

# Mikroračunala : programiranje mikrokontrolera porodice Atmel u programskom okruženju Atmel studio 6

---

**Vrhovski, Zoran; Miletić, Marko**

**Authored book / Autorska knjiga**

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

*Publication year / Godina izdavanja:* **2014**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:144:612885>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-04-02**



*Repository / Repozitorij:*

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

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija(){
    lcd_init();
    adc_init();
}

int main(void){

    inicijalizacija();

    uint16_t ADC5;
    float VPA5;
    const float VREF = 5.0;

    while (1)
    {
        lcd_clrscr();
        lcd_home();
        ADC5 = adc_read_10bit(5);
        VPA5 = ADC5 * VREF / 1024.0;
    }
}
```

## MIKRORAČUNALA

Programiranje mikrokontrolera porodice Atmel  
u programskom okruženju Atmel Studio 6



Bjelovar, 2014.

VISOKA TEHNIČKA ŠKOLA U BJELOVARU

## MIKRORAČUNALA

Programiranje mikrokontrolera porodice Atmel u programskom  
okruženju Atmel Studio 6

Prvo izdanje



Bjelovar, 2014.

Zoran Vrhovski, mag. ing. el. techn. inf.  
Marko Miletić, bacc. ing. mech.

MIKRORAČUNALA  
Programiranje mikrokontrolera porodice Atmel u programskom okruženju  
Atmel Studio 6

Nakladnik:  
Visoka tehnička škola u Bjelovaru

Recenzenti:  
Mr. sc. Ivan Šumiga  
Dalibor Purković, mag. ing. el. techn. inf.

Lektorica:  
Ivana Jurković, prof.

Crteži:  
Zoran Vrhovski, mag. ing. el. techn. inf.  
Marko Miletić, bacc. ing. mech.

Priprema za tisak:  
Zoran Vrhovski, mag. ing. el. techn. inf.

Dizajn ovitka:  
Josip Horvat

Tisak:  
Croatiagraf d.o.o. Markovac Križevački, [www.croatiagraf.hr](http://www.croatiagraf.hr)  
Rujan, 2014.

CIP zapis dostupan u računalnome katalogu Nacionalne i sveučilišne knjižnice u  
Zagrebu pod brojem 884830.  
ISBN 978-953-7676-17-9

Napomena:  
Niti jedan dio knjige ne smije se preslikavati niti umnožavati  
bez prethodne suglasnosti autora.

Zoran Vrhovski, mag. ing. el. techn. inf.  
Marko Miletić, bacc. ing. mech.

# MIKRORAČUNALA

**Programiranje mikrokontrolera porodice Atmel u programskom  
okruženju Atmel Studio 6**

Prvo izdanje



Bjelovar, 2014.



# Predgovor

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

Ovaj je udžbenik iz Mikroracunala namijenjen prvenstveno studentima Stručnog studija mehatronike na Visokoj tehničkoj školi u Bjelovaru koji slušaju kolegij *Mikroracunala*. Udžbenik se može koristiti i na svim drugim veleučilištima i sveučilištima koja se u sklopu svog programa bave mikroracunalima.

Osnovni je cilj ovog udžbenika približiti programiranje mikrokontrolera ATmega16 studentima i ostalima koji će čitati i proučavati ovaj materijal. Znanja stečena pomoću ovog udžbenika mogu se primijeniti na AVR porodicu mikrokontrolera Atmel te na ostale porodice mikrokontrolera. Udžbenik sadrži 174 stranice s 50 riješenih vježbi i 50 zadataka za vježbu. Svaka vježba u ovom udžbeniku izrađena je u programskom okruženju *Atmel Studio 6* i dostupna je na mrežnoj stranici [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Svako poglavlje u uvodnom dijelu sadrži teorijski dio koji je potrebno proučiti za što bolje razumijevanje riješenih vježbi i zadataka za vježbu. Studentima se preporučuje da najprije prouče riješene vježbe, a nakon toga da rješavaju zadatke za vježbu.

Zahvaljujemo se recenzentima, mr. sc. Ivanu Šumigi i Daliboru Purkoviću, mag. ing. el. techn. inf. na korisnim savjetima i sugestijama te lektorici Ivani Jurković, prof. na strpljenju u čitanju ove knjige i usklađivanju teksta s hrvatskim standardnim jezikom.

Posebno se zahvaljujemo Visokoj tehničkoj školi u Bjelovaru na financijskoj potpori bez koje izdavanje ove knjige ne bi bilo moguće.

Zoran Vrhovski, mag. ing. el. techn. inf.  
Marko Miletić, bacc. ing. mech.





# Sadržaj

<b>1</b>	<b>Uvod</b>	<b>1</b>
<b>2</b>	<b>Razvojno okruženje <i>Atmel Studio 6</i></b>	<b>3</b>
2.1	Odabir razvojnog okruženja . . . . .	3
2.2	Stvaranje projekta u razvojnem okruženju <i>Atmel Studio 6</i> . . . . .	4
<b>3</b>	<b>Snimanje i pokretanje strojnog koda na mikrokontroleru ATmega16</b>	<b>11</b>
3.1	Razvojno okruženje ATmega16 . . . . .	11
3.2	Programiranje mikrokontrolera ATmega16 . . . . .	14
<b>4</b>	<b>Digitalni izlazi i ulazi</b>	<b>21</b>
4.1	Digitalni izlazi mikrokontrolera ATmega16 . . . . .	22
