjelobrojni interval [0, 100], rezultat analogno-digitalne pretvorbe na ADC4 pinu skaliran na cjelobrojni interval [0, 100] i srednju vrijednost skaliranih rezultata analogno-digitalne pretvorbe na pinovima ADC4 i ADC5.
- svakih 500 ms mikroupravljač serijskom komunikacijom na aplikaciju sa slike 10.4 šalje poruku oblika "P;%u;%u;%u;%u;%u;%u\*\r", gdje su prva tri cijela broja jednaka 0, 5 i 4, a druga tri cijela broja predstavljaju vrijeme rada mikroupravljača u minutama, rezultat analogno-digitalne pretvorbe na ADC5 pinu i rezultat analogno-digitalne pretvorbe na ADC4 pinu.

- na rastući brid tipkala spojenog na pin PD0 mijenjati stanje crvene LED diode spojene na pinu PB4 te istovremeno mijenjati stanje crvene LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B40\*\r" i "B41\*\r".
- na rastući brid tipkala spojenog na pin PD1 mijenjati stanje žute LED diode spojene na pinu PB5 te istovremeno mijenjati stanje žute LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B50\*\r" i "B51\*\r".
- na rastući brid tipkala spojenog na pin PF6 mijenjati stanje zelene LED diode spojene na pinu PB6 te istovremeno mijenjati stanje zelene LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B60\*\r" i "B61\*\r".
- na rastući brid tipkala spojenog na pin PF7 mijenjati stanje plave LED diode spojene na pinu PB7 te istovremeno mijenjati stanje plave LED diode u aplikaciji sa slike 10.4 slanjem poruka oblika "B70\*\r" i "B71\*\r".

Dakle, stanja fizičkih i virtualnih LED dioda mogu se mijenjati i pomoću fizičkih tipkala i pomoću virtualnih tipkala neometano. Brzinu prijenosa podataka postavite na 38400 b/s. Sve poruke aplikacije i mikroupravljača zaključane su specijalnim znakom '\r' (*Carriage Return*) koji definira kraj poruke i nije dio korisne poruke. Poruke poslane iz aplikacije potrebno je prikazivati na LCD displeju.

---



# Poglavlje 11

## Vanjski prekidi

Vanjski prekidi (engl. *External Interrupts*) jesu prekidi koji su izazvani vanjskim događajima. Vanjske događaje najčešće generiraju senzori na svome izlazu u obliku rastućih (engl. *rising*) i padajućih (engl. *falling*) bridova signala. Primjer takvih senzora jesu enkoderi, kapacitivni senzori, induktivni senzori, krajnji prekidači, tipkala i drugi. Izlazi senzora spajaju se na pinove mikroupravljača koji omogućuju detekciju rastućih i padajućih bridova signala. Ovi bridovi signala izazivaju prekide u mikroupravljaču koji pozivaju prekidnu rutinu onog trenutka kada se brid signala pojavi. U prekidnoj rutini moguće je na odgovarajući način odgovoriti na vanjski prekid.

### 11.1 Vježbe - vanjski prekidi

Dvije su vrste prekida koji se mogu generirati na pinovima mikroupravljača ATmega32U4: vanjski prekidi (engl. *External Interrupts* - INT) i prekidi izazvani promjenom stanja na pinu (engl. *Pin Change Interrupts* - PCINT). Vanjski prekidi mikroupravljača ATmega32U4 mogu se generirati na pinovima:

- INT0 (PD0),
- INT1 (PD1),
- INT2 (PD2),
- INT3 (PD3),
- INT6 (PE6).

Prekidi se na navedenim pinovima mogu generirati čak i kada su pinovi konfigurirani kao izlazni pinovi, što omogućuje softverski prekid. Na pinovima INT0, INT1, INT2, INT3 i INT6 zahtjev za prekid može generirati padajući i/ili rastući brid signala te niska razina signala, što je moguće konfigurirati za svaki vanjski prekid posebno. Vanjski prekid INT0, INT1, INT2 i INT3 konfigurira se u registru **EICRA** pomoću bitova **ISCx1** i **ISCx0** prema tablici 11.1. Prekid na pinu INT6 konfigurira se u registru **EICRB** pomoću bitova **ISC61** i **ISC60** prema tablici 11.2.

Vanjski prekidi omogućuju se u registru **EIMSK** na sljedeći način:

- vanjski prekid na pinu INT0 (PD0) bit će omogućen ako u registar **EIMSK** na mjesto bita **INT0** upišete vrijednost 1,

- vanjski prekid na pinu INT1 (PD1) bit će omogućen ako u registar **EIMSK** na mjesto bita **INT1** upišete vrijednost 1,
- vanjski prekid na pinu INT2 (PD2) bit će omogućen ako u registar **EIMSK** na mjesto bita **INT2** upišete vrijednost 1,
- vanjski prekid na pinu INT3 (PD3) bit će omogućen ako u registar **EIMSK** na mjesto bita **INT3** upišete vrijednost 1,
- vanjski prekid na pinu INT6 (PE6) bit će omogućen ako u registar **EIMSK** na mjesto bita **INT6** upišete vrijednost 1,

Tablica 11.1: Konfiguracija za vanjski prekid na pinu INTx (PDx) (x = 0, 1, 2, 3)

ISCx1	ISCx0	Način rada za prekid INTx
0	0	Niska razina signala na pinu INTx (PDx) generira zahtjev za prekid
0	1	Bilo koja logička promjena signala na pinu INTx (PDx) generira asinkroni zahtjev za prekid
1	0	Padajući brid signala na pinu INTx (PDx) generira asinkroni zahtjev za prekid
1	1	Rastući brid signala na pinu INTx (PDx) generira asinkroni zahtjev za prekid

Tablica 11.2: Konfiguracija za vanjski prekid na pinu INT6 (PE6)

ISC61	ISC60	Način rada za prekid INT6
0	0	Niska razina signala na pinu INT6 (PE6) generira zahtjev za prekid
0	1	Bilo koja logička promjena signala na pinu INT6 (PE6) generira zahtjev za prekid
1	0	Padajući brid između dvaju uzoraka signala na pinu INT6 (PE6) generira zahtjev za prekid
1	1	Rastući brid između dvaju uzoraka signala na pinu INT6 (PE6) generira zahtjev za prekid

Primjer omogućenja i konfiguracije vanjskih prekida INT0, INT1 i INT2 prikazan je programskim kodom 11.1. Vanjski prekid INT0 konfiguriran je tako da oba brida signala na pinu INT0 generiraju prekid. INT1 konfiguriran je tako da rastući brid signala na pinu INT1 generira prekid, dok je INT2 konfiguriran tako da padajući brid signala na pinu INT2 generira prekid. Načini na koji se generiraju prekidi definirani su u registru **EICRA** prema tablici 11.1.

Programski kod 11.1: Omogućenje i konfiguracija vanjskih prekida INT0, INT1 i INT2

```
// prvi način - INT0
EIMSK |= (1 << INT0); // omogući prekid INT0
EICRA &= ~( (1 << ISC01) | (1 << ISC00) );
EICRA |= (0 << ISC01) | (1 << ISC00); // oba brida generiraju prekid INT0
// drugi način - INT1
set_bit_reg(EIMSK, INT1); // INT1 = 1 // omogući prekid INT1
// rastući brid generira prekid INT1
set_bit_reg(EICRA, ISC10); // ISC10 = 1
set_bit_reg(EICRA, ISC11); // ISC11 = 1
// treći način - INT2
int2_enable(); // omogući prekid INT2
int2_set_falling_edge(); // padajući brid generira prekid INT2
```

Ukoliko se vanjski prekid ne omogući, zahtjev za prekid neće se generirati. Ako su vanjski prekidi omogućeni, tada će oni pozivati sljedeće prekidne rutine:

- `ISR(INT0_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT0 (PD0),
- `ISR(INT1_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT1 (PD1),
- `ISR(INT2_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT2 (PD2),
- `ISR(INT3_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT3 (PD3),
- `ISR(INT6_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT6 (PE6).

Kada se koriste vanjski prekidi na pinovima INT0 (PD0), INT1 (PD1), INT2 (PD2), INT3 (PD3) i INT6 (PE6), tada pinove PD0, PD1, PD2, PD3 i PE6 postavite kao ulazne pinove<sup>1</sup>. Ako padajuće i rastuće bridove generiraju tipkala, krajnji prekidači i senzori s otvorenim kolektorom, tada je potrebno uključiti pritezne otpornike na pinovima PD0, PD1, PD2, PD3 i PE6. U slučaju senzora s *push-pull* izlazom, pritezne otpornike nije potrebno uključiti. Primijetite da se, ukoliko koristite serijsku komunikaciju na mikroupravljaču ATmega32U4, INT2 (PD2) i INT3 (PD3) ne mogu koristiti kao vanjski prekidi.

Autor je pripremio brojne funkcije koje omogućuju jednostavnu konfiguraciju vanjskih prekida pomoću zaglavlja "`interrupt.h`" koje se nalazi u mapi "`Interrupt`". U nastavku je popis funkcija koje se koriste za konfiguraciju vanjskih prekida:

- `int0_enable()` - funkcija koja omogućuje vanjski prekid na pinu INT0,
- `int1_enable()` - funkcija koja omogućuje vanjski prekid na pinu INT1,
- `int2_enable()` - funkcija koja omogućuje vanjski prekid na pinu INT2,
- `int3_enable()` - funkcija koja omogućuje vanjski prekid na pinu INT3,
- `int6_enable()` - funkcija koja omogućuje vanjski prekid na pinu INT6,
- `int0_disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT0,
- `int1_disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT1,
- `int2_disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT2,
- `int3_disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT3,
- `int6_disable()` - funkcija koja onemogućuje vanjski prekid na pinu INT6,
- `int0_set_low_level()` - funkcija koja vanjski prekid INT0 konfigurira na način da niska razina signala na pinu INT0 izaziva prekid,
- `int1_set_low_level()` - funkcija koja vanjski prekid INT1 konfigurira na način da niska razina signala na pinu INT1 izaziva prekid,

---

<sup>1</sup>Ovo je preporuka, iako će se vanjski prekidi generirati i ako su pinovi PD2, PD3 i PB2 postavljeni kao izlazni pinovi.

- `int2_set_low_level()` - funkcija koja vanjski prekid INT2 konfigurira na način da niska razina signala na pinu INT2 izaziva prekid,
- `int3_set_low_level()` - funkcija koja vanjski prekid INT3 konfigurira na način da niska razina signala na pinu INT3 izaziva prekid,
- `int6_set_low_level()` - funkcija koja vanjski prekid INT6 konfigurira na način da niska razina signala na pinu INT6 izaziva prekid,
- `int0_set_rising_falling_edge()` - funkcija koja vanjski prekid INT0 konfigurira na način da oba brida signala na pinu INT0 (i rastući i padajući) izazivaju prekid,
- `int1_set_rising_falling_edge()` - funkcija koja vanjski prekid INT1 konfigurira na način da oba brida signala na pinu INT1 (i rastući i padajući) izazivaju prekid,
- `int2_set_rising_falling_edge()` - funkcija koja vanjski prekid INT2 konfigurira na način da oba brida signala na pinu INT2 (i rastući i padajući) izazivaju prekid,
- `int3_set_rising_falling_edge()` - funkcija koja vanjski prekid INT3 konfigurira na način da oba brida signala na pinu INT3 (i rastući i padajući) izazivaju prekid,
- `int6_set_rising_falling_edge()` - funkcija koja vanjski prekid INT6 konfigurira na način da oba brida signala na pinu INT6 (i rastući i padajući) izazivaju prekid,
- `int0_set_falling_edge()` - funkcija koja vanjski prekid INT0 konfigurira na način da padajući brid signala na pinu INT0 izaziva prekid,
- `int1_set_falling_edge()` - funkcija koja vanjski prekid INT1 konfigurira na način da padajući brid signala na pinu INT1 izaziva prekid,
- `int2_set_falling_edge()` - funkcija koja vanjski prekid INT2 konfigurira na način da padajući brid signala na pinu INT2 izaziva prekid,
- `int3_set_falling_edge()` - funkcija koja vanjski prekid INT3 konfigurira na način da padajući brid signala na pinu INT3 izaziva prekid,
- `int6_set_falling_edge()` - funkcija koja vanjski prekid INT6 konfigurira na način da padajući brid signala na pinu INT6 izaziva prekid,
- `int0_set_rising_edge()` - funkcija koja vanjski prekid INT0 konfigurira na način da rastući brid signala na pinu INT0 izaziva prekid,
- `int1_set_rising_edge()` - funkcija koja vanjski prekid INT1 konfigurira na način da rastući brid signala na pinu INT1 izaziva prekid,
- `int2_set_rising_edge()` - funkcija koja vanjski prekid INT2 konfigurira na način da rastući brid signala na pinu INT2 izaziva prekid,
- `int3_set_rising_edge()` - funkcija koja vanjski prekid INT3 konfigurira na način da rastući brid signala na pinu INT3 izaziva prekid,
- `int6_set_rising_edge()` - funkcija koja vanjski prekid INT6 konfigurira na način da rastući brid signala na pinu INT6 izaziva prekid.

Prekidi izazvani promjenom stanja na pinu (PCINT prekidi) mikroupravljača ATmega32U4 mogu se generirati na pinovima:

- PCINT0 (PB0),

- PCINT1 (PB1),
- PCINT2 (PB2),
- PCINT3 (PB3),
- PCINT4 (PB4),
- PCINT5 (PB5),
- PCINT6 (PB6),
- PCINT7 (PB7).

Na navedenim pinovima prekid se može generirati čak i kada su pinovi konfigurirani kao izlazni pinovi, što omogućuje softverski prekid. Zahtjev za PCINT prekid može generirati bilo koja promjena signala (rastući i padajući brid) na pinovima PCINT0 - PCINT7. Koji god od navedenih osam pinova izazove prekid, uvijek se poziva smo jedna prekidna rutina, i to `ISR(PCINT0_vect)`. PCINT prekidi omogućuju se u registru `PCICR` tako da na mjesto bita `PCIE0` upišete vrijednost 1. Koji će od pinova PCINT0 - PCINT7 moći generirati zahtjev za prekid određuje se registrom `PCMSK0` sa slike 11.1. Na primjer, ako želimo da nam PCINT2 (PB2) može generirati prekid, tada ćemo u registar `PCMSK0` na mjesto bita `PCINT2` upisati vrijednost 1.

Bit	7	6	5	4	3	2	1	0
PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0

Slika 11.1: Registar `PCMSK0`

Autor je pripremio brojne funkcije koje omogućuju jednostavnu konfiguraciju PCINT prekida pomoću zaglavlja `"interrupt.h"` koje se nalazi u mapi `"Interrupt"`. U nastavku je popis funkcija koje se koriste za konfiguraciju PCINT prekida:

- `pcint_enable()` - funkcija koja omogućuje PCINT prekid,
- `pcint_disable()` - funkcija koja onemogućuje PCINT prekid,
- `pcint_pin_enable(uint8_t pin)` - funkcija koja omogućuje prekid na pinu koji je definiran argumentom `pin` (argument `pin` poprima predefinirane vrijednosti `PCINT0`, `PCINT1`, `PCINT2`, `PCINT3`, `PCINT4`, `PCINT5`, `PCINT6`, `PCINT7` prikazane na slici 11.1),
- `pcint_pin_disable(uint8_t pin)` - funkcija koja onemogućuje prekid na pinu koji je definiran argumentom `pin` (argument `pin` poprima predefinirane vrijednosti `PCINT0`, `PCINT1`, `PCINT2`, `PCINT3`, `PCINT4`, `PCINT5`, `PCINT6`, `PCINT7` prikazane na slici 11.1),
- `pcint_pin_enable_all()` - funkcija koja omogućuje prekid na svim pinovima PCINT0 - PCINT7,
- `pcint_pin_disable_all()` - funkcija koja onemogućuje prekid na svim pinovima PCINT0 - PCINT7.

U ovoj će se vježbi za generiranje prekida koristiti tipkala spojena na pinove PD0 i PD1 prema slici 4.5 te pinovi PB4 i PB5 na kojima su spojene LED diode prema slici 4.1 kako bismo generirali softverski prekid.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) 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 Ivica Ivić, datoteka koju ćete stvoriti zvat će se `Ivica Ivic`. Datoteku `Vanjski prekidi.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atstln` u datoteci `\\Vanjski prekidi\vjezbe`. U otvorenom projektu nalaze se sve vježbe koja 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 11.1.1

Napravite program kojim ćete:

- na pinu INT0 (PD0) brojati rastuće bridove signala tipkala spojenog na pin PD0,
- na pinu INT1 (PD1) brojati padajuće i rastuće bridove signala tipkala spojenog na pin PD1,
- na svaki rastući brid signala na pinu INT1 promijeniti stanje crvene LED diode spojene na pin PB4.

Vrijednosti brojača bridova signala na pinovima INT0 i INT1 prikažite na LCD displeju.

U projektnom stablu otvorite datoteku `vjezba1111.cpp`. Omogućite samo prevođenje datoteke `vjezba1111.cpp`. Početni sadržaj datoteke `vjezba1111.cpp` prikazan je programskim kodom 11.2.

Programski kod 11.2: Početni sadržaj datoteke `vjezba1111.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "Interrupt/interrupt.h"

int brojac_int0 = 0;
int brojac_int1 = 0;
bool lcd_update = true;

// prekidna rutina za vanjski prekid INT0
ISR(INT0_vect) {
    brojac_int0++;
    lcd_update = true;
}

// prekidna rutina za vanjski prekid INT1
ISR(INT1_vect) {
    brojac_int1++;
    lcd_update = true;
    if(get_pin(PIND, PD1)) {
        toggle_port(PORTB, PB4);
    }
}

void inicijalizacija() {
```

```

    lcd_init(); // konfiguriranje LCD displeja
    // konfiguracija prekida INTO
    EIMSK |= (1 << INTO); // omogući prekid INTO
    // padajući brid generira prekid na INTO
    EICRA &= ~((1 << ISCO1) | (1 << ISCO0));
    EICRA |= (1 << ISCO1) | (0 << ISCO0);
    // konfiguracija prekida INT1

    interrupt_enable(); //omogući globalni prekid
    // konfiguriranje ulaza PD0 i PD1 te izlaza PB4
    DDRD &= ~((1 << PD0) | (1 << PD1));
    PORTD |= (1 << PD0) | (1 << PD1);
    DDRB |= (1 << PB4);
}

int main(void) {

    inicijalizacija();

    while(1) {

        if(lcd_update) {
            lcd_clrscr();
            lcd_home();
            lcd_print("INT0: %d\nINT1: %d", brojac_int0, brojac_int1);
            lcd_update = false;
        }

        asm("nop"); // instrukcija koja kasni jedan ciklus CPU-a
    }
    return 0;
}

```

Za generiranje vanjskih prekida INTO i INT1 koristit ćemo tipkala spojena na pinove PD0 i PD1. U programskom kodu 11.2 u funkciji `inicijalizacija()` vanjski prekid INTO omogućen je u registru `EIMSK` tako da smo na mjesto bita `INT0` u registru `EIMSK` upisali vrijednost 1. Vanjski prekid INTO konfiguriran je tako da se zahtjev za prekid generira na padajući brid signala. Za ovu konfiguraciju, prema tablici 11.1 ( $x = 0$ ), bit `ISCO0` u registru `EICRA` mora biti jednak 0, a bit `ISCO1` u registru `EICRA` mora biti jednak 1.

Vanjskih prekid INT1 potrebno je konfigurirati tako da se zahtjev za prekid generira na oba brida signala (rastući i padajući). U programski kod 11.2 u funkciju `inicijalizacija()` potrebno je dodati sljedeći niz naredaba (ispod konfiguracije vanjskog prekida INTO):

- `EIMSK |= (1 << INT1);` - omogućenje vanjskog prekida INT1,
- `EICRA &= ~((1 << ISC11) | (1 << ISC10));` - čišćenje stanja bitova `ISC11` i `ISC10` (što nije nužno napraviti ako se radi o prvoj konfiguraciji u programskom kodu),
- `EICRA |= (0 << ISC11) | (1 << ISC10);` - zahtjev za prekid generiraju i rastući i padajući brid pa, prema tablici 11.1 ( $x = 1$ ), bit `ISC10` u registru `EICRA` mora biti jednak 1, a bit `ISC11` u registru `EICRA` mora biti jednak 0.

U programskom kodu 11.2 u funkciji `inicijalizacija()` dodatno je inicijaliziran LCD displej, globalno su omogućeni prekidi te su PD0 i PD1 konfigurirani kao ulazni, a PB4 kao izlazni pin.

Za brojanje bridova signala deklarirane su dvije globalne varijable `brojac_int0` i `brojac_int1`. Vrijednosti ovih varijabli uvećavaju se u prekidnim rutinama na sljedeći način:

- u prekidnoj rutini `ISR(INT0_vect)` vrijednost varijable `brojac_int0` uvećava se za jedan na svaki padajući brid signala na pinu INT0 (PD0) (svaki puta kad se pritisne tipkalo),
- u prekidnoj rutini `ISR(INT1_vect)` vrijednost varijable `brojac_int1` uvećava se za jedan na svaki padajući i na svaki rastući brid signala na pinu INT1 (PD1) (svaki puta kada se pritisne i kada se otpusti tipkalo).

U `while` petlji vrijednosti brojača bridova signala prikazuju se na LCD displeju, i to samo onda kada se dogodila promjena vrijednosti u varijablama `brojac_int0` i `brojac_int1`. Dakle, ispis na LCD displej uvjetovan je logičkom varijablom `lcd_update` koja se pri svakom pozivu prekidnih rutina `ISR(INT0_vect)` i `ISR(INT1_vect)` postavlja u vrijednost `true`. Nakon ispisa novih vrijednosti na LCD, varijabla `lcd_update` postavlja se u vrijednost `false`.

U `while` petlji pojavljuje se i jedna nova naredba: `asm("nop");`. Radi se o instrukciji NOP (engl. *no operation*) koja u pravilu zakasni program mikroupravljača za jedan impuls radnog takta (kašnjenje iznosi  $1/F_{CPU}$  s). Zašto se ona ovdje nalazi? Naime, `gcc` prevoditelj radi optimizaciju programskog koda. Procjena optimizacije jest da će se programski kod uvjetovan varijablom `lcd_update` izvesti samo jednom u petlji. Iz tog će razloga prevoditelj obrisati `while` petlju. 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. Umjesto naredbe `asm("nop")`, mogli smo koristiti i funkciju `_delay_us(1)` koja bi opet osigurala da prevoditelj ne obriše `while` petlju. Optimizaciju koju radi prevoditelj moguće je isključiti, u tom se slučaju programski kod 11.2 pri prevođenju u strojni kod poveća točno 2,5 puta. U slučaju da vam se pri testiranju programa učini da je funkcija `main()` završila s izvođenjem, a postoji mogućnost da prevoditelj optimizacijom obriše `while` petlju, na kraj `while` petlje dodajte naredbu `asm("nop")` ili funkciju `_delay_us(1)`. Autor prednost daje korištenju naredbe `asm("nop")` jer će u našem slučaju ona 16 puta kraće zaustaviti `while` petlju.

Prevedite datoteku `vjezba1111.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Pritisnite tipkala spojena na pinove PD0 i PD1 te promatrajte vrijednosti brojača bridova signala na LCD displeju.

Obrišite naredbu `asm("nop")` te ponovno prevedite datoteku `vjezba1111.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Primijetite da se pritiskom tipkala PD0 i PD1 stanje brojača bridova signala ne mijenja. Ostavite i dalje obrisanu naredbu `asm("nop")`.

Drugi, bolji, način kako spriječiti prevoditelj da optimizira kod `while` petlje jest korištenje ključne riječi `volatile` ispred tipa deklarirane varijable. Ključna riječ `volatile` ima za cilj spriječiti primjenu optimizacije programskog koda nad objektima koji se mogu promijeniti na načine koje prevoditelj ne može utvrditi. Takav je slučaj globalnih varijabli koje se mijenjaju u prekidnim rutinama. Zamijenite deklaraciju globalne varijable `bool lcd_update = true` deklaracijom `volatile bool lcd_update = true` te ponovno prevedite datoteku `vjezba1111.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Primijetite da se pritiskom tipkala PD0 i PD1 stanje brojača bridova signala sada mijenja. Preporuka je da se svakoj globalnoj varijabli koju mijenja prekidna rutina prije tipa deklaracije doda ključna riječ `volatile`.

Konfiguraciju vanjskih prekida moguće je napraviti pomoću funkcija napisanih u zaglavlju `"interrupt.h"` koje se nalazi u mapi `"Interrupt"`. Funkcije smo opisali u uvodnom djelu ovog poglavlja. Zamijenite konfiguraciju vanjskih prekida INT0 i INT1 u datoteci `vjezba1111.cpp` programskim kodom 11.3.



Programski kod 11.3: Nova konfiguracija vanjskih prekida INT0 i INT1

```
// konfiguracija prekida INT0
int0_enable(); // omogući prekid INT0
int0_set_falling_edge(); // oba brida generiraju prekid INT1
// konfiguracija prekida INT1
int1_enable(); // omogući prekid INT1
int1_set_rising_falling_edge(); // oba brida generiraju prekid INT1
```

Prevedite datoteku `vjezba1111.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Pritisnite tipkala spojena na pinove PD0 i PD1 te promatrajte vrijednosti brojača bridova signala na LCD displeju.

Zatvorite datoteku `vjezba1111.cpp` i onemogućite prevođenje ove datoteke.



### Vježba 11.1.2

Napravite program kojim ćete na proizvoljan način uključivati i isključivati crvenu LED diodu spojenu na pin PB4 i zelenu LED diodu spojenu na pin PB6. Stanje crvene LED diode potrebno je preslikati na žutu LED diodu spojenu na pin PB5, a stanje zelene LED diode potrebno je preslikati na plavu LED diodu spojenu na pin PB7 koristeći PCINT prekid. Dakle, PCINT prekid generirat će se softverski, promjenom stanja na pinovima PB4 i PB6.

U projektnom stablu otvorite datoteku `vjezba1112.cpp`. Omogućite samo prevođenje datoteke `vjezba1112.cpp`. Početni sadržaj datoteke `vjezba1112.cpp` prikazan je programskim kodom 11.4.

Programski kod 11.4: Početni sadržaj datoteke `vjezba1112.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"

DigitalOutput LED_crvena(B4);
DigitalOutput LED_zuta(B5);
DigitalOutput LED_zelena(B6);
DigitalOutput LED_plava(B7);

uint8_t pinb_old;

ISR(PCINT0_vect) {

    uint8_t pin_changed = pinb_old ^ PINB;

    if (pin_changed & (1 << PB4)) {
        LED_zuta.toggle();
    }

    if (pin_changed & (1 << PB6)) {
        if ((pinb_old & (1 << PB6))) {
            LED_plava.off();
        } else {
            LED_plava.on();
        }
    }

    pinb_old = PINB;
}
```

```

}

void inicijalizacija() {

    //omogući PCINT prekide
    pcint_enable();
    //omogući prekide na pinovima PCINT4 (PB4) i PCINT6 (PB6)
    pcint_pin_enable(PCINT4);
    pcint_pin_enable(PCINT6);

    pinb_old = PINB;
    interrupt_enable(); //omogući globalni prekid
}

int main(void) {

    inicijalizacija();
    uint8_t brojac = 0;

    while(1) {

        brojac++;

        if ((brojac % 5) == 0) LED_crvena.toggle();
        if ((brojac % 9) == 0) LED_zelena.toggle();

        _delay_ms(100);
    }
    return 0;
}

```

PCINT prekidi u osnovi su namijenjeni za obradu rastućih i padajućih bridova signala senzora spojenih na PCINT pinove koji se konfiguriraju kao ulazi. Ako PCINT pinove koristimo kao izlaze i mijenjamo im stanja, tada je moguće generirati softverski PCINT prekid. U programskom kodu 11.4 stvoreni su globalni objekti tipa `DigitalOutput` koji su povezani s crvenom, žutom, zelenom i plavom LED diodom koje su redom spojene na pinove PB4, PB5, PB6 i PB7.

U `while` petlji se stanje crvene LED diode mijenja svakih 500 ms, a stanje zelene LED diode svakih 900 ms. Stanje crvene LED diode potrebno je preslikati na žutu LED diodu, a stanje zelene LED diode potrebno je preslikati na plavu LED diodu. Preslikavanje se mora dogoditi odmah po promjeni stanja crvene i zelene LED diode. PCINT prekidi generiraju se pomoću pinova PB0 - PB7 pozivajući pritom prekidnu rutinu `ISR(PCINT0_vect)`. Da bi zadovoljili zahtjeve vježbe, pinovi PB4 i PB6 na kojima su spojene crvena i zelena LED dioda moraju generirati PCINT prekide. U programskom kodu 11.4 u funkciji `inicijalizacija()` to je postignuto pozivom sljedećih funkcija:

- `pcint_enable()` - funkcija koja omogućuje PCINT prekid,
- `pcint_pin_enable(PCINT4)` - funkcija koja omogućuje da PCINT prekid generira pin PCINT4 (PB4),
- `pcint_pin_enable(PCINT6)` - funkcija koja omogućuje da PCINT prekid generira pin PCINT6 (PB6).

Navedene funkcije nalaze se u zaglavlju `"interrupt.h"` koje se nalazi u mapi `"Interrupt"`. Važno je znati da preostalih šest pinova na portu B neće generirati prekide. U programskom kodu 11.4 u funkciji `inicijalizacija()` dodatno su globalno omogućeni prekidi te je u globalnu varijablu `pinb_old` dodijeljena vrijednost registra `PINB`. Varijabla `pinb_old` koristi se kako bi se u prekidnoj rutini detektiralo koji je pin na portu B izazvao prekid. Rastući i padajući brid

signala na pinovima PB4 i PB6, prema konfiguraciji u programskom kodu 11.4, pozivaju prekidnu rutinu `ISR(PCINT0_vect)`. U prekidnoj rutini najprije se provjerava koji je pin na portu B izazvao prekid pomoću naredbe `pin_changed = pinb_old ^ PINB;`. Ova naredba koristi bitovni operator *ex-ili*. Dvoulazni logički sklop *ex-ili* kao rezultat daje logičku jedinicu samo kada su mu ulazi različiti (jedan ulaz 0, drugi ulaz 1). Primjer detekcije pina PB4 koji je izazvao prekid prikazan je na slici 11.2.

U varijabli `pin_changed` vrijednost 1 imat će samo ona pozicija bita gdje se dogodila promjena u registru `PINB` u odnosu na prethodno stanje registra `PINB` koje se nalazi u varijabli `pinb_old`. Provjerom varijable `pin_changed` sad je lako ustanoviti koji je pin izazvao prekid. Je li prekid izazvala promjena signala na pinu PB4 provjerit ćemo pomoću uvjeta `if (pin_changed & (1 << PB4))`. Ovaj uvjet bit će zadovoljen samo ako bit na poziciji 4 u varijabli `pin_changed` ima vrijednost 1.

$$\begin{array}{r}
 \text{pinb\_old} \\
 \wedge \\
 \text{PINB} \\
 = \\
 \text{pin\_changed}
 \end{array}
 \begin{array}{|c|c|c|c|c|c|c|c|}
 \hline
 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
 \hline
 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
 \hline
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 \hline
 \end{array}$$

Slika 11.2: Detekcija pina koji je izazvao prekid naredbom `pin_changed = pinb_old ^ PINB;`

U ovisnosti o tome koliko pinova na portu B izaziva prekide, toliko je u prekidnoj rutini `ISR(PCINT0_vect)` potrebno imati provjera pina koji je izazvao prekid. Općenito, provjera za pin  $x$  ( $x = \text{PB0}, \text{PB1}, \text{PB2}, \text{PB3}, \text{PB4}, \text{PB5}, \text{PB6}, \text{PB7}$ ) može se napraviti uvjetom `if (pin_changed & (1 << x))`.

Unutar uvjeta `if (pin_changed & (1 << PB4))` žuta LED dioda mijenja stanje. S obzirom na to da su sve LED diode početno ugašene, ova promjena stanja trenutno će pratiti promjenu stanja crvene LED diode. Je li prekid izazvala promjena signala na pinu PB6 provjeravamo pomoću uvjeta `if (pin_changed & (1 << PB6))`. Unutar ovog uvjeta prikazano je na koji se način može detektirati rastući ili padajući brid signala na pinu PB6. Naime, ako se dogodio PCINT prekid koji je izazvao pin PB6 i ako je stanje pina PB6 nisko, pin je prešao iz visokog stanja u nisko stanje i dogodio se padajući brid. U suprotnom se dogodio rastući brid. Uvjetom `if ((pinb_old & (1 << PB6)))` provjerava se je li staro stanje pina PB6 bilo visoko (imalo vrijednost 1). Ukoliko je staro stanje bilo visoko, tada je padajući brid signala na pinu PB6 izazvao prekid te se plava LED dioda isključuje. U suprotnom se dogodio rastući brid signala na pinu PB6 koji će uključiti plavu LED diodu. Na kraju prekidne rutine `ISR(PCINT0_vect)` trenutno stanje registra `PINB` sprema se u varijablu `pinb_old` kako bi se pri novoj promjeni stanja signala na nekom pinu porta B moglo detektirati koji je pin izazvao prekid.

Prevedite datoteku `vjezba1112.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Pratite što se događa s crvenom i žutom LED diodom te zelenom i plavom LED diodom.

Zakomentirajte poziv funkcija `pcint_pin_enable(PCINT4)` i `pcint_pin_enable(PCINT6)` te ponovno prevedite datoteku `vjezba1112.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Primijetite da se stanja žute i plave LED diode više ne mijenjaju.

Zatvorite datoteku `vjezba1112.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 11.1.1 Zadaci - vanjski prekidi

#### **Zadatak 11.1.1**

Napravite program kojim ćete:

- na pinu INT0 (PD0) brojati padajuće i rastuće bridove signala tipkala spojenog na pin PD0,
- na pinu INT1 (PD1) brojati padajuće bridove signala tipkala spojenog na pin PD1,
- na svaki padajući brid signala na pinu INT0 promijeniti stanje plave LED diode spojene na pin PB7.