4.1.1	Vježbe - digitalni izlazi mikrokontrolera ATmega16 . . . . .	22
4.1.2	Zadaci - digitalni izlazi mikrokontrolera ATmega16 . . . . .	36
4.2	Digitalni ulazi mikrokontrolera ATmega16 . . . . .	38
4.2.1	Vježbe - digitalni ulazi mikrokontrolera ATmega16 . . . . .	39
4.2.2	Zadaci - digitalni ulazi mikrokontrolera ATmega16 . . . . .	49
<b>5</b>	<b>LCD displej</b>	<b>51</b>
5.1	Vježbe - LCD displej . . . . .	51
5.2	Zadaci - LCD displej . . . . .	67
<b>6</b>	<b>EEPROM memorija</b>	<b>69</b>
6.1	Vježbe - EEPROM memorija . . . . .	69
6.2	Zadaci - EEPROM memorija . . . . .	72
<b>7</b>	<b>Analogno-digitalna pretvorba</b>	<b>73</b>
7.1	Vježbe - analogno-digitalna pretvorba . . . . .	73
7.2	Zadaci - analogno-digitalna pretvorba . . . . .	87
<b>8</b>	<b>Tajmeri i brojači</b>	<b>89</b>
8.1	Vježbe - tajmeri i brojači . . . . .	90
8.1.1	Normalan način rada tajmera . . . . .	92
8.1.2	<i>Fast PWM</i> način rada tajmera . . . . .	94

---

8.2	Zadaci - tajmeri i brojači . . . . .	116
<b>9</b>	<b>Numerički displej</b>	<b>119</b>
9.1	Vježbe - numerički displej . . . . .	121
9.2	Zadaci - numerički displej . . . . .	133
<b>10</b>	<b>Univerzalna asinkrona serijska komunikacija</b>	<b>135</b>
10.1	Vježbe - univerzalna asinkrona serijska komunikacija . . . . .	137
10.2	Zadaci - univerzalna asinkrona serijska komunikacija . . . . .	151
<b>11</b>	<b>Vanjski prekidi</b>	<b>153</b>
11.1	Vježbe - vanjski prekidi . . . . .	153
11.2	Zadaci - vanjski prekidi . . . . .	158
<b>12</b>	<b>Odabrani senzori i aktuatori</b>	<b>159</b>
12.1	Vježbe - odabrani senzori i aktuatori . . . . .	159
12.1.1	Tranzistor kao sklopka i relej . . . . .	159
12.1.2	Ultrazvučni senzor HC-SR04 . . . . .	161
12.1.3	Temperaturni senzor LM35 . . . . .	163
12.1.4	Temperaturni senzor DS18B20 . . . . .	163
12.2	Zadaci - odabrani senzori i aktuatori . . . . .	172
	<b>Bibliografija</b>	<b>173</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.

Mikrokontroleri 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 mikrokontrolera koji služi za obradu signala, upravljanje sustavima, prikupljanje informacija iz sustava i slično. Mikrokontroleri 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 mikrokontrolera koji su sastavni dio sustava sigurnosti, putnog računala, navigacijskog sustava, ozvučenja, sustava zračnih jastuka i drugih.

Ovaj udžbenik opisuje razvoj programa za mikrokontroler ATmega16 u programskom okruženju *Atmel Studio 6*. Programi su pisani u sintaksi programskog jezika C. Vježbe koje će biti opisane u ovom udžbeniku napisane su u programskom okruženju *Atmel Studio 6* i nalaze se na mrežnoj stranici [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Uz vježbe se na mrežnoj stranici nalaze i rješenja vježbi kako biste mogli provjeriti ispravnost vlastitih programskih rješenja.

U ovom uvodnom poglavlju proći ćemo ukratko kroz sva poglavlja ovog udžbenika kako bi zainteresirali čitatelje, a to su na prvom mjestu studenti Stručnog studija mehatronike na Visokoj tehničkoj školi u Bjelovaru.

U drugom poglavlju ukratko su opisana programska okruženja koja se koriste za razvoj programa za mikrokontroler ATmega16. Nadalje, opisano je stvaranje projekata u programskom okruženju *Atmel Studio 6*.

Razvojno okruženje s mikrokontrolerom ATmega16 i programiranje mikrokontrolera ATmega16 pomoću softvera *eXtreme Burner – AVR* opisano je u trećem poglavlju.

Digitalni izlazi i ulazi mikrokontrolera ATmega16 kao veza prema vanjskom okruženju opisani su u četvrtom poglavlju. Mikrokontroler ATmega16 ima 32 digitalna pina 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 mikrokontroler. Prikazana je konfiguracija raznih tipova LCD displeja koji se mogu spojiti na mikrokontroler ATmega16.

EEPROM memorija koja podatke čuva i kad nema napajanja mikrokontrolera opisana je u

šestom poglavlju.

U sedmom poglavlju opisano je korištenje analogno-digitalne pretvorbe na mikrokontroleru ATmega16. Analogni senzori koji se koriste za mjerenje napona, tlaka, vlage, temperature i drugih fizikalnih veličina često se koriste u kombinaciji s mikrokontrolerima 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 dva načina rada tajmera; normalan način rada i *Fast PWM* način rada. Ovo poglavlje predstavlja uvod u prekide i prekidne rutine.

U devetom poglavlju opisan je numerički displej koji se koristi za prikazivanje brojeva realnih i cjelobrojnih vrijednosti. Za prikaz brojeva na numeričkim displejima koriste se tajmeri.