Vrijednosti brojača bridova signala na pinovima INT0 i INT1 prikažite na LCD displeju.

---

#### **Zadatak 11.1.2**

Napravite program kojim ćete na proizvoljan način uključivati i isključivati žutu LED diodu spojenu na pin PB5. Stanje žute LED diode potrebno je preslikati na plavu LED diodu spojenu na pin PB7 koristeći PCINT prekid. Dakle, PCINT prekid generirat će se softverski, promjenom stanja na pinu PB5.

---

## Poglavlje 12

# Povezivanje odabranih elektroničkih modula na mikroupravljač

Velik je broj elektroničkih modula koji se mogu spojiti na mikroupravljač ATmega32U4. U ovoj vježbi odabrali smo sljedeće elektroničke module:

- rotacijski enkoder,
- tranzistor kao sklopka i relej,
- ultrazvučni senzor HC-SR04,
- temperaturni senzor LM35,
- servomotor i
- RGB dioda.

Neki od elektroničkih modula nalaze se na razvojnom okruženju sa slike 3.1, dok se ostali elektronički moduli spajaju na konektore razvojnog okruženja. Za svaki od navedenih elektroničkih modula napraviti ćemo programski kod kojim ćemo obrađivati signale elektroničkih modula.

S mrežne stranice [www.vub.hr/mikroracunala](http://www.vub.hr/mikroracunala) skinite datoteku `Moduli.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati **Vaše Ime i Prezime** ne koristeći pritom diakritičke znakove. Na primjer, ako je Vaše ime Ivica Ivić, datoteka koju ćete stvoriti zvat će se `Ivica Ivic`. Datoteku `Moduli.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `VUB mikroracunala.atstln` 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`.

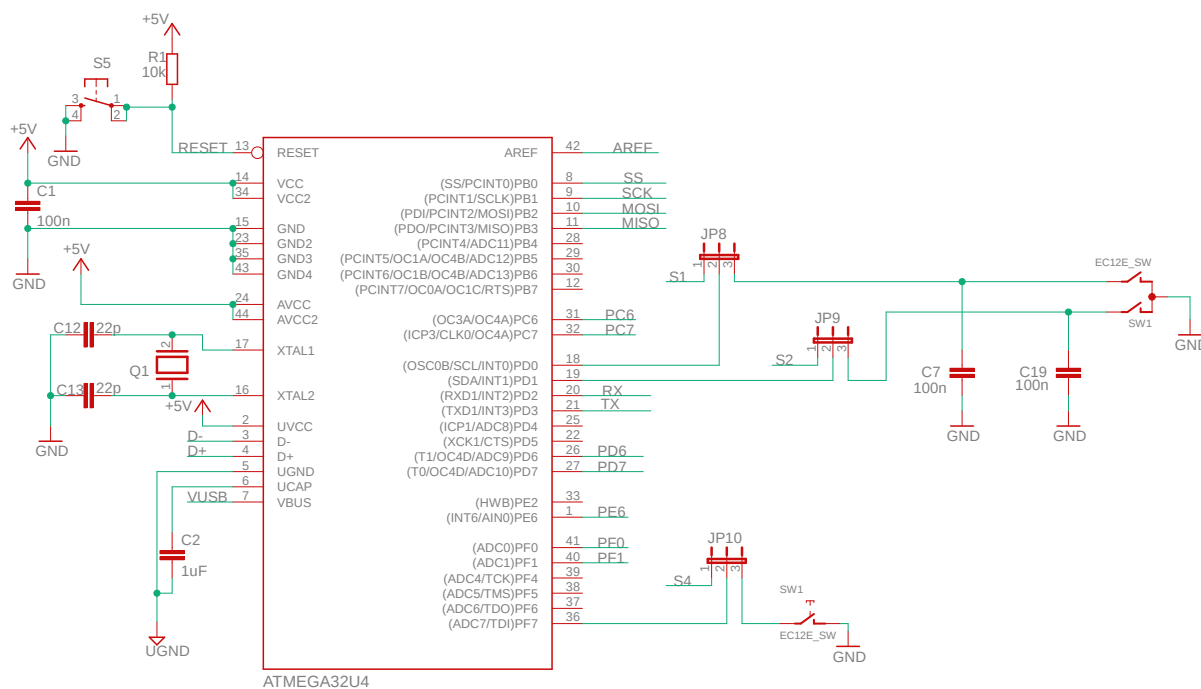
U datoteci s vježbama nalaze se i rješenja vježbi koja možete koristiti za provjeru ispravnosti programskih zadataka.

### 12.1 Rotacijski enkoder

Enkoderi općenito služe za mjerenje kutne i linearne pozicije te kutne i linearne brzine. Iz tog razloga postoje rotacijski i linearni enkoderi. Rotacijski enkoderi mjere kutnu brzinu i poziciju kuta, a linearni enkoderi mjere linearnu brzinu i linearnu poziciju.

Rotacijski enkoderni najčešće se koriste za mjerenje brzine vrtnje i pozicije kuta osovine radnog stroja. Ta osovina može biti povezana na rotor električnog motora ili na neki prijenosnik u radnom stroju. Rotacijski enkoder kakav ćemo mi koristiti u vježbi koristi se za zadavanje referentne vrijednosti. Okretanjem rotacijskog enkodera u jednom smjeru referentna se vrijednost povećava, a okretanjem u drugom smjeru referentna se vrijednost smanjuje. Shema spajanja rotacijskog enkodera na mikroupravljač ATmega32U4 prikazana je na slici 12.1. Korišteni rotacijski enkoder ima dva kanala: A i B kanal koji generiraju niz impulsa koji su pomaknuti za  $90^\circ$  (slika 12.2). Kanal A spojen je na pin PD0 (INT0), a kanal B spojen je na pin PD1 (INT1). Detektiranjem impulsa na kanalu A i B možemo mjeriti i poziciju kuta i smjer vrtnje. Kanal A i B spojeni su na vanjske prekide jer generirani impulsi na kanalu A i B mogu imati visoku frekvenciju pa je važno "uhvatiti" svaki impuls.

Dodatno, ovaj rotacijski enkoder ima tipkalo koje je spojeno na pin PF7, a kojim se može resetirati kutna pozicija rotacijskog enkodera. Kao što možemo primijetiti, kanal A, kanal B i tipkalo rotacijskog enkodera dijele iste pinove kao i tipkala koja smo dosad koristili u vježbama. Stoga, da bismo koristili rotacijski enkoder, kratkospojnike JP8, JP9 i JP10 potrebno je spojiti između trnova 2 i 3 (slika 12.1).



Slika 12.1: Shema spajanja rotacijskog enkodera na mikroupravljač ATmega32U4

Obrada impulsa koji se generiraju kanalima A i B nije jednostavan zadatak. Postoji više pristupa u obradi signala. Pristupi se razlikuju u obradi broja rastućih i padajućih bridova signala na kanalu A i/ili kanalu B. Da bismo objasnili obradu signala i na taj način mjerenje kutne pozicije, pogledat ćemo sliku 12.2.

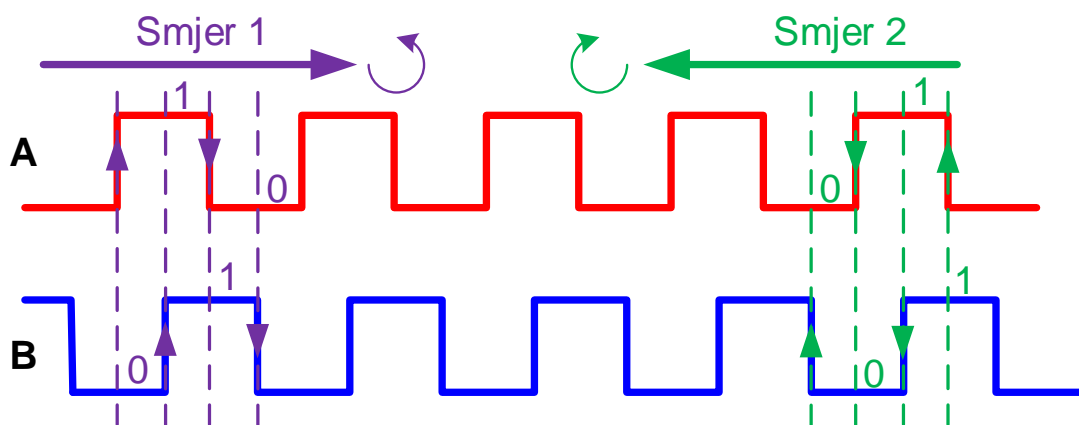
Na slici 12.2 definirani su smjerovi: Smjer 1 i Smjer 2. Jedan je od smjerova rotacija enkodera u smjeru kazaljke na satu, a drugi je smjer rotacija enkodera u smjeru suprotnom od kazaljke na satu. Na našem je razvojnom okruženju Smjer 1 rotacija suprotna od kazaljke na satu, a Smjer 2 rotacija u smjeru kazaljke na satu.

Pri obradi impulsa s rotacijskog enkodera mogu se koristiti i rastući i padajući brid signala na kanalu A, a može se koristiti i samo rastući brid signala na kanalu A. Razlika će biti samo u frekvenciji, odnosno u preciznosti mjerenja pozicije. U oba slučaja, kanal A služi za generiranje

impulsa i obrađuje se kao vanjski prekid, dok kanal B služi za detektiranje smjera. Kada se na kanalu A obrađuju i rastući i padajući brid signala, tada govorimo o načinu korištenja rotacijskog enkodera u dva kvadranta. Ako se na kanalu A obrađuje samo rastući brid signala, tada govorimo o načinu korištenja rotacijskog enkodera u jednom kvadrantu. Za obradu u jednom kvadrantu vrijedi:

- kada se na kanalu A detektira rastući brid, a stanje kanala B je 0, tada je smjer vrtnje Smjer 1,
- kada se na kanalu A detektira rastući brid, a stanje kanala B je 1, tada je smjer vrtnje Smjer 2.

Primijetimo da u ovom pristupu kanal B zapravo određuje smjer vrtnje. Za Smjer 1 brojač impulsa (pozicije) možemo smanjivati za jedan, a za Smjer 2 brojač impulsa (pozicije) možemo povećavati za jedan.



Slika 12.2: Prikaz rastućih i padajućih bridova signala na kanalima A i B rotacijskog enkodera koji su pomaknuti za  $90^\circ$

Za Smjer 1 i način rada u dva kvadranta vrijedi:

- kada se na kanalu A detektira rastući brid, stanje kanala B jest 0,
- kada se na kanalu A detektira padajući brid, stanje kanala B jest 1.

Za Smjer 2 i način rada u dva kvadranta vrijedi:

- kada se na kanalu A detektira rastući brid, stanje kanala B jest 1,
- kada se na kanalu A detektira padajući brid, stanje kanala B jest 0.

Primijetimo da u ovom pristupu kanal B ponovno određuje smjer vrtnje. Postoji i način korištenja rotacijskog enkodera u četiri kvadranta. U tom je slučaju i na kanalu A i na kanalu B potrebno detektirati i rastuće i padajuće bridove. Za Smjer 1 i način rada u četiri kvadranta vrijedi:

- kada se na kanalu A detektira rastući brid, stanje kanala B jest 0,
- kada se na kanalu B detektira rastući brid, stanje kanala A jest 1,
- kada se na kanalu A detektira padajući brid, stanje kanala B jest 1,
- kada se na kanalu B detektira padajući brid, stanje kanala A jest 0.

Za Smjer 2 i način rada u četiri kvadranta vrijedi:

- kada se na kanalu A detektira rastući brid, stanje kanala B jest 1,
- kada se na kanalu B detektira padajući brid, stanje kanala A jest 1.
- kada se na kanalu A detektira padajući brid, stanje kanala B jest 0,
- kada se na kanalu B detektira rastući brid, stanje kanala A jest 0,

Način rada u četiri kvadranta zahtijeva da se kanali A i B obrađuju pomoću vanjskih prekida.



### Vježba 12.1.1

Napravite program kojim ćete pomoću rotacijskog enkodera zadavati referentnu vrijednost u intervalu od [0, 100]. Tipkalom na rotacijskom enkoderu referentna vrijednost postavlja se na 0. U prvom retku LCD displeja potrebno je ispisati referentnu vrijednost, a drugi redak LCD displeja potrebno je popunjavati (simulirati *loading*) u ovisnosti o referentnoj vrijednosti. Shema spajanja rotacijskog enkodera na mikroupravljač ATmega32U4 prikazana je na slici 12.1.

U projektnom stablu otvorite datoteku `vjezba1211.cpp`. Omogućite samo prevođenje datoteke `vjezba1211.cpp`. Početni sadržaj datoteke `vjezba1211.cpp` prikazan je programskim kodom 12.1.

Programski kod 12.1: Početni sadržaj datoteke `vjezba1211.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"
#include "DigitalIO/DigitalIO.h"

// ulazi za rotacijski enkoder
DigitalInput Encoder_ChannelA(D0);
DigitalInput Encoder_ChannelB(D1);
DigitalInput Encoder_PushButton(F7);

int8_t encoder_position = 0;
bool new_position = true;

#define ENCODER_LOW_LEVEL 0
#define ENCODER_HIGH_LEVEL 100

void encoder_inc_dec(bool direction) {
    // direction = true - Smjer 1, direction = false - Smjer 2
    // Smjer 1 - u smjeru kazaljke na satu
    // Smjer 2 - suprotno smjeru kazaljke na satu
    if(direction == false) {
        if (encoder_position < ENCODER_HIGH_LEVEL) {
            encoder_position++;
        }
    }
    else {
        if(encoder_position > ENCODER_LOW_LEVEL) {
            encoder_position--;
        }
    }
}
```



```

    new_position = true;
}

// jednokvadrantni način rada, INTO - rastući brid

ISR(INT0_vect) {
    // direction = true - Smjer 1, direction = false - Smjer 2
    bool direction;
    direction = !Encoder_ChannelB.state();
    encoder_inc_dec(direction);
}

void inicijalizacija() {

    lcd_init(); //konfiguracije LCD displeja
    int0_enable(); // omogući INTO
    int0_set_rising_edge(); // rastući brid generira prekid na INTO
    // pritezni otpornici na ulazima za rotacijski enkoder
    Encoder_ChannelA.pullup_on();
    Encoder_ChannelB.pullup_on();
    Encoder_PushButton.pullup_on();
    interrupt_enable();
}

int main(void){

    inicijalizacija();
    uint8_t loading = 0;
    while(1) {

        if(new_position) {
            lcd_clrscr();
            lcd_home();
            lcd_print("%d\n", encoder_position);
            loading = 16*encoder_position/100.0;
            for(uint8_t i = 0; i < loading; i++) {
                lcd_char(255);
            }
            new_position = false;
        }