Univerzalna asinkrona serijska komunikacija opisana je u desetom poglavlju. Uz ovo poglavlje dostupna je aplikacija kojom se testira asinkrona serijska komunikacija između računala i mikrokontrolera ATmega16. Ovo poglavlje bitno je zbog upravljanja i nadzora sustava pomoću računala gdje je mikrokontroler 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 odabrani senzori i aktuatori koji se mogu spojiti na mikrokontroler ATmega16. Aktuatori koji su opisani u ovom poglavlju su tranzistor kao sklopka i relej. Senzori koji su opisani u ovom poglavlju su ultrazvučni senzor HC-SR04, temperaturni senzor LM35 i temperaturni senzor DS18B20.

## Poglavlje 2

# Razvojno okruženje *Atmel Studio 6*

### 2.1 Odabir razvojnog okruženja

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

- BASCOM,
- *Code Vision*,
- *AVR Studio 5* i
- *Atmel Studio 6*.

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

Programsko okruženje *Code Vision*, kao i BASCOM, ima podršku za niz elektroničkih uređaja. Prednost naspram programskog okruženja BASCOM je 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 6* kojeg ćemo koristiti u nastavku ovog udžbenika. *Atmel Studio 6* profesionalno je razvojno programsko okruženje za stvaranje aplikacija mikrokontrolera 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 6* ima sve prednosti koje donosi programsko okruženje *Visual Studio*. Razvojno okruženje *Atmel Studio 6* besplatno je za razliku od programskih okruženja BASCOM i *Code Vision*.

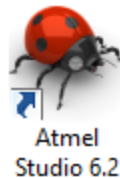
---

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

Od navedenih razvojnih okruženja, zbog svih prednosti, za daljnji razvoj programskih rješenja koristit ćemo *Atmel Studio 6*. Najnoviju verziju razvojnog okruženja *Atmel Studio 6* možete preuzeti na stranici [www.atmel.com/tools/atmelstudio.aspx](http://www.atmel.com/tools/atmelstudio.aspx).

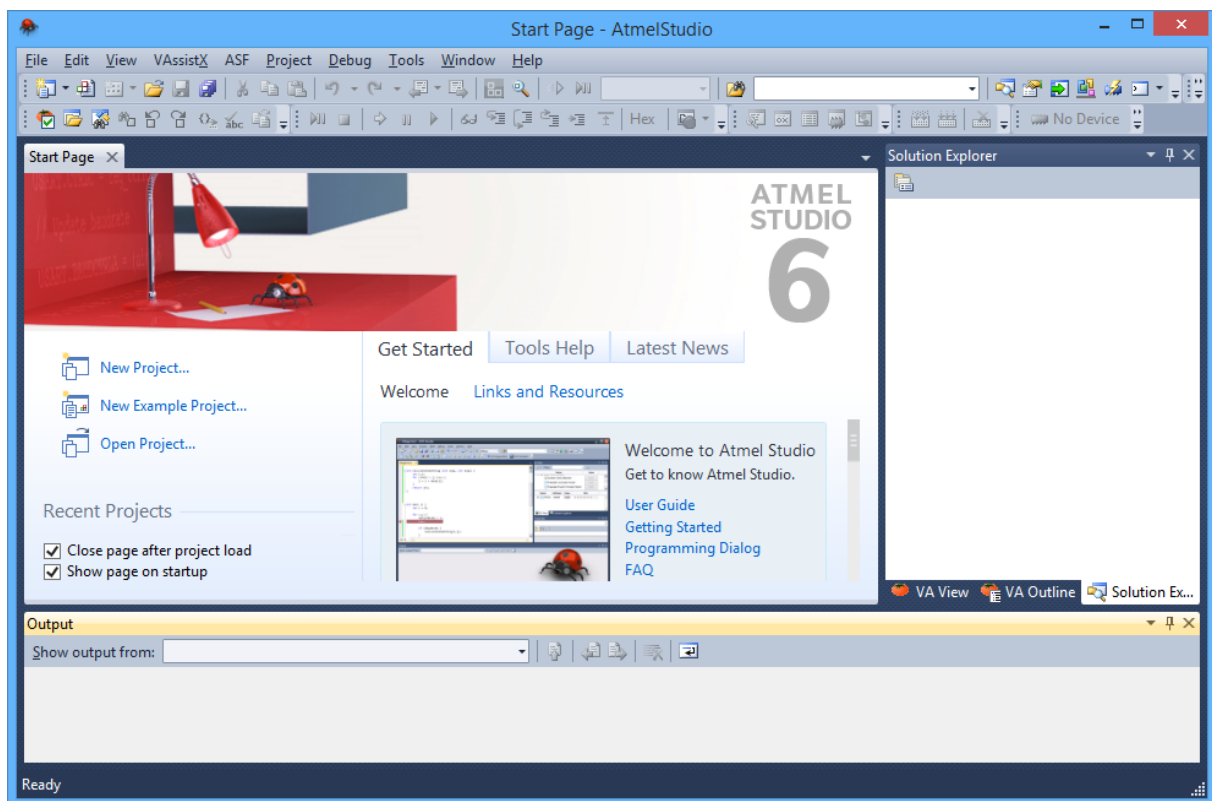
## 2.2 Stvaranje projekta u razvojnom okruženju *Atmel Studio 6*

Razvojno okruženje *Atmel Studio 6* možemo pokrenuti dvostrukim klikom na ikonu sa slike 2.1.



Slika 2.1: *Atmel Studio 6* - ikona

Nakon pokretanja razvojnog okruženja *Atmel Studio 6*, pojavit će se početna stranica prikazana na slici 2.2.

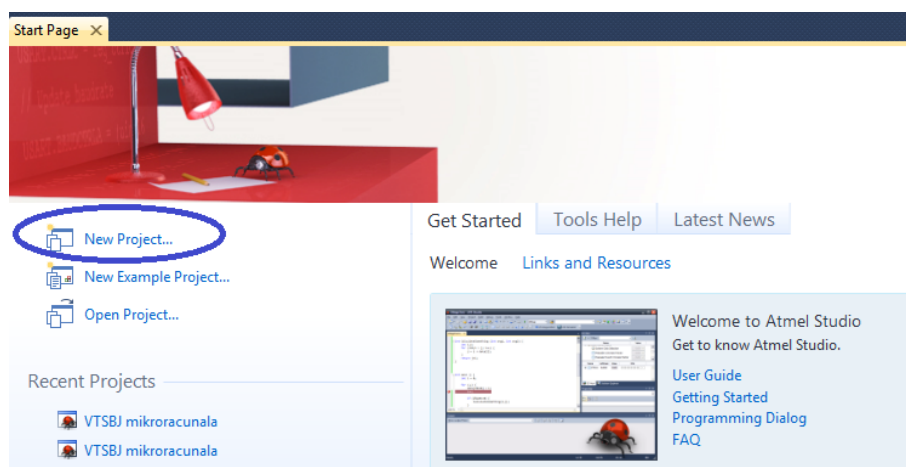


Slika 2.2: Razvojno okruženje *Atmel Studio 6* - početna stranica

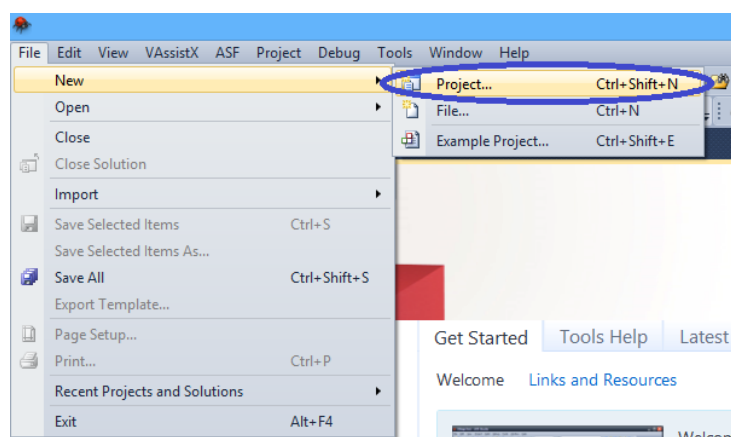
Početna stranica omogućuje nam stvaranje novog projekta, otvaranje postojećeg projekta ili otvaranje nedavno otvorenog projekta. Novi projekt možemo stvoriti na jedan od dva načina:

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 tipa, imena i lokacije projekta.



Slika 2.3: Razvojno okruženje *Atmel Studio 6* - stvaranje novog projekta (1)

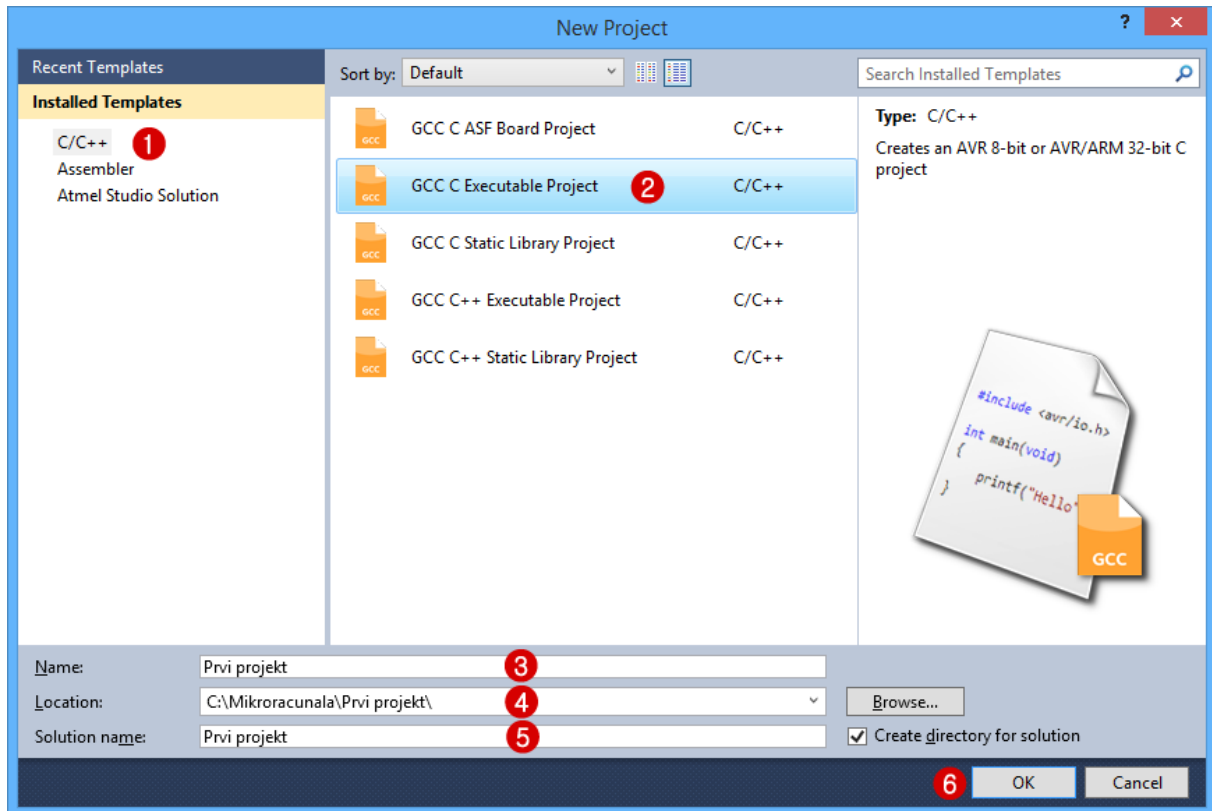


Slika 2.4: Razvojno okruženje *Atmel Studio 6* - stvaranje novog projekta (2)

U prozoru sa slike 2.5 potrebno je definirati parametre označene brojevima od 1 do 6:

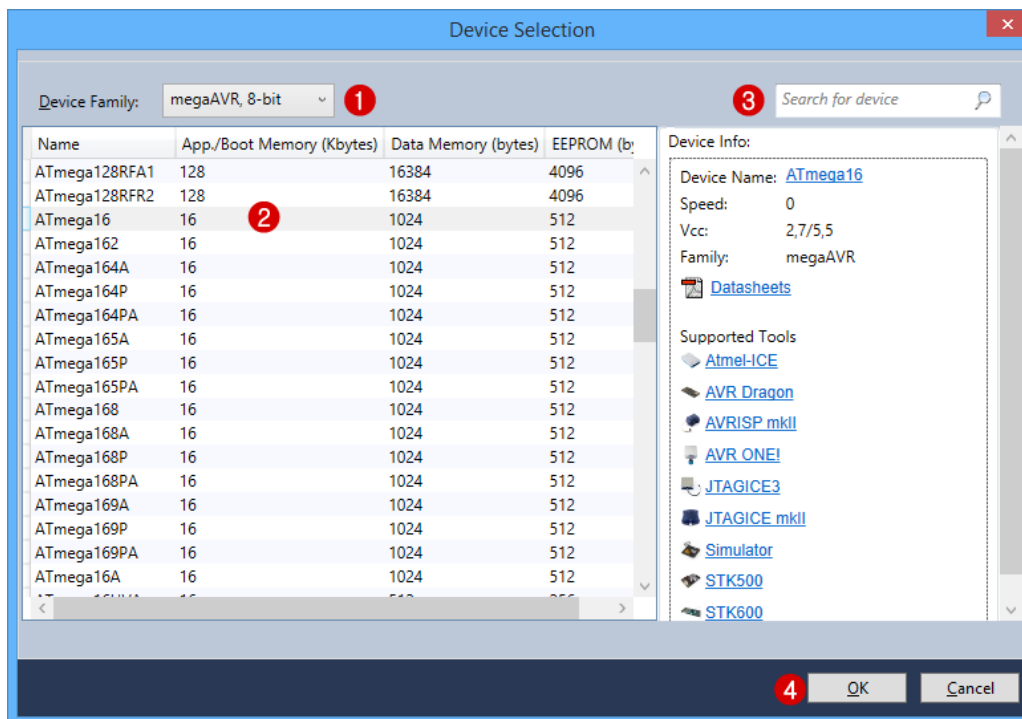
1. U izborniku *Installed Templates* odaberite C/C++.
2. Odaberite *GCC C Executable Project*.
3. Upišite ime projekta (npr. Prvi projekt).
4. Odaberite lokaciju projekta (npr. C:\Mikroracunala\Prvi projekt\).
5. Odaberite ime solucije<sup>2</sup> (npr. Prvi projekt).
6. Pritisnite OK za nastavak.

<sup>2</sup>U razvojnom okruženju *Atmel Studio 6* jedna solucija može imati nekoliko projekata.



Slika 2.5: Razvojno okruženje *Atmel Studio 6* - prozor za unos tipa, imena i lokacije projekta

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



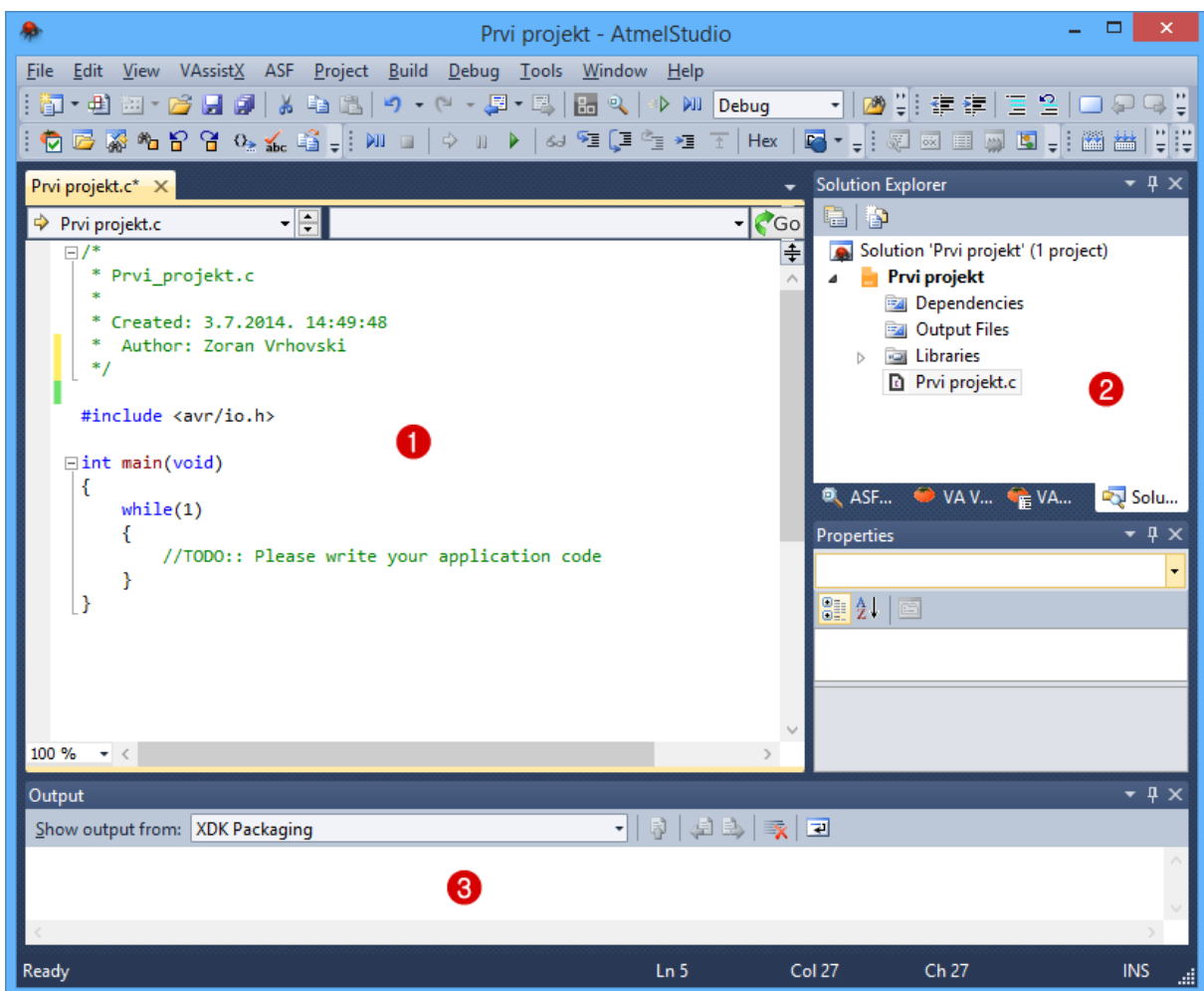
Slika 2.6: Razvojno okruženje *Atmel Studio 6* - odabir mikrokontrolera



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

1. Odaberite porodicu mikrokontrolera (npr. megaAVR, 8-bit).
2. Odaberite mikrokontroler iz odabrane porodice (npr. ATmega16).
3. Korak 1 i 2 mogu se preskočiti ukoliko poznajete točno ime mikrokontrolera kojeg 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 6* - 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. 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.

Prvi projekt stvoren u razvojnom okruženju *Atmel Studio 6* imat će zadaću uključivati LED diode spojene na port B (pinovi 4 do 7) pomoću tipkala spojenih na port B (pinovi 0 do 3). U ovom trenutku nećemo proučavati programski kod, već samo način na koji se koristi razvojno okruženje *Atmel Studio 6* za stvaranje strojnog koda. Programski kod prikazan je u nastavku.

```

/*
 * Prvi_projekt.c
 *
 * Vrijeme kreiranja: 3.7.2014. 14:49:48
 * Autori: Zoran Vrhovski, Marko Miletić
 */
#include <avr/io.h>

int main(void)
{
    // gornja četiri pina na portu B izlazni (LED diode)
    // donja četiri pina na portu B ulazni (tipkala)
    DDRB = 0xF0;

    // isključena gornja četiri pina na portu B
    // uključeni pritezni otpornici na donja četiri pina porta B
    PORTB = 0x0F;

    uint8_t i; // pomoćna varijabla veličine 1B

    while(1)
    {
        i = ~PINB & 0x0F; // čitanje stanja samo ulaznih pinova

        i = i << 4; // stanja pinova posmaknuta za četiri mjesta ulijevo

        PORTB &= 0x0F; // brisanje starog stanja izlaznih pinova

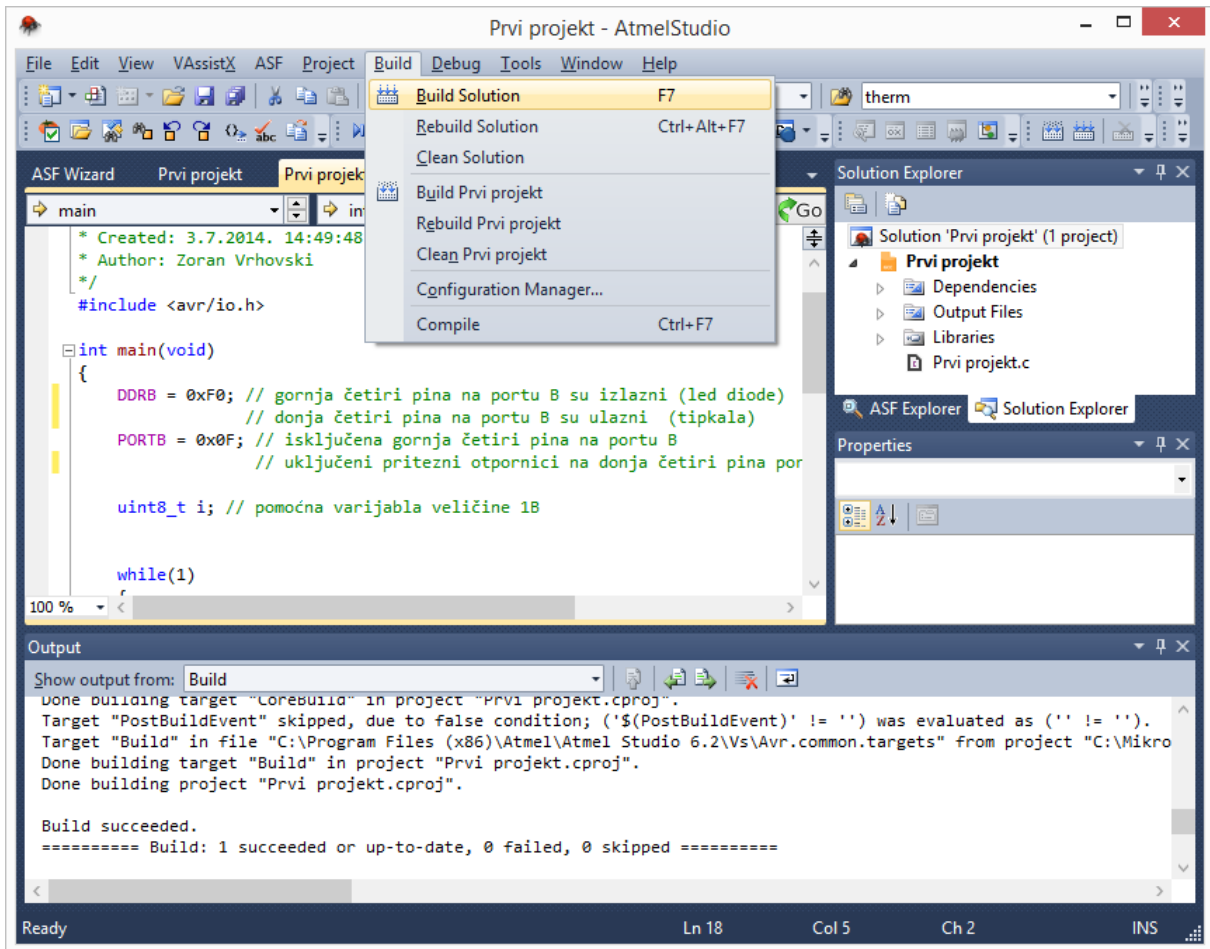
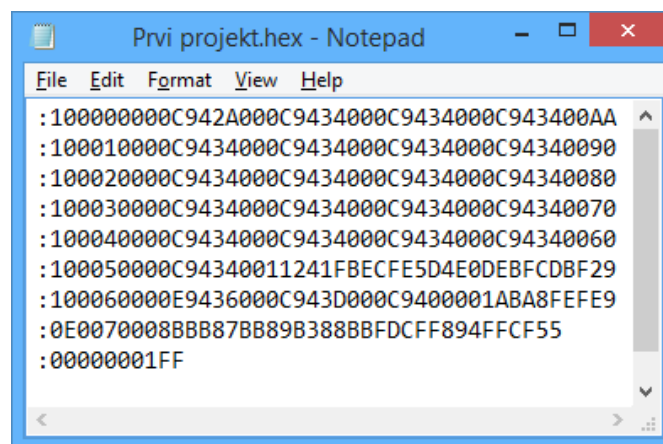
        PORTB |= i & 0xF0; // stanje ulaznih pinova preslikano na izlaz
    }
}

```

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” (eng. *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 6* - 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 mikrokontroler se snima pomoću softvera za programiranje mikrokontrolera.

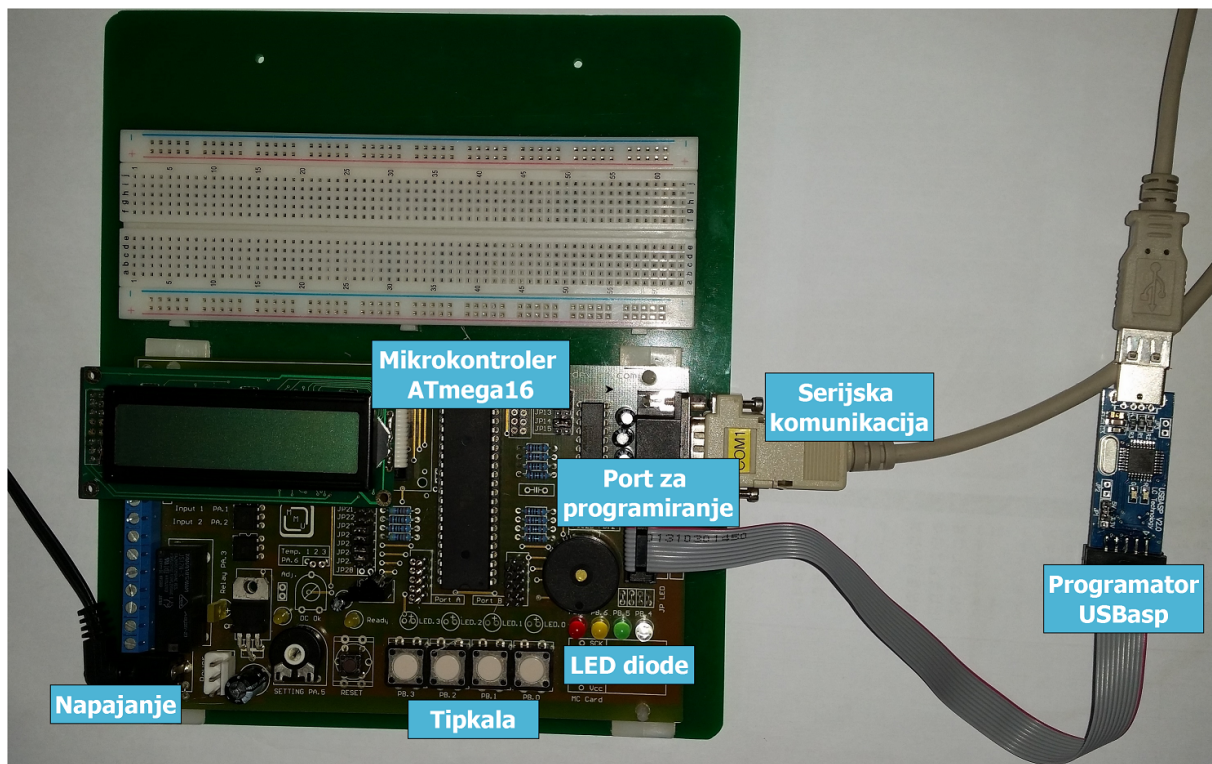


## Poglavlje 3

# Snimanje i pokretanje strojnog koda na mikrokontroleru ATmega16

### 3.1 Razvojno okruženje ATmega16

U prethodnom poglavlju napisali smo programski kod i preveli ga u strojni kod pomoću razvojnog okruženja *Atmel Studio 6*. Strojni kod potrebno je snimiti na mikrokontroler ATmega16 pomoću programatora. U sklopu laboratorijskih vježbi iz kolegija Mikroročunala koristit ćemo razvojno okruženje s mikrokontrolerom ATmega16 prikazano na slici 3.1.



Slika 3.1: Razvojno okruženje s mikrokontrolerom ATmega16

Na slici 3.1 prikazani su osnovni dijelovi razvojnog okruženja s mikrokontrolerom ATmega16:

- napajanje,
- mikrokontroler ATmega16,

- tipkala,
- LED diode,
- serijska komunikacija,
- port za programiranje i
- programator USBasp.

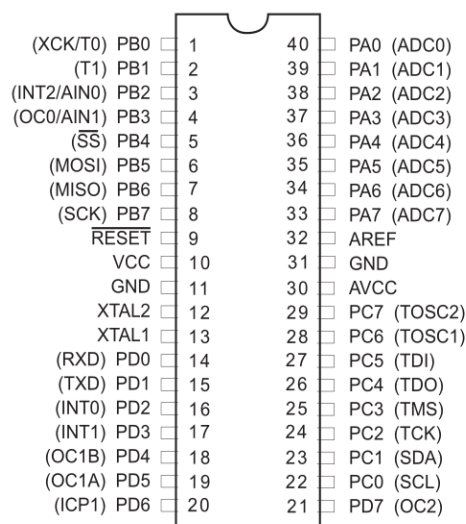
Ostali dijelovi razvojnog okruženja sa slike 3.1 bit će opisani kroz naredna poglavlja.

Napajanje razvojnog okruženja može se kretati od 12 do 35 V iz razloga što je ulazni napon stabiliziran na 5 V pomoću sklopa LM7805.

Mikrokontroler ATmega16 jedan je od najkorištenijih mikrokontrolera opće namjene. Karakteristike ATmega16 mikrokontrolera su sljedeće [1]:

- visoke performanse,
- mala snaga,
- Atmel<sup>®</sup> AVR 8-bitni mikrokontroler,
- RISC arhitektura:
  - 131 instrukcija,
  - 32 registra opće namjene,
  - do 16 MIPS (16 milijuna instrukcija u jednoj sekundi) na frekvenciji od 16 MHz,
- 16 kB (eng. *In-System Self-Programmable*) programske *Flash* memorije,
- 512 B EEPROM memorije,
- 1 kB SRAM memorije,
- ciklus pisanja/brisanja: *Flash* 10 000 puta/EEPROM 100 000 puta,
- JTAG sučelje,