        if(Encoder_PushButton.isFalling_edge()) {
            encoder_position = 0;
            new_position = true;
        }
    }
}

```

Rotacijski enkoder spojen je na sljedeće pinove mikroupravljača ATmega32U4:

- kanal A rotacijskog enkodera spojen je na pin PD0 (INT0),
- kanal B rotacijskog enkodera spojen je na pin PD1 (INT1),
- tipkalo rotacijskog enkodera spojeno je na pin PF7.

U vježbi ćemo prikazati načine rada enkodera i obradu signala u jednom kvadrantu, dvama kvadrantima i četirima kvadrantima. U programskom kodu 12.1 prikazan je način rada enkodera u jednom kvadrantu. U ovom je načinu rada na kanalu A potrebno detektirati rastuće bridove, a temeljem stanja signala na kanalu B određuje se smjer vrtnje rotacijskog enkodera. S obzirom na to da postoji potreba za detektiranjem rastućih bridova signala na kanalu A koji je spojen na pin PD0 (INT0), koristit ćemo vanjski prekid INT0.

U programskom kodu 12.1 u funkciji `inicijalizacija()` vanjski prekid `INT0` konfiguriran je tako da se prekidna rutina `ISR(INT0_vect)` poziva na rastući brid signala na pinu `INT0`. Dodatno, globalno su omogućeni prekidi, konfiguriran je LCD displej te su za globalne objekte tipa `DigitalInput`, koji su povezani s rotacijskim enkoderom, uključeni pritezni otpornici.

Na rastući brid signala na kanalu A poziva se prekidna rutina `ISR(INT0_vect)`. U prekidnoj rutini nalazi se varijabla `direction`. Uloga ove varijable jest utvrditi smjer vrtnje rotacijskog enkodera. Varijabla `direction` mora imati vrijednost `false` za Smjer 2 (vrtnja u smjeru kazaljke na satu), odnosno vrijednost `true` za Smjer 1 (vrtnja u smjeru suprotnom od kazaljke na satu) (slika 12.2). Kada se pojavi rastući brid na kanalu A i stanje kanala B jest 0, tada je smjer vrtnje jednak Smjeru 1, a varijabla `direction` imat će vrijednost `true`. Kada se pojavi rastući brid na kanalu A i stanje kanala B jest 1, tada je smjer vrtnje jednak Smjeru 2, a varijabla `direction` imat će vrijednost `false`. Dakle, varijabla `direction` jednaka je invertiranom stanju kanala B.

Nakon što se odredi smjer vrtnje, poziva se funkcija `encoder_inc_dec` koja uvećava za jedan ili smanjuje za jedan varijablu `encoder_position`. Ako varijabla `direction` ima vrijednost `false`, varijabla `encoder_position` uvećava se za jedan jer se rotacijski enkoder vrti u smjeru kazaljke na satu. U suprotnom se varijabla `encoder_position` smanjuje za jedan. Također, varijabla `encoder_position` ne može se uvećati iznad 100, niti smanjiti ispod 0, što smo uvjetovali konstantama `ENCODER_LOW_LEVEL` i `ENCODER_HIGH_LEVEL`. Kada se obradi povećavanje, odnosno smanjenje, varijable `encoder_position`, varijabla `new_position` poprima vrijednost `true` kako bismo u `while` petlji imali informaciju da je dostupna nova kutna pozicija i da ju je potrebno ispisati na LCD displej. Nova vrijednost varijable `encoder_position` ispisuje se u prvom retku LCD displeja, a dodatno se u drugom retku LCD displeja ispisuje znak s ASCII kodom 255 kojim se linearno popunjava drugi redak LCD displeja, u ovisnosti o varijabli `encoder_position` koja poprima vrijednosti iz intervala [0, 100]. U `while` petlji se vrijednost varijable `encoder_position` postavlja na 0 kada se dogodi padajući brid tipkala rotacijskog enkodera.

Prevedite datoteku `vjezba1211.cpp` u strojni kod i snimite ga na mikroupravljač `ATmega32U4`. Testirajte program na razvojnom okruženju s mikroupravljačem `ATmega32U4`. Okrećite rotacijski enkoder u smjeru kazaljke na satu i u suprotnom smjeru te pratite kako se povećava varijabla `encoder_position`. Pritisnite tipkalo rotacijskog enkodera. Za rotacijski enkoder koji se nalazi na razvojnom okruženju način rada u jednom kvadrantu zapravo je i najbolje rješenje. U nastavku će biti pokazan način rada u dvama i četirima kvadrantima koji su vrlo važni kada se koriste optički rotacijski enkodere koje koristimo za pozicioniranje radnog stroja.

U programskom kodu 12.2 prikazani su prekidna rutina `ISR(INT0_vect)` i funkcija `inicijalizacija()` za način rada rotacijskog enkodera u dvama kvadrantima. Ovim programskim kodom potrebno je zamijeniti prekidnu rutinu `ISR(INT0_vect)` i funkciju `inicijalizacija()` u datoteci `vjezba1211.cpp`. U ovom načinu rada na kanalu A potrebno je detektirati i rastuće i padajuće bridove. Smjer vrtnje rotacijskog enkodera dobit će se na temelju stanja kanala A i stanja kanala B nakon što se pojavio rastući odnosno padajući brid signala na kanalu A. S obzirom da postoji potreba za detektiranjem rastućih i padajućih bridova signala na kanalu A koji je spojen na pin `PD0` (`INT0`), koristit ćemo vanjski prekid `INT0`.

Programski kod 12.2: Prekidna rutina `ISR(INT0_vect)` i funkcija `inicijalizacija()` za način rada rotacijskog enkodera u dvama kvadrantima

```
// dvokvadrantni način rada, INT0 - rastući i padajući brid
ISR(INT0_vect) {
    // direction = true - Smjer 1, direction = false - Smjer 2
    bool direction;
    direction = Encoder_ChannelA.state() ^ Encoder_ChannelB.state();
    encoder_inc_dec(direction);
}
```

```

void inicijalizacija() {

    lcd_init(); //konfiguracije LCD displeja
    int0_enable(); // omogući INTO
    int0_set_rising_falling_edge(); // oba brida generiraju prekid na INTO
    // pritezni otpornici na ulazima za rotacijski enkoder
    Encoder_ChannelA.pullup_on();
    Encoder_ChannelB.pullup_on();
    Encoder_PushButton.pullup_on();
    interrupt_enable();
}

```

U programskom kodu 12.2 u funkciji `inicijalizacija()` vanjski prekid `INT0` konfiguriran je tako da se prekidna rutina `ISR(INT0_vect)` poziva na oba brida signala (rastući i padajući) na pinu `INT0`. Ostale naredbe funkcije `inicijalizacija()` nisu promijenjene u odnosu na programski kod 12.1.

Na rastući i padajući brid signala na kanalu A poziva se prekidna rutina `ISR(INT0_vect)`. U prekidnoj rutini ponovno se nalazi varijabla `direction` kojom se određuje smjer vrtnje rotacijskog enkodera. Varijabla `direction` mora imati vrijednost `false` za Smjer 2 (vrtnja u smjeru kazaljke na satu), odnosno vrijednost `true` za Smjer 1 (vrtnja u smjeru suprotnom od kazaljke na satu) (slika 12.2). Ako se kreira kombinacijska tablica za varijablu `direction`, ovisno o stanju na kanalima A i B, primijetiti ćete da se varijabla `direction` može dobiti kao *ex-ili* funkcija stanja na kanalu A i kanalu B (vidi programski kod 12.2). Razlika u načinu rada u jednom i dvama kvadrantima samo je u kreiranju varijable `direction`.

Prevedite datoteku `vjezba1211.cpp` u strojni kod i snimite ga na mikroupravljač `ATmega32U4`. Testirajte program na razvojnom okruženju s mikroupravljačem `ATmega32U4`. Okrećite rotacijski enkoder u smjeru kazaljke na satu i u suprotnom smjeru te pratite kako se povećava varijabla `encoder_position`. Pritisnite tipkalo rotacijskog enkodera.

U programskom kodu 12.3 prikazane su prekidne rutine `ISR(INT0_vect)` i `ISR(INT1_vect)` te funkcija `inicijalizacija()` za način rada rotacijskog enkodera u četirima kvadrantima. Ovim programskim kodom potrebno je zamijeniti prekidnu rutinu `ISR(INT0_vect)` i funkciju `inicijalizacija()` u datoteci `vjezba1211.cpp`. U ovom načinu rada na kanalu A i na kanalu B potrebno je detektirati i rastuće i padajuće bridove. Smjer vrtnje rotacijskog enkodera dobit će se na temelju stanja kanala A i stanja kanala B nakon što se pojavio rastući odnosno padajući brid signala na kanalima A i B. S obzirom na to da postoji potreba za detektiranjem rastućih i padajućih bridova signala na kanalu A koji je spojen na pin `PD0` (`INT0`) i na kanalu B koji je spojen na pin `PD1` (`INT1`), koristit ćemo vanjske prekide `INT0` i `INT1`.

Programski kod 12.3: Prekidne rutine `ISR(INT0_vect)` i `ISR(INT1_vect)` te funkcija `inicijalizacija()` za način rada rotacijskog enkodera u četirima kvadrantima

```

// četverokvadrantni način rada, INT0 i INT1 - rastući i padajući brid
ISR(INT0_vect) {
    // direction = true - Smjer 1, direction = false - Smjer 2
    bool direction;
    direction = Encoder_ChannelA.state() ^ Encoder_ChannelB.state();
    encoder_inc_dec(direction);
}

ISR(INT1_vect) {
    // direction = true - Smjer 1, direction = false - Smjer 2
    bool direction;
    direction = !(Encoder_ChannelA.state() ^ Encoder_ChannelB.state());
    encoder_inc_dec(direction);
}

```

```

void inicijalizacija() {
    lcd_init(); //konfiguracije LCD displeja
    int0_enable(); // omogući INTO
    int0_set_rising_falling_edge(); // oba brida generiraju prekid na INTO
    int1_enable(); // omogući INT1
    int1_set_rising_falling_edge(); // oba brida generiraju prekid na INT1
    // pritezni otpornici na ulazima za rotacijski enkoder
    Encoder_ChannelA.pullup_on();
    Encoder_ChannelB.pullup_on();
    Encoder_PushButton.pullup_on();
    interrupt_enable();
}

```

U programskom kodu 12.3 u funkciji `inicijalizacija()` vanjski prekidi `INT0` i `INT1` konfigurirani su tako da se prekidne rutine `ISR(INT0_vect)` i `ISR(INT1_vect)` pozivaju na oba brida signala (rastući i padajući). Ostale naredbe funkcije `inicijalizacija()` nisu promijenjene u odnosu na programski kod 12.1.

Na rastući i padajući brid signala na kanalu A poziva se prekidna rutina `ISR(INT0_vect)`, dok se na rastući i padajući brid signala na kanalu B poziva se prekidna rutina `ISR(INT1_vect)`. U prekidnim rutinama nalazi se varijabla `direction` kojom se određuje smjer vrtnje rotacijskog enkodera. Varijabla `direction` mora imati vrijednost `false` za Smjer 2 (vrtnja u smjeru kazaljke na satu), odnosno vrijednost `true` za Smjer 1 (vrtnja u smjeru suprotnom od kazaljke na satu) (slika 12.2).

Sada se kombinacijska tablica za varijablu `direction` u ovisnosti o stanjima na kanalu A i kanalu B mora kreirati za slučaj kada se dogodi brid na kanalu A i za slučaj kada se dogodi brid na kanalu B. Varijabla `direction` može se dobiti kao *ex-ili* funkcija stanja na kanalu A i kanalu B kada se pojavi brid na kanalu A (vidi programski kod 12.3). S druge strane, varijabla `direction` može se dobiti kao *ex-nili* funkcija (invertirana *ex-ili*) stanja na kanalu A i kanalu B kada se pojavi brid na kanalu B (vidi programski kod 12.3).

Prevedite datoteku `vjezba1211.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Okrećite rotacijski enkoder u smjeru kazaljke na satu i u suprotnom smjeru te pratite kako se povećava varijabla `encoder_position`. Pritisnite tipkalo rotacijskog enkodera.

Zatvorite datoteku `vjezba1211.cpp` i onemogućite prevođenje ove datoteke.

### 12.1.1 Zadatak - rotacijski enkoder

#### Zadatak 12.1.1

Napravite program kojim ćete pomoću rotacijskog enkodera zadavati referentnu vrijednost u intervalu od  $[0, 400]$  tako da se referentna vrijednost mijenja u koracima iznosa 4 (0, 4, 8, ..., 400). Tipkalom na rotacijskom enkoderu referentna vrijednost postavlja se na 200. U prvom retku LCD displeja potrebno je ispisati referentnu vrijednost, a drugi redak LCD displeja potrebno je popunjavati (simulirati *loading*) u ovisnosti o referentnoj vrijednosti. Shema spajanja rotacijskog enkodera na mikroupravljač ATmega32U4 prikazana je na slici 12.1.

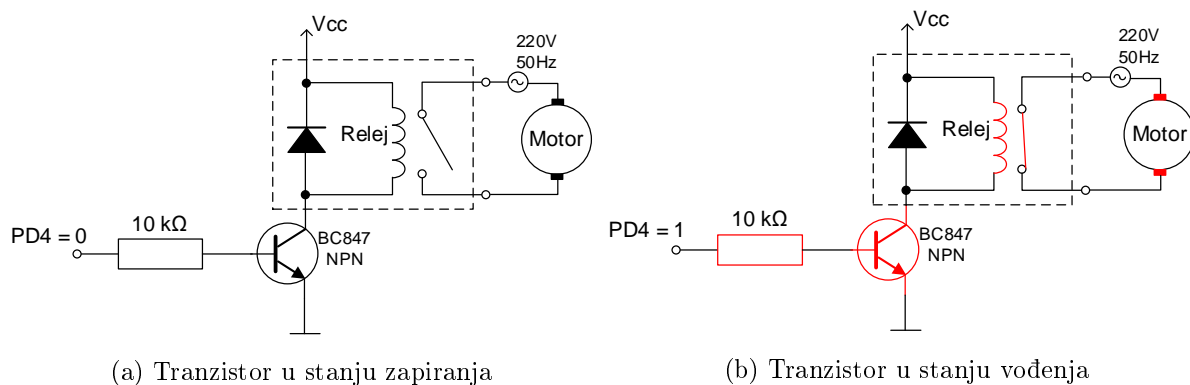
## 12.2 Tranzistor kao sklopka i relej

Bipolarni tranzistor kao sklopka često se koristi za uključenje i isključenje električnih uređaja (trošila) frekvencijom ne većom od 50 kHz. Za veće frekvencije koristi se MOSFET tranzistor [4]. Iako se tranzistor kao sklopka često koristi, on ima i neke nedostatke:

- izlazni napon i struja tranzistora kao sklopke ograničeni su,
- tranzistor kao sklopka na svom izlazu može uklopiti samo istosmjerni napon.

Ove probleme možemo riješiti pomoću releja. Releji su elektromehaničke sklopke koje rade na principu elektromagnetskog polja koje se javlja protjecanjem struje kroz zavojnicu releja. Elektromagnetsko polje uklapa ili isklapa metalnu kotvu te na taj način uključuje ili isključuje električni uređaj (trošilo). Releji mogu uklopiti istosmjerni i izmjenični napon te se mogu dizajnirati za velike struje trošila. Životni vijek releja ovisi o broju uključjenja trošila, što mu je, uz malu frekvenciju rada, glavni nedostatak.

U nastavku ćemo prikazati klasičan primjer koji se koristi za uključenje i isključenje izmjeničnog trošila (u ovom slučaju izmjeničnog električnog motora) pomoću mikroupravljača. Shema spajanja releja na mikroupravljač posredno pomoću tranzistora BC847 prikazana je na slici 12.3.



Slika 12.3: Shema spajanja releja na mikroupravljač posredno pomoću tranzistora BC847

Na slici 12.3 prikazan je bipolarni tranzistor koji kao trošilo uključuje i isključuje relej. Releji kao trošilo uključuje i isključuje izmjenični motor. Pretpostavimo da je struja zavojnice releja (upravljačka struja) veća od 40 mA. Tu struju ne može dati digitalni pin mikroupravljača pa se kao posrednik u uključanju i isključenju releja koristi bipolarni tranzistor. Bazu tranzistora potrebno je spojiti na digitalni pin mikroupravljača (na primjer pin PD4). Releji se postavljaju u kolektorski krug tranzistora jer je s pozicije tranzistora relej trošilo. Paralelno s relejem potrebno je spojiti diodu koja štiti tranzistor od induciranog napona na zavojnici releja u trenutku njegova isključenja. Pretpostavimo da je napon upravljačkog kruga releja jednak  $V_{cc}$  (npr.  $V_{cc} = 5\text{ V}$ ).

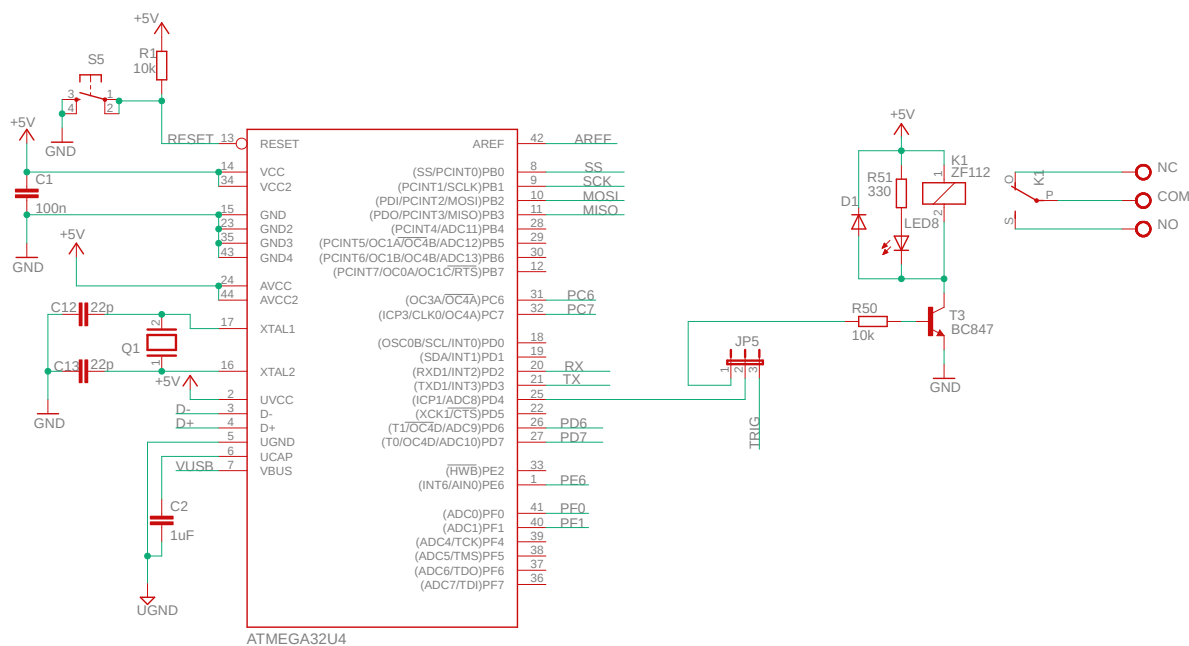
Kada na digitalni pin PD4 dovedemo nisko stanje (0 V), tranzistor će biti u stanju zapiranja (slika 12.3a). U tom slučaju kroz zavojnicu releja ne protječe struja i relej je isključen. Ako je relej isključen, prema slici 12.3a, izmjenični je motor također isključen.

Kada na digitalni pin PD4 dovedemo visoko stanje (5 V), tranzistor će biti u stanju vođenja (slika 12.3b). U tom slučaju kroz zavojnicu releja protječe struja i relej je uključen. Ako je relej uključen, prema slici 12.3b, izmjenični je motor također uključen.

Ovaj primjer pokazuje nam da s malom upravljačkom strujom možemo upravljati velikom strujom trošila. Shema spajanja bipolarnog tranzistora BC847 s relejem u kolektorskom krugu na mikroupravljač ATmega32U4 prikazana je na slici 12.4. Baza bipolarnog tranzistora BC847

spojena je na pin PD4 preko kratkospojnika JP5. Kratkospojnik je potrebno postaviti između trnova 1 i 2. U kolektorskom krugu bipolarnog tranzistora BC847 nalazi se relej u paralelnom spoju s otpornikom i LED diodom koja svijetli kada je relej uključen. Releji na slici 12.4 ima tri kontakta:

- NC (engl. *Normally Closed*) - kontakt koji je zatvoren sa zajedničkim kontaktom kada relej nije uključen,
- NO (engl. *Normally Open*) - kontakt koji je otvoren sa zajedničkim kontaktom kada relej nije uključen,
- COM (engl. *Common*) - zajednički kontakt.



Slika 12.4: Shema spajanja bipolarnog tranzistora BC847 s relejem u kolektorskom krugu na mikroupravljač ATmega32U4

Trošilo se na relej uvijek spaja između kontakata NC i COM ili NO i COM, ovisno o logici uključivanja releja. Na primjer, ako je izmjenični motor spojen između kontakata NO i COM, tada će izmjenični motor biti uključen ako je na pinu PD4 visoko stanje (5 V). Ako je izmjenični motor spojen između kontakata NC i COM, tada će izmjenični motor biti uključen ako je na pinu PD4 nisko stanje (0 V).



### Vježba 12.2.1

Napravite program kojim će se relej koji je spojen na pin PD4 uključivati i isključivati na sljedeći način:

- relej se uključuje ako je tipkalo spojeno na pin PD0 bilo pritisnuto duže od 5 s, a relej je prethodno bio isključen,
- relej se isključuje ako je tipkalo spojeno na pin PD0 bilo pritisnuto duže od 5 s, a relej je prethodno bio uključen.

Schema spajanja bipolarnog tranzistora BC847 s relejem u kolektorskom krugu na mikroupravljač ATmega32U4 prikazana je na slici 12.4.

U projektnom stablu otvorite datoteku `vjezba1221.cpp`. Omogućite samo prevođenje datoteke `vjezba1221.cpp`. Početni sadržaj datoteke `vjezba1221.cpp` prikazan je programskim kodom 12.4.

Programski kod 12.4: Početni sadržaj datoteke `vjezba1221.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "DigitalIO/DigitalIO.h"

int main(void) {
    // objekti za relej i tipkalo
    DigitalOutput Relej(D4);
    DigitalInput Tipkalo1(D0);
    Tipkalo1.pullup_on();
    // pomoćne varijable kojima se prati vrijeme pritisnutog tipkala
    uint16_t time_on = 0;
    bool time_is_up = false;

    while(1) {
        // ako je tipkalo pritisnuto i vrijeme od 5s nije isteklo
        if(Tipkalo1.state() == false && !time_is_up) {
            time_on++;
        }
        // kada se otpusti tipkalo
        if(Tipkalo1.isRising_edge()) {
            time_is_up = false;
        }
        // ako je vrijeme veće od 5s i relej je bio isključen
        if((time_on > 500) && (Relej.state() == false)) {
            Relej.on();
            time_is_up = true;
            time_on = 0;
        }
        // ako je vrijeme veće od 5s i relej je bio uključen
        if((time_on > 500) && (Relej.state() == true)) {
            Relej.off();
            time_is_up = true;
            time_on = 0;
        }
        _delay_ms(10);
    }
    return 0;
}
```