- dva 8-bitna tajmera (eng. *timers*)/brojača (eng. *counters*) s djelitelem frekvencije radnog takta,
- jedan 16-bitni tajmer/brojač s djelitelem frekvencije radnog takta,
- osam kanala analogno-digitalnih pretvornika (rezolucija od 10 bitova),
- serijsko sučelje s dvije žice (eng. *Two-Wire Serial Interface*),
- *Master/Slave* SPI sučelje (eng. *Serial Peripheral Interface*),
- programirajući USART (eng. *Universal Synchronous and Asynchronous Serial Receiver and Transmitter*),
- programirajući vremenski brojač za nadzor ispravnog rada (eng. *Watchdog timer*),
- *Power-on Reset*,
- programirajući detektor pada napona napajanja (eng. *Brown Out Detection*),
- unutarnji i vanjski izvor prekida (eng. *interrupts*),

- 32 digitalna ulazno/izlazna pina,
- DIL podnožje s 40 pinova,
- radni napon: 4,5 - 5,5 V,
- radni takt: 0 - 16 MHz,
- potrošnja (1 MHz, 3 V, 25 °C, ATmega16):
  - aktivan: 1,1 mA,
  - u mirovanju: 0,35 mA.



Slika 3.2: Raspored pinova na mikrokontroleru ATmega16 [1]

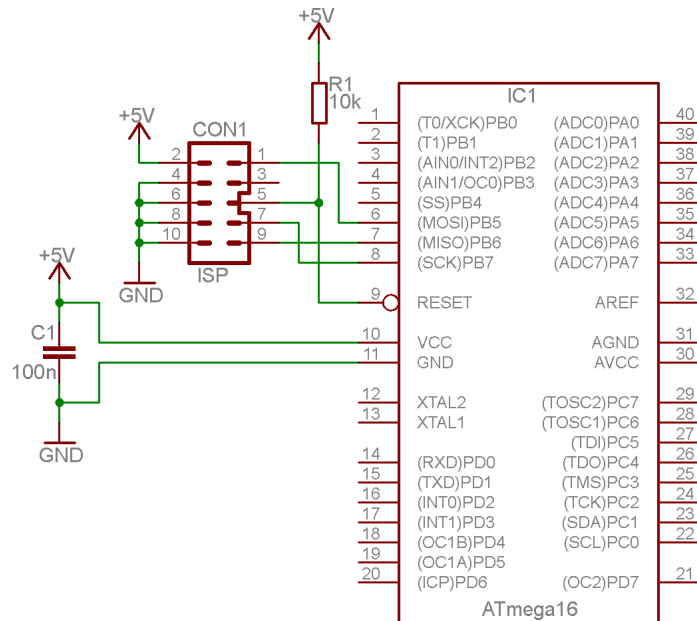
Raspored pinova na mikrokontroleru ATmega16 prikazan je na slici 3.2. Mikrokontroler ima 32 digitalna pina grupirana u četiri skupine koje nazivamo portovima: port A, port B, port C i port D.

Tipkala i LED diode spojeni su na port B mikrokontrolera i koristit će se za testiranje digitalnih ulaza i digitalnih izlaza. Serijska komunikacija koristit će se za komunikaciju mikrokontrolera s računalom. To nam omogućuje upravljanje i nadzor sustava putem računala.

Za programiranje mikrokontrolera ATmega16 koristit ćemo programator USBasp prikazan na slici 3.1. Prednosti ovog programatora naspram ostalih su kompatibilnost s USB sučeljem, niska cijena i besplatni softver za programiranje. Programator je potrebno spojiti na port za programiranje koji ima oznaku ISP<sup>1</sup>. Shema spajanja programatora s mikrokontrolerom ATmega16 prikazana je na slici 3.3. Za programiranje mikrokontrolera potrebno je napajanje od 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 nalazi na portu B (pinovi 5 do 7).

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

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



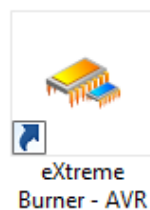
Slika 3.3: Shema spajanja programatora s mikrokontrolerom ATmega16

## 3.2 Programiranje mikrokontrolera ATmega16

Kao programsku podršku programatoru USBasp koristit ćemo softver *eXtreme Burner – AVR*. Najnoviju verziju softvera *eXtreme Burner – AVR* možete pronaći na stranici [www.extremeelectronics.co.in/software/BurnerAVR/](http://www.extremeelectronics.co.in/software/BurnerAVR/).

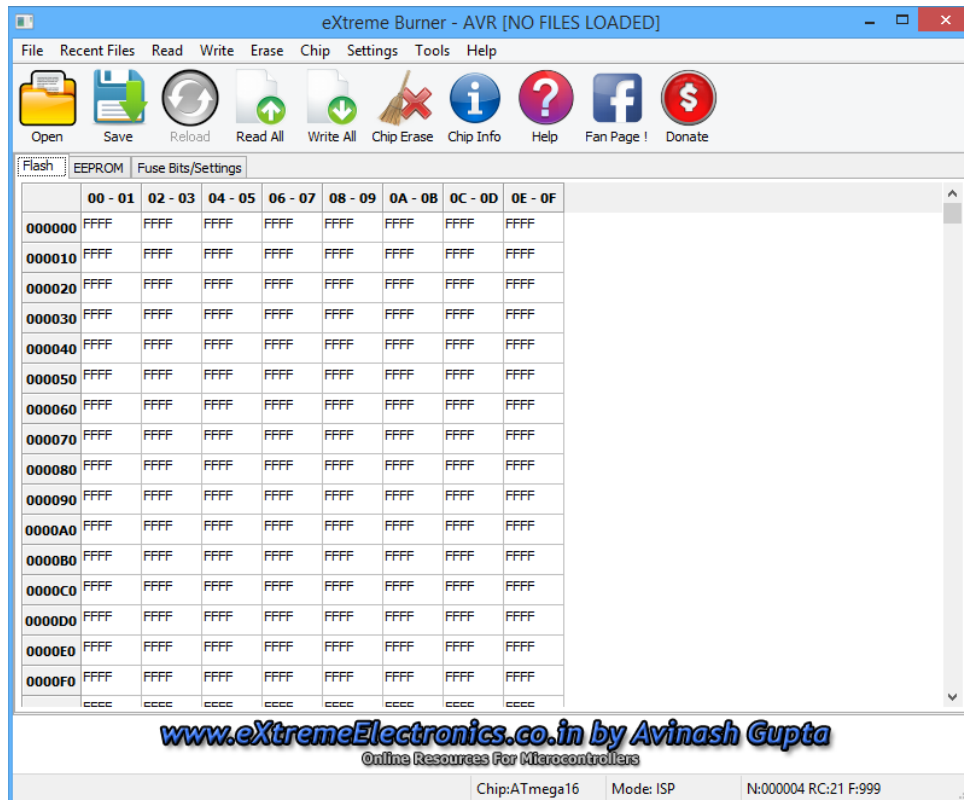
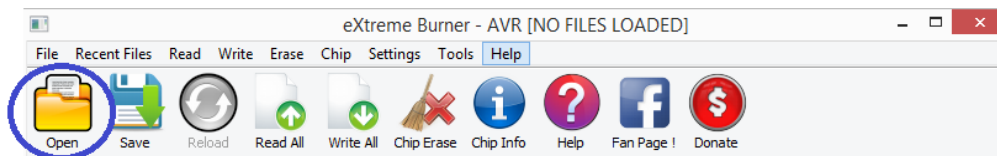
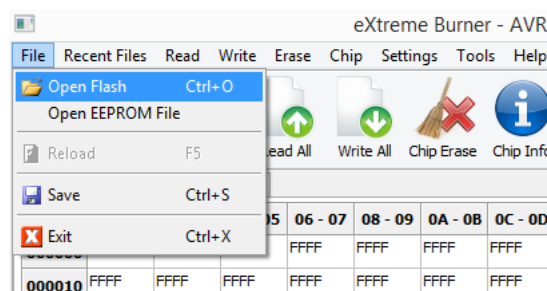