Bipolarni tranzistor koji uključuje relej spojen je na pin PD4. Za potrebe upravljanja relejem stvoren je objekt `Relej` tipa `DigitalOutput` koji je povezan s pinom PD4. Uključivanje releja provodit će se tipkalom na pinu PD0, za što je stvoren objekt `Tipkalo1` tipa `DigitalInput`.

U `main()` funkciji deklarirana je varijabla `time_on` koja se koristi za mjerenje isteka vremena i varijabla `time_is_up` koja detektira istek vremena. U `while` petlji ispituje se je li pritisnuto tipkalo i je li vrijeme od zadanih 5 sekundi isteklo. U slučaju zadovoljenja tih dvaju uvjeta, varijabla `time_on` uvećava se za 1. Ova varijabla uvećava se za 1 svakih 10 ms (kašnjenje u `while` petlji), ako je pritisnuto tipkalo i vrijeme od 5 s nije isteklo. Vrijeme iznosa 5 s isteći će kada varijabla `time_on` poprimi iznos 500. Kada varijabla `time_on` dosegne vrijednost 500, tada:

- relej se uključuje ako je prethodno bio isključen,

- relej se isključuje ako je prethodno bio uključen.

Također, varijabla `time_is_up` poprima vrijednost `true` kako bi se zbog nastavljanja držanja tipkala spojenog na pin PD0 spriječilo periodičko uključenje i isključenje releja. Ponovna promjena stanja releja bit će moguća kada se tipkalo otpusti (na rastući brid signala tipkala).

Prevedite datoteku `vjezba1221.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Držite tipkalo spojeno na pin PD0 najmanje 5 s. Istek vremena umjesto pomoću kašnjenja u `while` petlji može se mjeriti pomoću tajmera u normalnom načinu rada. Korištenje tajmera za istek vremena svakako je bolje rješenje.

Zatvorite datoteku `vjezba1221.cpp` i onemogućite prevođenje ove datoteke.

### 12.2.1 Zadatak - tranzistor kao sklopka i relej

#### Zadatak 12.2.1

Napravite program kojim će se relej koji je spojen na pin PD4 uključivati i isključivati na sljedeći način:

- relej se uključuje ako je tipkalo spojeno na pin PD1 bilo pritisnuto duže od 8 s, a relej je prethodno bio isključen,
- relej se isključuje ako je tipkalo spojeno na pin PD1 bilo pritisnuto duže od 4 s, a relej je prethodno bio uključen.

Shema spajanja bipolarnog tranzistora BC847 s relejem u kolektorskom krugu na mikroupravljač ATmega32U4 prikazana je na slici 12.4.

---

## 12.3 Ultrazvučni senzor HC-SR04

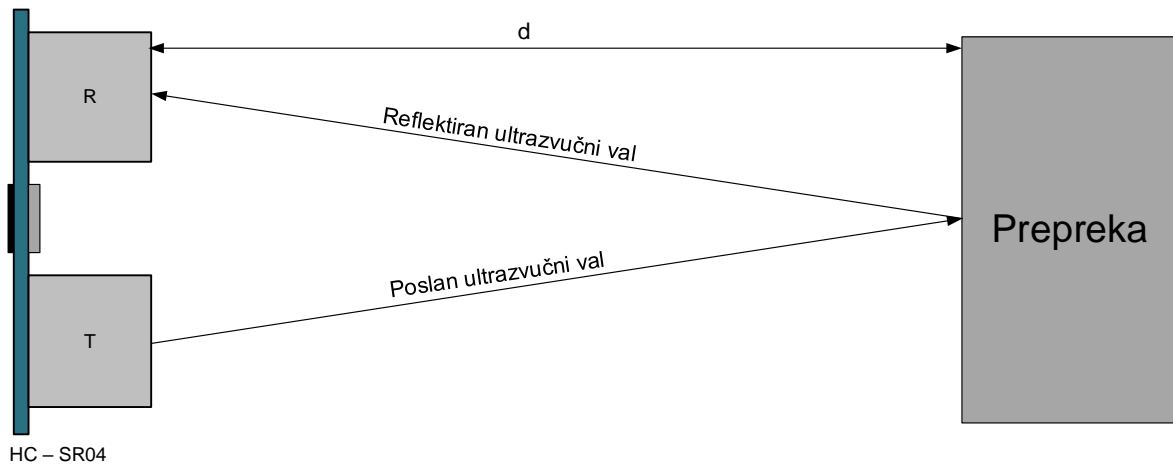
Ultrazvučni senzor HC-SR04 koristi se za mjerenje udaljenosti u rasponu od 2 cm do 4 m, s preciznošću od 3 mm. Ovaj senzor ima zvučnik koji šalje ultrazvučne valove u prostor te mikروفon koji prima reflektirane ultrazvučne valove od prepreka u prostoru (slika 12.5).

Princip rada ultrazvučnog senzora HC-SR04 prikazan je vremenskim dijagramom na slici 12.6 [5]. Pomoću digitalnog pina mikroupravljača na ulazni pin `Trig` ultrazvučnog senzora HC-SR04 potrebno je poslati impuls ne kraći od  $10 \mu\text{s}$ . Ovaj impuls služi za pokretanje mehanizma mjerenja ultrazvučnog senzora HC-SR04. Zvučnik ultrazvučnog senzora HC-SR04 s oznakom `T` (engl. *transmitter*) u prostor šalje ultrazvučni val, odnosno osam impulsa, frekvencijom 40 kHz. Ultrazvučni val odbija se od prepreke i reflektira na mikروفon s oznakom `R` (engl. *receiver*). Mikroupravljač koji se nalazi na ultrazvučnom senzoru HC-SR04 obrađuje reflektirane valove te na izlazni pin `Echo`<sup>1</sup> ultrazvučnog senzora HC-SR04 generira impuls koji traje vrijeme  $t_{ECHO}$ . Vrijeme  $t_{ECHO}$  jednako je vremenu koje je potrebno ultrazvučnom valu da od ultrazvučnog senzora HC-SR04 dođe do prepreke i nazad, a potrebno ga je mjeriti pomoću mikroupravljača.

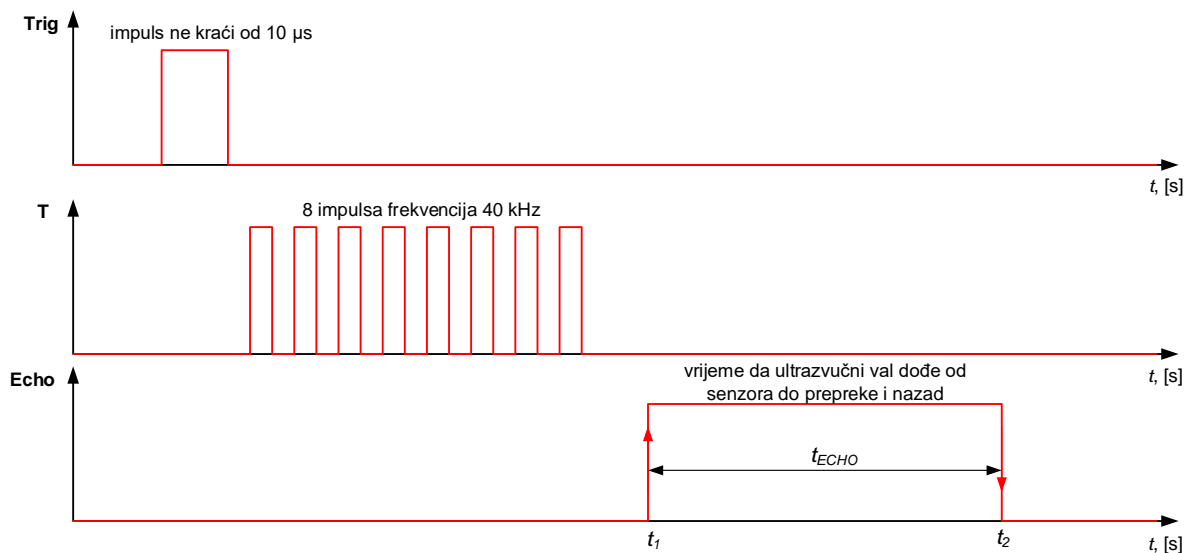
---

<sup>1</sup>Tip izlaza jest *push-pull*. Ovakav izlaz može se spojiti na ulaz mikroupravljača bez uključenja priteznog otpornika.





Slika 12.5: Rasprostiranje ultrazvučnog vala kroz prostor



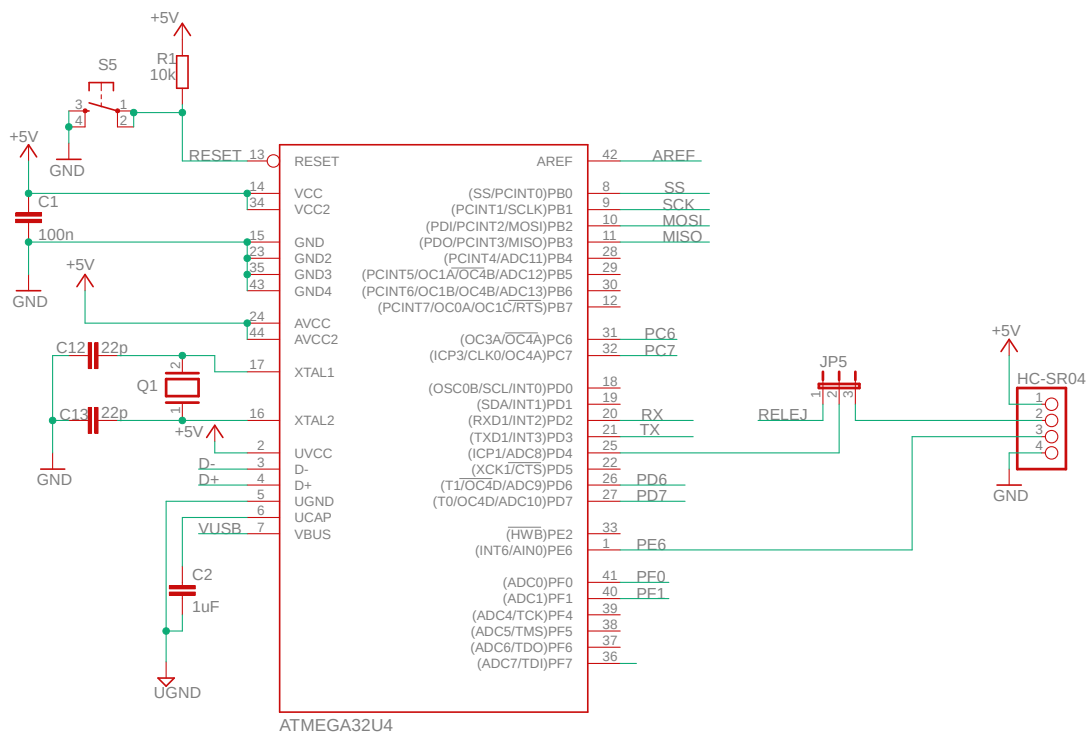
Slika 12.6: Vremenski dijagram signala ultrazvučnog senzora HC-SR04

Shema spajanja ultrazvučnog senzora HC-SR04 na mikroupravljač ATmega32U4 prikazana je na slici 12.7. Pin Trig ultrazvučnog senzora HC-SR04 spojen je na pin mikroupravljača PD4 preko kratkospojnika JP5 (kratkospojnik je potrebno postaviti između trnova 2 i 3). Na pin mikroupravljača PE6 (INT6) spojen je pin Echo ultrazvučnog senzora HC-SR04. Razlog tomu jest to što se generirani signal na pinu Echo mora obrađivati vanjskim prekidačem. Povezivanje ultrazvučnog senzora HC-SR04 s konektorom na razvojno okruženje prikazano je na slici 12.8. Ultrazvučni senzor HC-SR04 nije sastavni dio razvojnog okruženja na slici 3.1, već se spaja na definiran konektor tako da su zvučnik i mikrofon senzora okrenuti od pločice razvojnog okruženja.

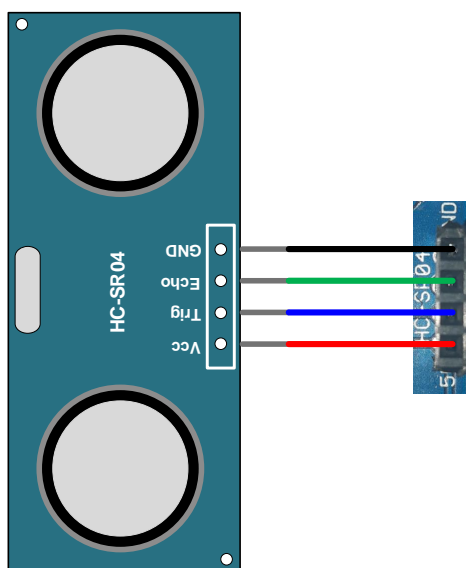
Udaljenost prepreke od ultrazvučnog senzora HC-SR04 možemo izračunati pomoću relacije:

$$d = \frac{t_{ECHO} \cdot v_z}{2}, \quad (12.1)$$

gdje je  $v_z$  brzina zvuka u zraku koja iznosi 343 m/s na temperaturi od 20 °C. Umnožak  $t_{ECHO} \cdot v_z$  u relaciji (12.1) dijeli se s 2 jer ultrazvučni val prevaljuje dvije udaljenosti od prepreke do ultrazvučnog senzora HC-SR04.