Softver *eXtreme Burner – AVR* možemo pokrenuti dvostrukim klikom na ikonu sa slike 3.4. Nakon pokretanja softvera *eXtreme Burner – AVR* pojavit će se prozor sa slike 3.5. Strojni kod kojeg smo stvorili pomoću razvojnog programskog okruženja *Atmel Studio 6* potrebno je učitati u softver *eXtreme Burner – AVR*. To možemo učiniti na dva načina:

1. na početnoj stranici softvera *eXtreme Burner – AVR* odaberite *Open* (slika 3.6) ili
2. u izborniku softvera *eXtreme Burner – AVR* odaberite *File → Open Flash* (slika 3.7).

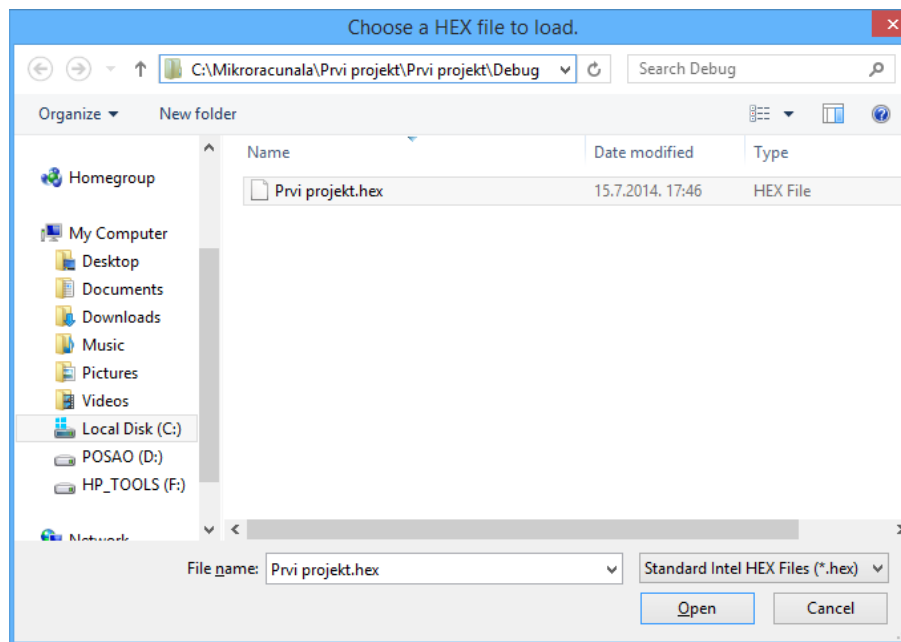


Slika 3.4: Softver *eXtreme Burner – AVR* - ikona



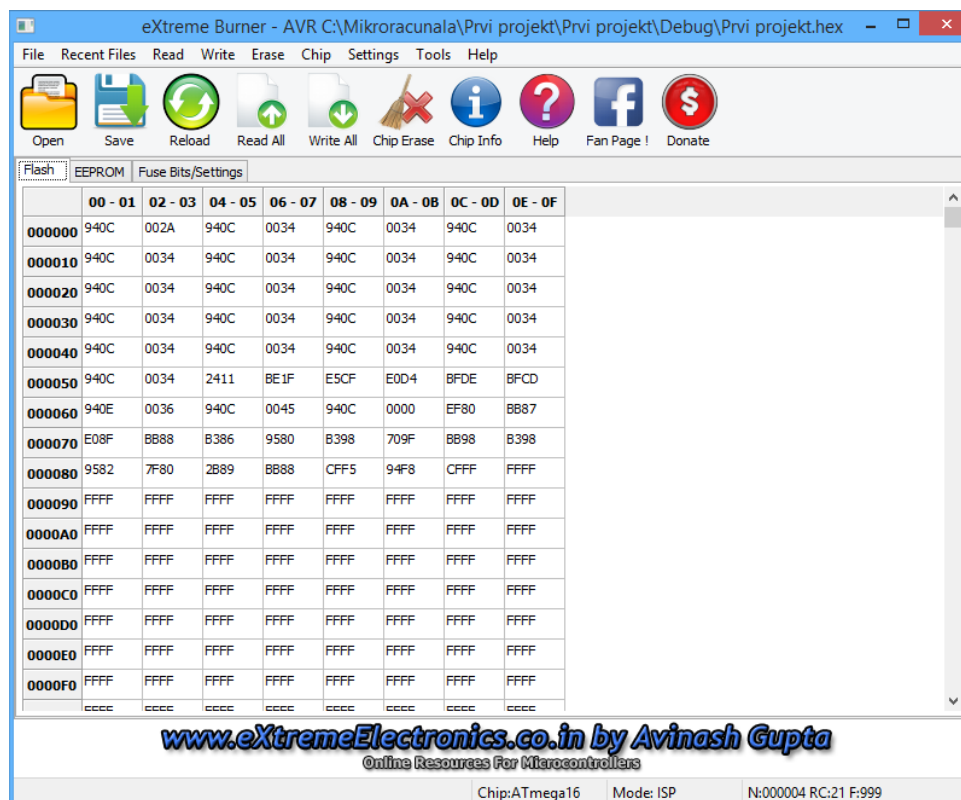
Slika 3.5: Softver *eXtreme Burner - AVR* - početni prozorSlika 3.6: Softver *eXtreme Burner - AVR* - učitavanje strojnog koda (1)Slika 3.7: Softver *eXtreme Burner - AVR* - učitavanje strojnog koda (2)

Prozor za učitavanje strojnog koda prikazan je na slici 3.8. Na slici 3.8 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 6* nalazi se na lokaciji C:\Mikroracunala\Prvi projekt\Prvi projekt\Debug. Na ovoj lokaciji potrebno je odabrati datoteku *Prvi projekt.hex* (slika 3.8).



Slika 3.8: Softver *eXtreme Burner – AVR* - prozor za učitavanje strojnog koda

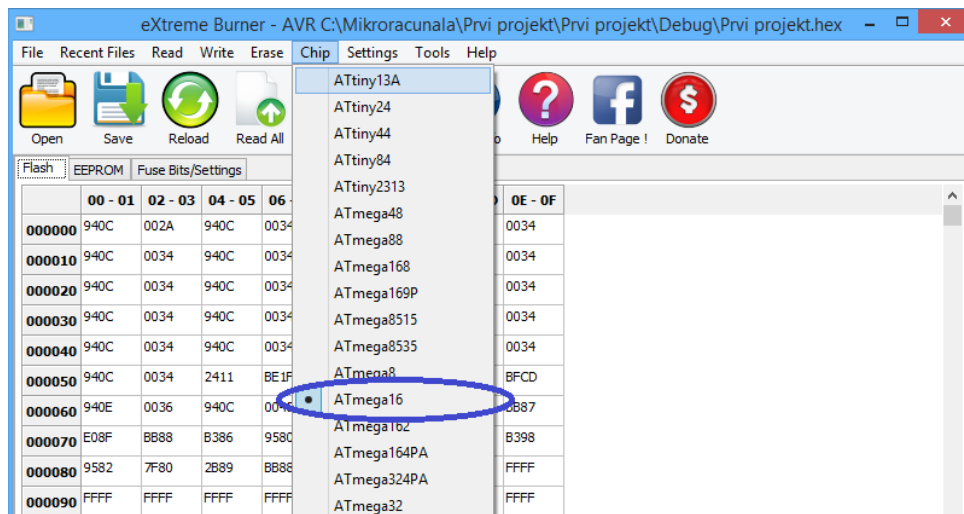
Strojni kod učitani u softver *eXtreme Burner – AVR* prikazan je na slici 3.9 i istovjetan je strojnom kodu sa slike 2.9. Strojni kod nalazi se na kartici *Flash* u softver *eXtreme Burner – AVR*. Osim kartice *Flash*, softveru *eXtreme Burner – AVR* ima karticu EEPROM<sup>3</sup> i *Fuse Bits/Settings*.



Slika 3.9: 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 mikrokontroler snimiti strojni kod. Za odabir mikrokontrolera potrebno je u izborniku *Chip* odabrati jedan od ponuđenih mikrokontrolera porodice *Atmel*. Mikrokontroler koji moramo odabrati je ATmega16 (slika 3.10).



Slika 3.10: Softver *eXtreme Burner - AVR* - odabir AVR mikrokontrolera

Ako prvi puta programiramo mikrokontroler ili mu se mijenjaju osnovne postavke, potrebno je namjestiti tzv. *Fuse* bitove. Ovi bitovi određuju rad mikrokontrolera. U softveru *eXtreme Burner - AVR* odaberite karticu *Fuse Bits/Settings* prikazanu na slici 3.11.



Slika 3.11: Softver *eXtreme Burner - AVR* - podešavanje *Fuse* bitova

*Fuse* bitovima moguće je mijenjati frekvenciju rada mikrokontrolera, izvor radnog takta koji

može biti unutanji i vanjski, način programiranja i drugo. Tvorničke su postavke mikrokontrolera ATmega16 sljedeće:

- frekvencija rada mikrokontrolera je 1 MHz,
- izvor radnog takta je unutarnji oscilator,
- omogućeno je programiranje putem SPI sučelja,
- omogućeno je ispravljanje pogrešaka na mikrokontroleru (eng. *JTAG Interface Enabled*).

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

- niži *Fuse* bitovi (eng. *Low Fuse*) su 0xE1,
- viši *Fuse* bitovi (eng. *High Fuse*) su 0x99.

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 (eng. *Low Fuse*) su 0xE4,