Slika 12.7: Shema spajanja ultrazvučnog senzora HC-SR04 na mikroupravljač ATmega32U4



Slika 12.8: Povezivanje ultrazvučnog senzora HC-SR04 s konektorom na razvojno okruženje sa slike 3.1

Ovisnost brzine zvuka u zraku o temperaturi zraka dana je relacijom:

$$v_z = 331 + 0.6 \cdot T, \quad (12.2)$$

gdje je  $T$  temperatura zraka u  $^{\circ}\text{C}$ . Relacijom (12.2) može se napraviti korekcija brzine zvuka u relaciji (12.1) u ovisnosti o temperaturi zraka.



### Vježba 12.3.1

Napravite program kojim ćete pomoću ultrazvučnog senzora HC-SR04 mjeriti udaljenost do prepreke u prostoru sa i bez korekcije brzine zvuka. Mjerenu udaljenost prikažite na LCD displeju u centimetrima. Shema spajanja ultrazvučnog senzora HC-SR04 na mikroupravljač ATmega32U4 prikazana je na slici 12.7.

U projektnom stablu otvorite datoteku `vjezba1231.cpp`. Omogućite samo prevođenje datoteke `vjezba1231.cpp`. Početni sadržaj datoteke `vjezba1231.cpp` prikazan je programskim kodom 12.5.

Programski kod 12.5: Početni sadržaj datoteke `vjezba1231.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "Timer/timer.h"
#include "Interrupt/interrupt.h"

volatile uint32_t broj_impulsa = 0;
bool hcsr04_measured = false;
volatile bool hcsr04_triggered = false;

// prekidna rutina za INT6
ISR(INT6_vect) {
    // ako je brid rastuci
    if(get_pin(PINE,PE6) == 1) {
        TCNT1 = 0; // t1 = 0
    }
    else { // ako je brid padajuci
        broj_impulsa = TCNT1;
        hcsr04_measured = true;
    }
}

void inicijalizacija() {
    lcd_init(); // inicijalizacija LCD displeja
    // inicijalizacija AD pretvorbe
    // oba brida generiraju prekid INT6
    int6_set_rising_falling_edge();
    int6_enable();
    // tajmer 1 u normalnom načinu rada
    timer1_set_normal_mode();
    timer1_set_prescaler(TIMER1_PRESCALER_8); //F_CPU/8
    // konfiguracija pinova za hcsr04
    output_port(DDRD,PD4); // trigg pin (PD4) - izlazni pin
    input_port(DDRE,PE6); // echo pin (PE6) - ulazni pin
    interrupt_enable(); // omogući prekide
}

void hcsr04_trigg() {
    set_port(PORTD, 4, 1);
    _delay_us(20); // trigger impuls 20 us
    set_port(PORTD, 4, 0);
    hcsr04_triggered = true;
}

int main(void) {
```

```

inicijalizacija();

float d, d_cor; // udaljenost
float t_echo; // trajanje impulsa na Echo pinu
uint16_t time_delay = 0;
uint16_t ADC_4; // vrijednost AD pretvorbe na pinu ADC4
float T; // temperatura
float v_z; // brzina zvuka u zraku

while(1) {
    // ako nije poslan trigg i ako je istekla 1 s
    if(!hcsr04_triggered && time_delay == 0) {
        hcsr04_trigg(); // pošalji trigg signal na hcsr04
    }
    // ako je mjerenje završeno
    if(hcsr04_measured) {
        t_echo = broj_impulsa * 8.0 / F_CPU; // impuls Echo u s
        v_z = 343.0; // brzina zvuka na 20 C
        d = t_echo / 2.0 * v_z * 100.0; // udaljenost u cm

        // napraviti dio programskog koda za korigiranu udaljenost

        lcd_clrscr();
        lcd_home();
        lcd_print("d1 = %0.2f cm\n", d);
        lcd_print("d2 = %0.2f cm", d_cor);
        hcsr04_measured = false;
        hcsr04_triggered = false;
    }
    _delay_ms(10);
    // ako je prošla 1 s, vrati brojač vremena na 0 s
    if(++time_delay > 100) {
        time_delay = 0;
    }
}
}

```

Ulazni pin Trig ultrazvučnog senzora HC-SR04 spojen je prema shemi na slici 12.7 na pin PD4. Na pinu PD4 generirat ćemo impuls ne kraći od 10  $\mu$ s koji pokreće mehanizam mjerenja udaljenosti. U programskom kodu 12.5 za generiranje impulsa trajanja 20  $\mu$ s na pinu Trig koristi se funkcija `hcsr04_trigg()`.

Proces mjerenja udaljenosti prepreke od ultrazvučnog senzora HC-SR04 pokrećemo pozivom funkcije `hcsr04_trigg()` svaku sekundu. Ponovno na pametan način postavljamo kašnjenje u `while` petlju tako da petlja može brzo odreagirati na zahtjev za ispisom mjerenja. Kašnjenje petlje je 10 ms, a funkciju `hcsr04_trigg()` pozivamo na svaku stotu iteraciju `while` petlje.

Impuls koji generira izlazni pin Echo ultrazvučnog senzora HC-SR04 mjerit ćemo pomoću sklopa *Timer/Counter1* u normalnom načinu rada i vanjskog prekida na sljedeći način:

- kada se pojavi rastući brid signala na pinu Echo (slika 12.6), pokrenite mjerenje vremena pomoću tajmera postavljenjem početne vrijednosti registra `TCNT1` na nulu,
- kada se pojavi padajući brid signala na pinu Echo, pročitajte vrijednost registra `TCNT1` i spremite je u varijablu `broj_impulsa`,
- pomoću vrijednosti varijable `broj_impulsa` odredite vrijeme  $t_{ECHO}$ ,
- udaljenost prepreke od ultrazvučnog senzora HC-SR04 izračunajte prema relaciji (12.1).

Trajanje jednog impulsa koji broji registar `TCNT1` ovisi o djelitelju frekvencije radnog takta i frekvenciji radnog takta, a može se izračunati na sljedeći način:

$$t_{\text{impulsa}} = \frac{\text{PRESCALER}}{\text{F\_CPU}}. \quad (12.3)$$

Vrijeme trajanja impulsa na pinu Echo ultrazvučnog senzora HC-SR04 možemo izračunati tako da ukupan broj impulsa koji je spremljen u varijablu `broj_impulsa` pomnožimo s trajanjem jednog impulsa:

$$t_{\text{ECHO}} = \text{broj\_impulsa} \cdot t_{\text{impulsa}} = \text{broj\_impulsa} \frac{\text{PRESCALER}}{\text{F\_CPU}}. \quad (12.4)$$

Frekvencija radnog takta u našem slučaju iznosi 16 MHz, a za djelitelj frekvencije radnog takta odabrat ćemo 8. Uvrstimo sada navedene parametre u relaciju (12.4) kako bismo dobili trajanje impulsa na pinu Echo:

$$t_{\text{ECHO}} = \text{broj\_impulsa} \frac{8}{16000000} = \frac{\text{broj\_impulsa}}{2000000}. \quad (12.5)$$

Temeljem izračunatog vremena  $t_{\text{ECHO}}$ , udaljenost prepreke od ultrazvučnog senzora HC-SR04 može se izračunati prema relaciji (12.1) uz konstantnu brzinu zvuka ili uz korekciju brzine zvuka u ovisnosti o temperaturi koja se može dobiti relacijom (12.2). Udaljenost dobivena relacijom (12.1) ima mjernu jedinicu m. Naš je zadatak prikazati udaljenost prepreke od ultrazvučnog senzora HC-SR04 u centimetrima pa ćemo relaciju (12.1) pomnožiti s konstantom  $\frac{100\text{cm}}{1\text{m}}$ . Udaljenost prepreke od ultrazvučnog senzora HC-SR04 u centimetrima bez korekcije brzine zvuka može se dobiti prema relaciji:

$$d[\text{cm}] = \frac{t_{\text{ECHO}}}{2} \cdot 343 \cdot 100. \quad (12.6)$$

Udaljenost prepreke od ultrazvučnog senzora HC-SR04 u centimetrima s korekcijom brzine zvuka s obzirom na temperaturu zraka  $T$  može se dobiti prema relaciji:

$$d[\text{cm}] = \frac{t_{\text{ECHO}}}{2} \cdot (331 + 0.6 \cdot T) \cdot 100. \quad (12.7)$$

U programskom kodu 12.5 u funkciji `inicijalizacija()` vanjski prekid INT6 konfiguriran je tako da se zahtjev za prekid generira na rastući i padajući brid signala. Za mjerenje vremena  $t_{\text{ECHO}}$  koristit ćemo sklop `Timer/Counter1` kao tajmer u normalnom načinu rada. Djelitelj frekvencije radnog takta postavljen je na vrijednost 8. U ovom slučaju sklop `Timer/Counter1` ne treba generirati prekid prilikom preljeva u registru `TCNT1` jer nam je potrebna samo informacija o broju impulsa u registru `TCNT1`. Također, u funkciji `inicijalizacija()` globalno su omogućeni prekidi te je pin PD4 konfiguriran kao izlazni, a pin PE6 (INT6) konfiguriran je kao ulazni pin. Pin Echo ultrazvučnog senzora HC-SR04 izlazni je pin tipa *push-pull* pa nije potrebno uključiti pritezni otpornik na pinu PE6.

Prekidna rutina `ISR(INT6_vect)` poziva se kod svakog padajućeg i rastućeg brida signala na pinu INT6 (PE6). Ako je brid signala na pinu INT6 (PE6) rastući, to znači da je pri pozivu prekidne rutine `ISR(INT6_vect)` stanje pina PE6 visoko. Ovu provjeru radimo pomoću naredbe `if(get_pin(PINE, PE6) == 1)`. Ako je brid signala na pinu PE6 rastući, početno stanje registra `TCNT1` postaviti ćemo na nulu. Sljedeći poziv prekidne rutine javit će se pri padajućem bridu na pinu INT6 (PE6). Kada se pojavi padajući brid signala na pinu PE6, vrijednost registra `TCNT1` sprema se u varijablu `broj_impulsa`. U prekidnoj rutini `ISR(INT6_vect)` varijabla `hcsr04_measured` postavlja se u vrijednost `true` kada je mjerenje vremena završilo. Ova varijabla koristi se u `while` petlji kako bi se izračun udaljenosti i ispis udaljenosti provodio samo kada je dostupno novo mjerenje.

U `while` petlji udaljenost se zvučnog senzora HC-SR04 od prepreke izračunava u centimetrima pomoću relacija (12.5) i (12.6) te se na LCD displej ispisuje na dva decimalna mjesta. Brzina zvuka u ovom se slučaju pretpostavlja konstantnom iznosa 343 m/s.

Programski kod 12.6 služi za mjerenje udaljenosti ultrazvučnog senzora HC-SR04 od prepreke, uz korekciju brzine zvuka. Taj programski kod napišite u datoteku `vjezba1231.cpp`. Udaljenost ultrazvučnog senzora HC-SR04 od prepreke izračunava se u centimetrima pomoću relacije (12.7), uz brzinu zvuka koja se izračunava prema relaciji (12.2). Temperatura okoline mjeri se pomoću NTC otpornika, kako smo pokazali u vježbi 7.1.5. Zbog korištenja analogno-digitalne pretvorbe, potrebno je inicijalizirati AD pretvorbu i uključiti zaglavlje za AD pretvorbu.

Programski kod 12.6: Programski kod za mjerenje udaljenosti ultrazvučnog senzora HC-SR04 od prepreke korekcijom brzine zvuka

```
ADC_4 = adc_read(ADC4); // AD pretvorba na kanalu ADC4
// temperatura NTC otpornika
T = 3435 / (log(ADC_4 / (1023.0 - ADC_4)) + 10.861) - 273.15;
v_z = 331.0 + 0.6 * T; // brzina zvuka na temperaturi T
d_cor = t_echo / 2.0 * v_z * 100.0; // udaljenost u cm
```

Prevedite datoteku `vjezba1231.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Usporedite udaljenosti od prepreka sa i bez korekcije brzine zvuka.

Zatvorite datoteku `vjezba1231.cpp` i onemogućite prevođenje ove datoteke.

### 12.3.1 Zadatak - ultrazvučni senzor HC-SR04

#### Zadatak 12.3.1

Napravite program kojim ćete pomoću ultrazvučnog senzora HC-SR04 mjeriti udaljenost do prepreke u prostoru s korekcijom brzine zvuka. Mjerenu udaljenost prikažite na LCD displeju u mm. Uključenjem crvene LED diode signalizirajte da se prepreka od ultrazvučnog senzora HC-SR04 nalazi na udaljenosti manjoj od 100 mm. Shema spajanja ultrazvučnog senzora HC-SR04 na mikroupravljač ATmega32U4 prikazana je na slici 12.7.

## 12.4 Temperaturni senzor LM35

Temperaturni senzor LM35 visoko je precizni senzor temperature s mjernim opsegom od  $-55^{\circ}\text{C}$  do  $150^{\circ}\text{C}$  [6]. Ovaj senzor na izlaznom pinu  $V_{out}$  generira napon koji je proporcionalan temperaturi u okolini senzora s konstantom proporcionalnosti koja iznosi  $10\text{ mV}/^{\circ}\text{C}$ , prema relaciji:

$$V_{out} = T \cdot 10 \frac{\text{mV}}{^{\circ}\text{C}} = T \cdot 0,01 \frac{\text{V}}{^{\circ}\text{C}}. \quad (12.8)$$

Temperaturu iz relacije (12.8) možemo izračunati na sljedeći način:

$$T = V_{out} \cdot 100 \frac{^{\circ}\text{C}}{\text{V}}. \quad (12.9)$$

Na primjer, ako je izlazni napon temperaturnog senzora LM35 jednak 350 mV, temperatura u njegovoj okolini jest  $35^{\circ}\text{C}$ .



Programski kod 12.7: Početni sadržaj datoteke vjezba1241.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija() {
    // konfigurirajte ADC i LCD
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_5; // vrijednost AD pretvorbe na pinu ADC5
    float U_ADC5; // napon na pinu ADC5
    const float V_REF = 5.0; // AVCC je referentni napon
    float T; // temperatura u okolini senzora LM35

    while (1) {

        ADC_5 = adc_read(ADC5);
        // izračunajte napon U_ADC5 i temperaturu T

        lcd_clrscr();
        lcd_home();
        lcd_print("T = %0.2f°C\n", T, 223);
        _delay_ms(1000);
    }

    return 0;
}
```

U programskom kodu 12.7 u funkciji `inicijalizacija()` konfigurirajte analogno-digitalnu pretvorbu i LCD displej. U `while` petlji svakih se 1000 ms na LCD displeju ispisuje vrijednost temperature u okolini temperaturnog senzora LM35. Za izračunavanje napona na ADC5 pinu i temperature okoline temperaturnog senzora LM35 koristite relacije (12.10) i (12.11).

Prevedite datoteku `vjezba1241.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku `vjezba1241.cpp` i onemogućite prevođenje ove datoteke.

### 12.4.1 Zadatak - temperaturni senzor LM35

#### Zadatak 12.4.1

Napravite program kojim ćete pomoću temperaturnih senzora LM35 i NTC mjeriti temperaturu okoline. Mjerenu temperaturu prikažite na LCD displeju u °C. Shema spajanja temperaturnog senzora LM35 na mikroupravljač ATmega32U4 prikazana je na slici 12.9, a shema spajanja temperaturnog senzora NTC na mikroupravljač ATmega32U4 prikazana je na slici 7.1.



## 12.5 Servomotor

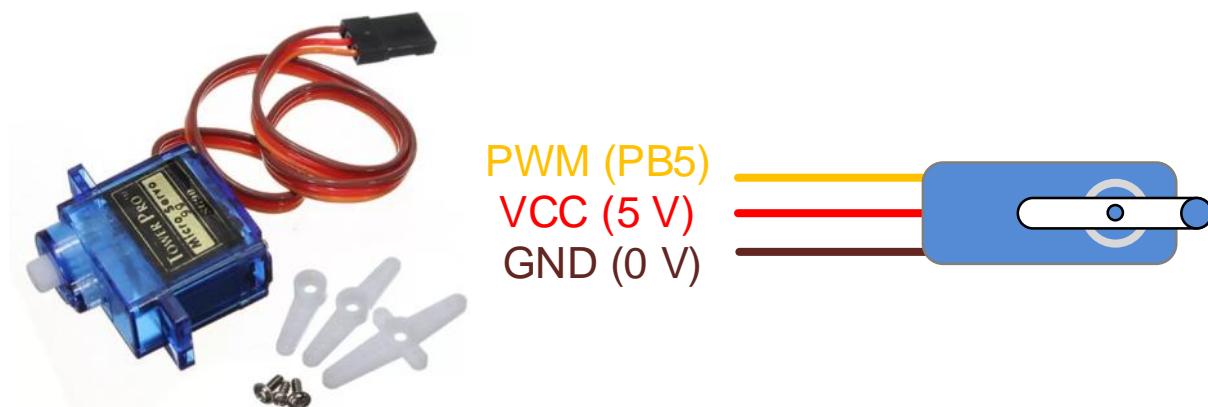
Servomotor je sustav koji se sastoji od istosmjernog motora, upravljačke elektronike koja služi za pozicioniranje izlazne osovine, prijenosnika snage koji okretanje rotora motora prenosi na izlaznu osovinu i mjernog senzora pozicije (potenciometar, enkoder, ...). Dakle, servomotor je istosmjerni motor koji je upravljan po poziciji. Servomotori manjih dimenzija (hobi servomotori) često se koriste u izradi prototipa robotskih ruku, robota inspiriranih prirodom, elemenata mobilnih robota koji se zakreću i slično. Za servomotore specifični su mala brzina okretanja i veliki moment na izlaznoj osovini [7]. Servomotor ima tri žice (slika 12.10):

- smeđa žica povezuje se na GND ili 0 V,
- crvena žica povezuje se na VCC ili 5 V,
- narančasta je žica signalna žica (signalna linija) i na nju se pomoću PWM signala šalje referentna vrijednost pozicije izlazne osovine.

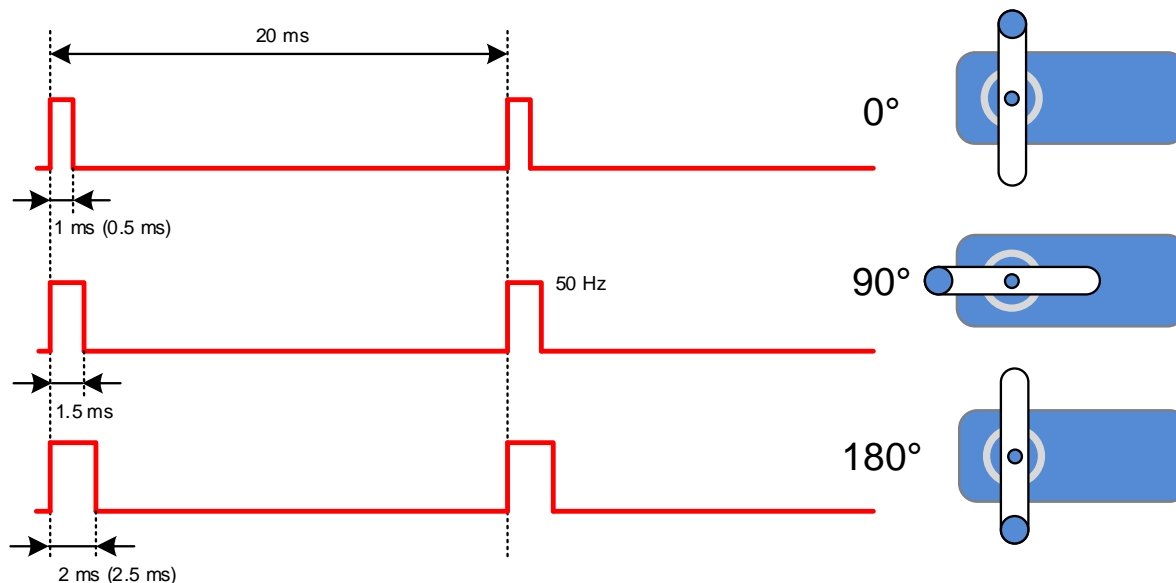
Pozicioniranje servomotora provodi se pomoću periodičkog signala (PWM signala) frekvencije 50 Hz (period 20 ms). PWM signal (slika 12.11) koji se šalje na servomotor generira se pomoću mikroupravljača.

Šrina impulsa PWM signala određuje zakret izlazne osovine. U pravilu bi pozicioniranje servomotora trebalo funkcionirati na sljedeći način (prema slici 12.11):

- ako širina PWM signala iznosi 1 ms (*duty cycle* iznosi 5 %), tada će izlazna osovina servomotora biti zakrenuta za 0°,
- ako širina PWM signala iznosi 1,5 ms (*duty cycle* iznosi 7,5 %), tada će izlazna osovina servomotora biti zakrenuta za 90°,
- ako širina PWM signala iznosi 2 ms (*duty cycle* iznosi 10 %), tada će izlazna osovina servomotora biti zakrenuta za 180°.



Slika 12.10: Povezivanje servomotora s napajanjem i mikroupravljačem



Slika 12.11: Pozicioniranje servomotora

Širina impulsa za zakret izlazne osovine iznosa  $0^\circ$  i  $180^\circ$  često se razlikuje od širine impulsa definirane slikom 12.11. Naime, kod nekih proizvođača širina impulsa PWM signala za zakret izlazne osovine iznosa  $0^\circ$  iznosi oko 0,5 ms, a širina PWM signala za zakret izlazne osovine iznosa  $180^\circ$  iznosi oko 2,5 ms. Iz tog je razloga važno testirati servomotor kako bi se utvrdila minimalna i maksimalna širina impulsa. Širina impulsa  $T_D$  [ms] za zadani zakret  $\alpha$  može se za servomotor na slici 12.11 odrediti prema relaciji:

$$T_D [\text{ms}] = 1 + \frac{1}{180^\circ} \alpha [^\circ] \quad (12.12)$$

Širina PWM signala  $duty$  [%] za zadani zakret  $\alpha$  može se odrediti prema relaciji:

$$duty [\%] = \frac{T_D}{20\text{ms}} = 5\% + \frac{5\%}{180^\circ} \alpha [^\circ] \quad (12.13)$$

Širina impulsa  $T_D$  [ms] za zadani zakret  $\alpha$  za servomotore koji se ne ponašaju prema slici 12.11 može se odrediti relacijom:

$$T_D [\text{ms}] = T_{min} + \frac{T_{max} - T_{min}}{180^\circ} \alpha [^\circ] \quad (12.14)$$

gdje su:

- $T_{min}$  - širina impulsa za zakret izlazne osovine iznosa  $0^\circ$ ,
- $T_{max}$  - širina impulsa za zakret izlazne osovine iznosa  $180^\circ$ .

Širina PWM signala  $duty$  [%] za zadani zakret  $\alpha$  može se odrediti prema relaciji:

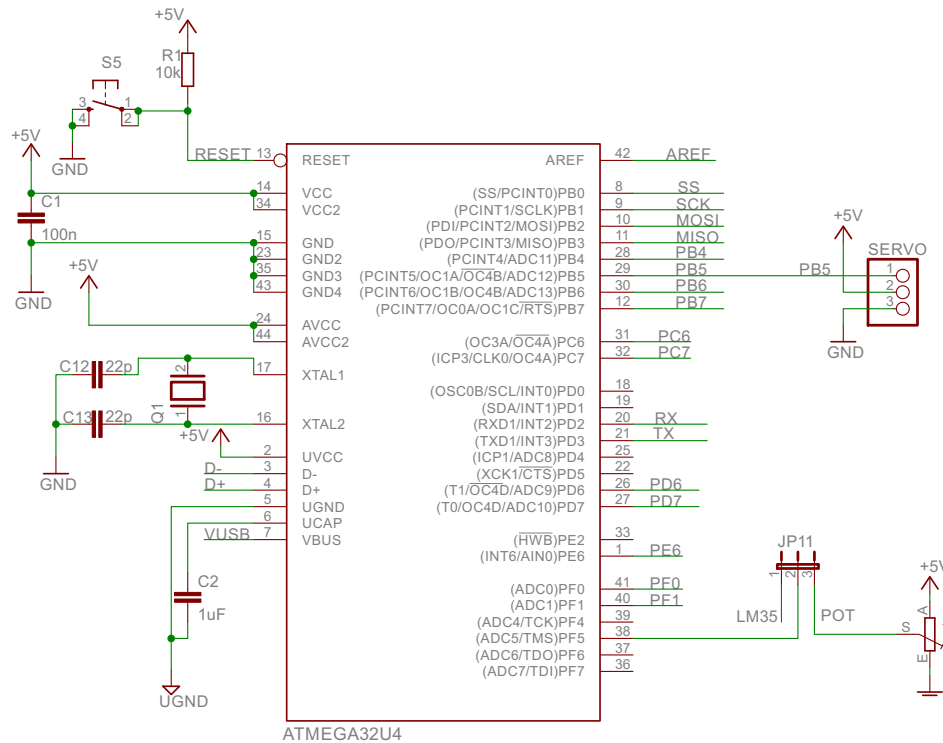
$$duty [\%] = \frac{T_D}{20\text{ms}} = duty_{min} [\%] + \frac{duty_{max} [\%] - duty_{min} [\%]}{180^\circ} \alpha [^\circ] \quad (12.15)$$

gdje su:

- $duty_{min}$  - širina impulsa PWM signala za zakret izlazne osovine iznosa  $0^\circ$ ,

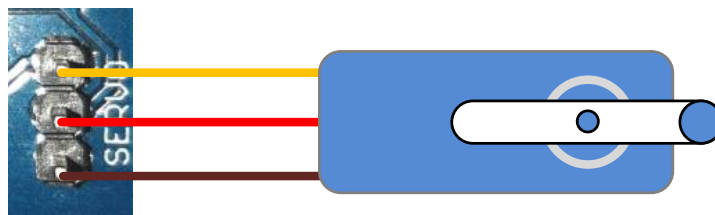
- $duty_{max}$  - širina impulsa PWM signala za zakret izlazne osovine iznosi  $180^\circ$ .

Shema spajanja servomotora i potencijometra na mikroupravljač ATmega32U4 prikazana je na slici 12.12. Signalna linija servomotora (narančasta žica) spojena je na pin PB5 (OC1A) jer se na tom pinu može generirati PWM signal frekvencije 50 Hz.



Slika 12.12: Shema spajanja servomotora i potencijometra na mikroupravljač ATmega32U4

Potencijometar je spojen na pin ADC5 preko kratkospojnika JP11 na način da je kratkospojnik postavljen između trnova 2 i 3. Spajanje servomotora na razvojno okruženje sa slike 3.1 prikazano je na slici 12.13. Potrebno je voditi računa da se konektor servomotora ispravno okrene, kako je prikazano na slici (vodite se bojama žica).



Slika 12.13: Spajanje servomotora na razvojno okruženje sa slike 3.1

### Vježba 12.5.1

Napravite program kojim ćete:

- pomoću potencijometra upravljati zakretom izlazne osovine servomotora u rasponu od  $0^\circ$  do  $180^\circ$  te na temelju tih zakreta pronaći minimalnu i maksimalnu širinu impulsa PWM signala za zadane zakrete izlazne osovine,

- napraviti funkciju koja će primati zakret izlazne osovine servomotora u rasponu od 0° do 180° te određivati širinu impulsa PWM signala za zadane zakrete,
- napraviti proizvoljno kontinuirano zakretanje izlazne osovine servomotora.

Schema spajanja servomotora i potencijometra na mikroupravljač ATmega32U4 prikazana je na slici 12.12.

U projektnom stablu otvorite datoteku vjezba1251.cpp. Omogućite samo prevođenje datoteke vjezba1251.cpp. Početni sadržaj datoteke vjezba1251.cpp prikazan je programskim kodom 12.8.

Programski kod 12.8: Početni sadržaj datoteke vjezba1251.cpp

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "ADC/adc.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"

#define DUTY_MIN 2.0 // širina PWM signala za kut 0
#define DUTY_MAX 12.5 // širina PWM signala za kut 180

void inicijalizacija(){
    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
    // timer 1 kao PWM 50 Hz
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_set_phase_correct_PWM_ICR1(2500);
    timer1_OC1A_enable_non_inverted_PWM();
    DDRB |= (1 << PB5); // izlazni pin za servomotor (OC1A)
}

int main(void) {

    inicijalizacija();

    uint16_t ADC_5;
    float duty, angle;

    while(1) {

        ADC_5 = adc_read(ADC5); // AD pretvorba na pinu ADC5
        // D = [2.0, 12.5] (0.4 ms - 2.5 ms)
        duty = DUTY_MIN + (DUTY_MAX - DUTY_MIN) * ADC_5 / 1023.0;
        // postavi duty cycle na kanalu OC1A
        OC1A_set_duty_cycle(duty, PWM_ICR1);
        lcd_clrscr();
        lcd_home();
        lcd_print("duty = %0.3f\n", duty);
        _delay_ms(500);
    }
    return 0;
}
```

Prvi i osnovni cilj u ovoj vježbi jest osigurati generiranje PWM signala frekvencije 50 Hz. Signalna linija servomotora spojena je na pin PB5 (OC1A) koji kao alternativnu primjenu ima generiranje PWM signala pomoću sklopa *Timer/Counter1*. Sklop *Timer/Counter1* može generirati PWM signal željene frekvencije i u *Fast PWM* načinu rada i u *Phase Correct* načinu

rada. Vršna vrijednost do koje registar `TCNT1` broji u oba PWM načina rada podešava se registrom `ICR1`. U ovoj vježbi koristit ćemo *Phase Correct* način rada. Frekvencija PWM signala na pinu OC1A za *Phase Correct* način rada može se izračunati prema sljedećoj relaciji:

$$F_{\_pcPWM} = \frac{F\_CPU}{PRESCALER \cdot (2 \cdot PWM\_ICR1)} = \frac{F\_CPU}{PRESCALER \cdot (2 \cdot ICR1)}. \quad (12.16)$$

Vrijednost registra `ICR1` za zadanu frekvenciju može se iz relacije (12.16) izračunati na sljedeći način:

$$ICR1 = \frac{F\_CPU}{PRESCALER \cdot 2 \cdot F_{\_pcPWM}}. \quad (12.17)$$

Za pozicioniranje servomotora potrebno je generirati PWM signal frekvencije 50 Hz, pa će vrijednost registra `ICR1` uz djelitelj frekvencije radnog takta iznosa 64 biti:

$$ICR1 = \frac{F\_CPU}{PRESCALER \cdot 2 \cdot F_{\_pcPWM}} = \frac{16000000}{64 \cdot 2 \cdot 50} = 2500. \quad (12.18)$$

Postavljanje sklopa *Timer/Counter1* u *Phase Correct* način rada te postavljanje vrijednosti registra `ICR1` na iznos 2500 provodi se funkcijom `timer1_set_phase_correct_PWM_ICR1(2500)`. U programskom kodu 12.8 u funkciji `inicijalizacija()` konfigurirani su analogno-digitalna pretvorba, LCD displej, sklop *Timer/Counter1* u *Phase Correct* način rada te je pin PB5 postavljen kao izlaz.

U `while` petlji postavlja se širina PWM signala za upravljanje pozicijom servomotora prema relaciji (12.15), uz izmjenu da je omjer  $\alpha/180$  zamijenjen omjerom `ADC_5/1023.0`. Konstante `DUTY_MIN` i `DUTY_MAX` koriste se za određivanje širine PWM signala kod zakreta izlazne osovine iznosa  $0^\circ$  i  $180^\circ$ . Trenutne vrijednosti konstanti jesu:

- `DUTY_MIN` iznosi 2,0, što odgovara širini impulsa od 0,4 ms,
- `DUTY_MAX` iznosi 12,5, što odgovara širini impulsa od 2,5 ms.

Prvi zadatak koji moramo napraviti jest utvrditi koja je minimalna (za  $0^\circ$ ), a koja maksimalna (za  $180^\circ$ ) širina impulsa PWM signala. Prevedite datoteku `vjezba1251.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Pomoću potenciometra je potrebno zakrenuti izlaznu osovinu servomotora na  $0^\circ$ . Na LCD displeju pročitajte vrijednost širine impulsa PWM signala koju je potrebno dodijeliti konstanti `DUTY_MIN`. Za servomotor koji je korišten pri testiranju ove vježbe, konstanta `DUTY_MIN` iznosi 2,38. Vrijednost 2,38 potrebno je dodijeliti konstanti `DUTY_MIN` u programskom kodu 12.8. Nakon toga je pomoću potenciometra potrebno zakrenuti izlaznu osovinu servomotora na  $180^\circ$ . Na LCD displeju pročitajte vrijednost širine impulsa PWM signala koju je potrebno dodijeliti konstanti `DUTY_MAX`. Za servomotor koji je korišten pri testiranju ove vježbe konstanta `DUTY_MAX` iznosi 10,96. Vrijednost 10,96 potrebno je dodijeliti konstanti `DUTY_MAX` u programskom kodu 12.8. Bitno je razumjeti da će ove konstante biti različite za različite proizvođače servomotora.

Širina impulsa za konstantu `DUTY_MIN` iznosa 2,38 jest 0,476 ms, a širina impulsa za konstantu `DUTY_MAX` iznosa 10,96 jest 2,192 ms. Možemo primijetiti da se ove širine impulsa razlikuju od širina definiranih slikom 12.11. Prevedite datoteku `vjezba1251.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Primijetite da sada puni opseg zakreta potenciometra odgovara opsegu zakreta izlazne osovine servomotora od  $0^\circ$  do  $180^\circ$ .

Sljedeći korak jest napraviti funkciju koja će primati zakret izlazne osovine servomotora u rasponu od  $0^\circ$  do  $180^\circ$  te postaviti širinu impulsa PWM signala za zadane zakrete. Funkcija je prikazana programskim kodom 12.9.

Programski kod 12.9: Funkcija za zakretanje izlazne osovine servomotora za željeni kut