- viši *Fuse* bitovi (eng. *High Fuse*) su 0xD9.

Navedeni *Fuse* bitovi (slika 3.11) odgovaraju sljedećim postavkama mikrokontrolera ATmega16:

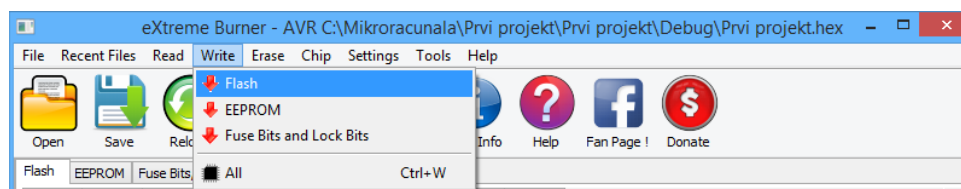
- frekvencija rada mikrokontrolera je 8 MHz,
- izvor radnog takta je unutarnji oscilator,
- omogućeno je programiranje putem SPI sučelja,
- onemogućeno je ispravljanje pogrešaka na mikrokontroleru.

Nakon što proračunamo *Fuse* bitove potrebno ih je snimiti na mikrokontroler. To ćemo učiniti tako da na kartici *Fuse Bits/Settings* slijedimo korake od 1 do 5 (slika 3.11):

1. u polje *Low Fuse* upišite E4,
2. u polje *High Fuse* upišite D9,
3. kvačicom označite *Write* u polju *Low Fuse*,
4. kvačicom označite *Write* u polju *High Fuse*,
5. odaberite *Write*.

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

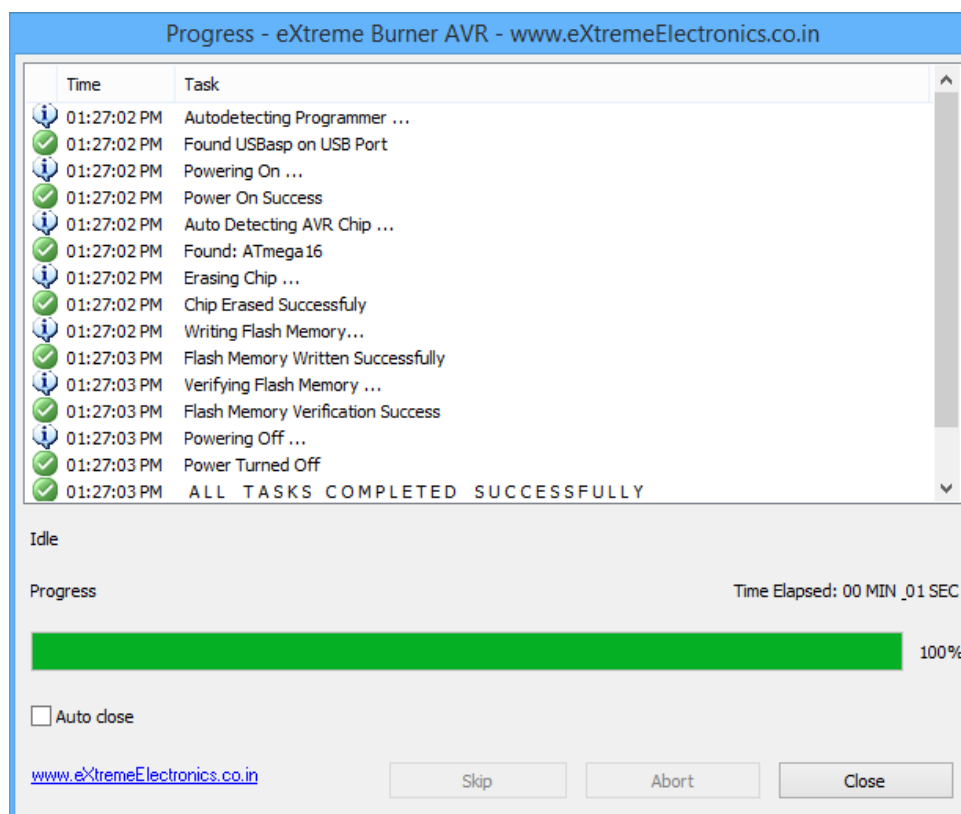
U konačnici, nakon namještanja postavki mikrokontrolera, potrebno je strojni kod snimiti u programsku memoriju mikrokontrolera tako da u softveru *eXtreme Burner – AVR* odaberete *Write → Flash*. Postupak je prikazan na slici 3.12.



Slika 3.12: Softver *eXtreme Burner – AVR* - snimanje strojnog koda u programsku memoriju mikrokontrolera ATmega16

Ukoliko je programiranje mikrokontrolera uspješno završilo, pojavit će se prozor na slici 3.13. Na slici 3.13 možemo vidjeti da proces programiranja mikrokontrolera ima nekoliko koraka:

1. detektiranje mikrokontrolera,
2. brisanje programske memorije mikrokontrolera,
3. snimanje strojnog koda u programsku memoriju mikrokontrolera,
4. verifikacija strojnog koda u mikrokontroleru.

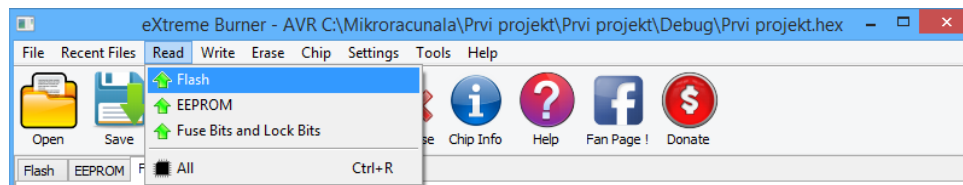


Slika 3.13: Softver *eXtreme Burner – AVR* - proces programiranja mikrokontrolera

Nakon programiranja mikrokontrolera potrebno je provjeriti ispravnost rada programskog koda kojeg 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 mikrokontroler. Ovaj se ciklus ponavlja dok god ne postignemo ispravan rad razvojnog okruženja u skladu s našim naumom.

Strojni kod iz mikrokontrolera moguće je pročitati tako da u softveru *eXtreme Burner – AVR* odaberete *Read* → *Flash* (slika 3.14). Strojni kod nije čitljiv i na temelju njega nećete znati što

i kako radi mikrokontroler. Strojni se kod ne može pretvoriti u programski kod koji je čitljiv programeru mikrokontrolera.



Slika 3.14: Softver *eXtreme Burner – AVR* - čitanje programske memorije mikrokontrolera

Princip programiranja mikrokontrolera isti je i za ostale mikrokontrolere iz porodice *Atmel* koji su na popisu mikrokontrolera u softveru *eXtreme Burner – AVR*.

Programiranje mikrokontrolera porodice *Atmel* moguće je i neposredno iz razvojnog okruženja *Atmel Studio 6*. U tu svrhu potrebno je imati programator AVRISP mkII proizvođača *Atmel*. Ovaj programator puno je skuplji od programatora USBasp, ali zato omogućuje razvoj aplikacije u jednom softveru i brže ispravljanje grešaka.