```
void set_servo_angle(float angle) {
    float duty = DUTY_MIN + (DUTY_MAX - DUTY_MIN) / 180 * angle;
    OC1A_set_duty_cycle(duty, PWM_ICR1);
}
```

Dodajte programski kod 12.9 u datoteku vjezba1251.cpp. Funkcija `set_servo_angle(float angle)` izračunava i postavlja širinu PWM signala prema relaciji (12.15). U datoteci `vjezba1251.cpp` zamijenite `while` petlju programskim kodom 12.10. Na LCD displeju sada ispisujemo kut izlazne osovine, a potencijetrom zadajemo referentnu vrijednost kuta i šaljemo ga u funkciju `set_servo_angle(float angle)`.

Programski kod 12.10: Novi sadržaj `while` petlje u datoteci `vjezba1251.cpp`

```
while(1) {

    ADC_5 = adc_read(ADC5); // AD pretvorba na pinu ADC5
    angle = ADC_5 / 1023.0 * 180;
    set_servo_angle(angle);
    lcd_clrscr();
    lcd_home();
    lcd_print("kut = %0.2f%c\n", angle, 223);
    _delay_ms(500);
}
```

Prevedite datoteku `vjezba1251.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Potencijetrom zadajte referentnu vrijednost kuta iznosa  $90^\circ$ . Iako bi se zakret trebao linearno mijenjati promjenom širine impulsa, to ponekad nije tako. Primijetite da izlazna osovina nije postigla kut od  $90^\circ$ . Jedno od mogućih rješenja za smanjivanje odstupanja jest uvođenje nove konstante `DUTY_MID` koja će imati vrijednost širine PWM impulsa za zakret izlazne osovine iznosa  $90^\circ$ . Ovaj postupak zahtijeva vraćanje korak unazad kako biste na LCD displeju pročitali kolika je širina impulsa PWM signala za zakret izlazne osovine iznosa  $90^\circ$ . Za servomotor koji je korišten pri testiranju ove vježbe konstanta `DUTY_MID` iznosi 6,037. Napravite konstantu `DUTY_MID` u datoteci `vjezba1251.cpp` te joj dodijelite vrijednost 6,037.

Sada se može napraviti nova funkcija za zakretanje izlazne osovine servomotora (programski kod 12.11). Promjena kuta sada je podijeljena u dva raspona:  $0^\circ$  do  $90^\circ$  i  $90^\circ$  do  $180^\circ$ . Dodajte programski kod 12.9 u datoteku `vjezba1251.cpp`. Zamijenite poziv funkcije `set_servo_angle(float angle)` pozivom funkcije `set_servo_angle_2(float angle)`.

Programski kod 12.11: Novi sadržaj datoteke `vjezba1251.cpp`

```
void set_servo_angle_2(float angle) {
    float duty;
    if(angle <= 90) {
        duty = DUTY_MIN + (DUTY_MID - DUTY_MIN) / 90 * angle;
    }
    else {
        duty = DUTY_MID + (DUTY_MAX - DUTY_MID) / 90 * (angle - 90);
    }
    OC1A_set_duty_cycle(duty, PWM_ICR1);
}
```

Prevedite datoteku `vjezba1251.cpp` u strojni kod i snimite ga na mikroupravljač ATmega32U4. Potencijetrom zadajte referentnu vrijednost kuta iznosa  $90^\circ$ . Sada bi izlazna osovina trebala postići zakret iznosa  $90^\circ$ .

Konačno, sada možemo napraviti proizvoljno kontinuirano zakretanje izlazne osovine servomotora. U datoteku `vjezba1251.cpp` napišite programski kod 12.12. Ovaj programski kod kontinuirano mijenja zakret osovine na sljedeći način:

- na zakretu izlazne osovine iznosa  $0^\circ$  servomotor se zaustavlja u trajanju od 500 ms i mijenja mu se smjer vrtnje,
- kontinuirano se mijenja zakret izlazne osovine od  $0^\circ$  do  $90^\circ$ ,
- na zakretu izlazne osovine iznosa  $90^\circ$  servomotor se zaustavlja u trajanju od 500 ms,
- kontinuirano se mijenja zakret izlazne osovine od  $90^\circ$  do  $180^\circ$ ,
- na zakretu izlazne osovine iznosa  $180^\circ$  servomotor se zaustavlja u trajanju od 500 ms i mijenja mu se smjer vrtnje,
- svi se koraci neprestano ponavljaju.

Programski kod 12.12: Novi sadržaj datoteke `vjezba1251.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "ADC/adc.h"
#include "Timer/timer.h"
#include "LCD/lcd.h"

#define DUTY_MIN 2.38 // širina PWM signala za kut 0
#define DUTY_MAX 10.96 // širina PWM signala za kut 180
#define DUTY_MID 6.037 // širina PWM signala za kut 90

void set_servo_angle_2(float angle) {
    float duty;
    if(angle <= 90) {
        duty = DUTY_MIN + (DUTY_MID - DUTY_MIN) / 90 * angle;
    }
    else {
        duty = DUTY_MID + (DUTY_MAX - DUTY_MID) / 90 * (angle - 90);
    }
    OC1A_set_duty_cycle(duty, PWM_ICR1);
}

void set_servo_angle(float angle) {
    float duty = DUTY_MIN + (DUTY_MAX - DUTY_MIN) / 180 * angle;
    OC1A_set_duty_cycle(duty, PWM_ICR1);
}

void inicijalizacija(){
    lcd_init(); // inicijalizacija LCD displeja
    adc_init(); // inicijalizacija AD pretvorbe
    // timer 1 kao PWM 50 Hz
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_set_phase_correct_PWM_ICR1(2500);
    timer1_OC1A_enable_non_inverted_PWM();
    DDRB |= (1 << PB5); // izlazni pin za servomotor (OC1A)
}

int main(void) {

    inicijalizacija();

    uint8_t i;
```

```

// direction = true - gibanje od 0 do 180
// direction = false - gibanje od 180 do 0
bool direction = true;

while(1) {
    // postavi zakret na servomotor
    set_servo_angle_2(i);
    // ako je zakret 0, pričekaj 500 ms i promijeni smjer
    if(i == 0) {
        direction = true;
        _delay_ms(500);
    }
    // ako je zakret 90, pričekaj 500 ms
    if(i == 90) _delay_ms(500);
    // ako je zakret 180, pričekaj 500 ms i promijeni smjer
    if(i == 180) {
        direction = false;
        _delay_ms(500);
    }
    // ako je direction = true povećavaj zakret, inače smanjuj
    if(direction) {
        i++;
    }
    else {
        i--;
    }
    _delay_ms(10);
}
return 0;
}

```

Prevedite datoteku vjezba1251.cpp u strojni kod i snimite ga na mikroupravljač ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4. Napravite drugačiji obrazac ponašanja zakreta izlazne osovine servomotora i testirajte ga na razvojnom okruženju s mikroupravljačem ATmega32U4.

Zatvorite datoteku vjezba1251.cpp i onemogućite prevođenje ove datoteke.

### 12.5.1 Zadatak - servomotor

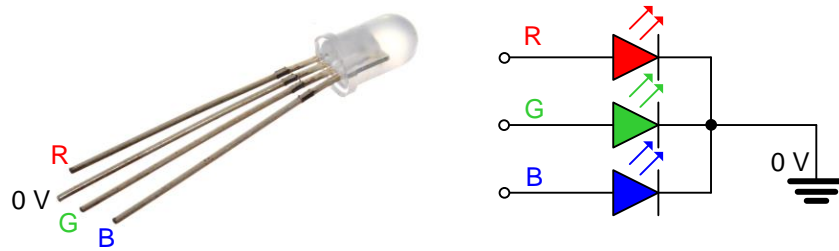
#### Zadatak 12.5.1

Napravite program kojim ćete zakretom izlazne osovine servomotora upravljati aplikacijom sa slike 10.4 pomoću klizača **Slider**. Shema spajanja servomotora i potenciometra na mikroupravljač ATmega32U4 prikazana je na slici 12.12.

## 12.6 RGB dioda

RGB dioda sastoji se od triju LED dioda u jednom kućištu (slika 12.14). Naziv RGB dolazi od crvene (R), zelene (G) i plave (B) LED diode koje se nalaze u zajedničkom kućištu. Miješanjem ovih triju boja u različitim intenzitetima mogu se dobiti sve moguće boje. Intenzitet pojedine LED diode može se mijenjati PWM signalom na svakoj od anoda RGB diode. Ako sve diode svijetle punim intenzitetom, svjetlost koju će davati RGB dioda bit će bijela. RGB dioda sa slike 12.14 jest dioda sa zajedničkom katodom.





Slika 12.14: RGB dioda

Shema spajanja RGB diode na mikroupravljač ATmega32U4 prikazana je na slici 4.1. Kratkospojnik JP4 potrebno je postaviti između trnova 2 i 3.



### Vježba 12.6.1

Napravite program kojim ćete pomoću aplikacije sa slike 10.4 mijenjati intenzitet RGB diode pomoću klizača R, G i B u okviru **Sliders**. Pritiskom na tipku **Send** RGB aplikacija će pomoću serijskog porta poslati poruku formata "**R%u;%u;%u\*\r**", pri čemu kvalifikatori "**%u**" poprimaju vrijednosti u rasponu [0, 100], a predstavljaju širinu impulsa PWM signala koji upravljaju intenzitetom R, G i B LED diode. Crvena (R) LED dioda spojena je na OC1A pin, zelena (G) LED dioda spojena je na OC1B pin, a plava (B) LED dioda spojena je na OC1C pin. Ako je na mikroupravljač pristigla poruka "**R50;0;90\*\r**", tada će širina impulsa PWM signala na pinu OC1A biti 50 %, na pinu OC1B 0 %, a na pinu OC1C 90 %.

U projektnom stablu otvorite datoteku `vjezba1261.cpp`. Omogućite samo prevođenje datoteke `vjezba1261.cpp`. Početni sadržaj datoteke `vjezba1261.cpp` prikazan je programskim kodom 12.13.

Programski kod 12.13: Početni sadržaj datoteke `vjezba1261.cpp`

```
#include <avr/io.h>
#include "AVR_VUB/avrvub.h"
#include <util/delay.h>
#include "LCD/lcd.h"
#include "uart/uart.h"
#include "Interrupt/interrupt.h"
#include "Timer/timer.h"

void inicijalizacija() {

    lcd_init(); // konfiguracija LCD displeja
    uart_init(19200); // UART konfiguracija
    // tajmer 1 u Phase Correct načinu rada
    timer1_set_prescaler(TIMER1_PRESCALER_64);
    timer1_set_phase_correct_PWM_8bit();
    // neinvertirajući PWM na kanalima OC1A, OC1B i OC1C
    timer1_OC1A_enable_non_inverted_PWM();
    timer1_OC1B_enable_non_inverted_PWM();
    timer1_OC1C_enable_non_inverted_PWM();
    // izlazni pinovi PB5 (R), PB6 (G), PB7 (B)
    DDRB |= (1 << PB5) | (1 << PB6) | (1 << PB7);
    interrupt_enable();
}

int main(void) {
```

```

inicijalizacija();
uint8_t RGB[3] = {0, 0, 0};

while(1) {

    if(uart_read_all()) {
        lcd_clrscr();
        lcd_home();
        lcd_print("%s\n", uart_buffer);

        if(uart_buffer[0] == 'R') {
            RGB[0] = 0;
            RGB[1] = 0;
            RGB[2] = 0;
            // čitanje poruke tipa R26;14;88*
            for(uint8_t i = 1, j = 0; uart_buffer[i] != '*'; i++) {
                if (uart_buffer[i] != ';') {
                    RGB[j] = RGB[j] * 10 + (uart_buffer[i] - 48);
                }
                else {
                    j++;
                }
            }
            //postavite širine PWM impulsa na pinovima OC1A, OC1B i OC1C
            OC1A_set_duty_cycle(RGB[0], PWM_8BIT);
            OC1B_set_duty_cycle(RGB[1], PWM_8BIT);
            OC1C_set_duty_cycle(RGB[2], PWM_8BIT);
            // ispis pojedinih širina impulsa
            lcd_print("%u %u %u *", RGB[0], RGB[1], RGB[2]);
        }
    }
}

```

U programskom kodu 12.13 u funkciji `inicijalizacija()` konfigurirani su LCD displej i serijska komunikacija. S obzirom na to da je crvena (R) LED dioda spojena na OC1A pin, zelena (G) LED dioda na OC1B pin, a plava (B) LED dioda na OC1C pin, na ovim je pinovima potrebno generirati PWM signal. Koristiti ćemo sklop *Timer/Counter1* u *Phase Correct* načinu rada s djelitelem frekvencije radnoga takta iznosa 64. Na pinovima OC1A, OC1B i OC1C omogućeni su neinvertirani PWM signali. Također, pinovi OC1A (PB5), OC1B (PB6) i OC1C (PB7) postavljeni su kao izlazni pinovi.

U `main()` funkciji deklarirano je polje podataka `RGB` s trima elementa. Svaki element polja predstavljaće širinu impulsa PWM signala u rasponu [0, 100]. U `while` petlji čeka se pristigla poruka u međuspremnik `uart_buffer`. Kada poruka dođe, iz nje je potrebno izvući vrijednosti intenziteta pojedine LED diode. Prvi znak u poruci jest `'R'`. Sljedeći dio poruke do znaka `','` jest širina impulsa PWM signala za crvenu LED diodu. Kroz pristiglu poruku prolazimo pomoću `for` petlje koja pretvara niz znakova koji predstavljaju širinu impulsa PWM signala u cijeli broj. Kada `for` petlja naiđe na znak `','`, tada znamo da nakon ovog znaka stiže sljedeća širina impulsa PWM signala za zelenu LED diodu. Na isti način dolazimo i do širine PWM signala za plavu LED diodu. Kada se odrede sve širine impulsa PWM signala za pinove OC1A, OC1B i OC1C, funkcijama `OC1A_set_duty_cycle(RGB[0], PWM_8BIT)`, `OC1B_set_duty_cycle(RGB[1], PWM_8BIT)`, i `OC1C_set_duty_cycle(RGB[2], PWM_8BIT)` postavljamo širinu impulsa PWM signala na pinove OC1A, OC1B i OC1C.

Prevedite datoteku `vjezba1261.cpp` u strojni kod i snimite ga na mikrokontroler ATmega32U4. Testirajte program na razvojnom okruženju s mikroupravljačem ATmega32U4 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port na kojem se nalazi CP2102 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na

tipku **Start COM**. Namjestite klizače **R**, **G** i **B** u okviru **Sliders** na željene vrijednosti i pritisnite tipku **Send RGB**.

Zatvorite datoteku `vjezba1261.cpp` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 7*.

### 12.6.1 Zadatak - RGB dioda

#### Zadatak 12.6.1

Napravite program kojim ćete pomoću aplikacije sa slike 10.4 mijenjati intenzitet RGB diode pomoću poruke formata `"RGB%u*%u*%u:\r"`, pri čemu kvalifikatori `"%u"` poprimaju vrijednosti u rasponu  $[0, 200]$ , a predstavljaju širinu impulsa PWM signala koji upravljaju intenzitetom **R**, **G** i **B** LED diode. Poruka se unosi pomoću tekstualnog okvira **Message** i pritiskom na tipku **Send**. Crvena (**R**) LED dioda spojena je na **OC1A** pin, zelena (**G**) LED dioda spojena je na **OC1B** pin, a plava (**B**) LED dioda spojena je na **OC1C** pin. Ako je na mikroupravljač pristigla poruka `"RGB40*80*180:\r"`, tada će širina impulsa PWM signala na pinu **OC1A** biti 20 %, na pinu **OC1B** 40 %, a na pinu **OC1C** 90 %. Za generiranje PWM signala koristite sklop *Timer/Counter1* u *Fast* načinu rada.

---



# Bibliografija

- [1] Atmel, *ATmega16U4/ATmega32U4: 8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller*. Microchip, <https://www.microchip.com/wwwproducts/en/ATmega32u4>, 2016.
- [2] XIAMAN OCULAR, [www.elmicro.com/files/lcd/gdm1602a\\_datasheet.pdf](http://www.elmicro.com/files/lcd/gdm1602a_datasheet.pdf), *Specification of LCD Module GDM1602A*, 2005.
- [3] Z. Vrhovski, *Predavanja iz kolegija MIKRORAČUNALA*. Veleučilište u Bjelovaru, Bjelovar, 2020.
- [4] G. Gridling and B. Weiss, *Introduction to Microcontrollers*. Vienna University of Technology, Institute of Computer Engineering, Vienna, 2007.
- [5] ELEC Freaks, [http://www.electfreaks.com/store/download/product/Sensor/HC-SR04/HC-SR04\\_Ultrasonic\\_Module\\_User\\_Guide.pdf](http://www.electfreaks.com/store/download/product/Sensor/HC-SR04/HC-SR04_Ultrasonic_Module_User_Guide.pdf), *HC-SR04 User Guide*, 2013.
- [6] TEXAS INSTRUMENTS, <http://www.ti.com/lit/ds/symlink/lm35.pdf>, *LM35 Precision Centigrade Temperature Sensors*, 2013.
- [7] How to Mechatronics, <https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/>, *How Servo Motor Works & How To Control Servos using Arduino*, 2018.





[www.vub.hr](http://www.vub.hr)