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

Najčešća greška koju rade početnici u programiranju mikrokontrolera je izostavljanje visoke razine na pinu RESET mikrokontrolera pa iz tog razloga mikrokontroler neće raditi. Na slici 3.3 možemo vidjeti da je visoko stanje na pin RESET dovedeno preko otpornika od 10 kΩ, što znači da je dovoljno na mikrokontroler dovesti samo napajanje kako bi on izvodio strojni kod.

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.

## Poglavlje 4

# Digitalni izlazi i ulazi

Mikrokontroler ATmega16 ima 32 digitalna pina koji se mogu konfigurirati kao izlazni pinovi ili kao ulazni pinovi. Smjer djelovanja digitalnog pina može se mijenjati tijekom rada mikrokontrolera. Na primjer, digitalni pin može biti izlazni pin te tijekom rada mikrokontrolera postati ulazni pin i obratno. Digitalni pinovi mikrokontrolera ATmega16 raspoređeni su u grupe po osam digitalnih pinova koje nazivamo portovima. Mikrokontroler ATmega16 ima sljedeće portove s oznakama digitalnih pinova:

- port A (oznake digitalnih pinova: PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7),
- port B (oznake digitalnih pinova: PB0, PB1, PB2, PB3, PB4, PB5, PB6, PB7),
- port C (oznake digitalnih pinova: PC0, PC1, PC2, PC3, PC4, PC5, PC6, PC7) i
- port D (oznake digitalnih pinova: PD0, PD1, PD2, PD3, PD4, PD5, PD6, PD7).

Svaki digitalni pin ima višestruku namjenu pa se npr. pinovi PD0 i PD1 koriste za serijsku komunikaciju. Način rada digitalnog pina na nekom portu određuju registri za konfiguraciju:

- DDR<sub>x</sub>, ( $x = A, B, C, D$ ), (eng. *Data Direction Register*) - registar smjera podataka,
- PORT<sub>x</sub>, ( $x = A, B, C, D$ ) - podatkovni registar i
- PIN<sub>x</sub>, ( $x = A, B, C, D$ ) - registar ulaznih pinova.

Svi navedeni registri širine su osam bitova. Pozicija bita  $i$  ( $i = 0, 1, \dots, 7$ ) u registru određuje konfiguraciju pina na poziciji  $i$ . Ako mijenjamo primjerice, vrijednost bita u registru DDRA na poziciji bita  $i = 5$ , tada se promijenjena vrijednost bita odnosi na konfiguraciju pina PA5.

Vrijednost bita na poziciji  $i$  ( $i = 0, 1, \dots, 7$ ) u registru DDR<sub>x</sub> određuje hoće li pin na poziciji  $i$  biti ulazni ili izlazni prema pravilima:

- ako je bit na poziciji  $i$  u registru DDR<sub>x</sub> jednak 0, tada će pin na poziciji  $i$  biti konfiguriran kao ulazni pin,
- ako je bit na poziciji  $i$  u registru DDR<sub>x</sub> jednak 1, tada će pin na poziciji  $i$  biti konfiguriran kao izlazni pin.

Na primjer, ako je  $DDRB = 0xF0 = 0b1111000^1$ , 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 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 mikrokontrolera ATmega16

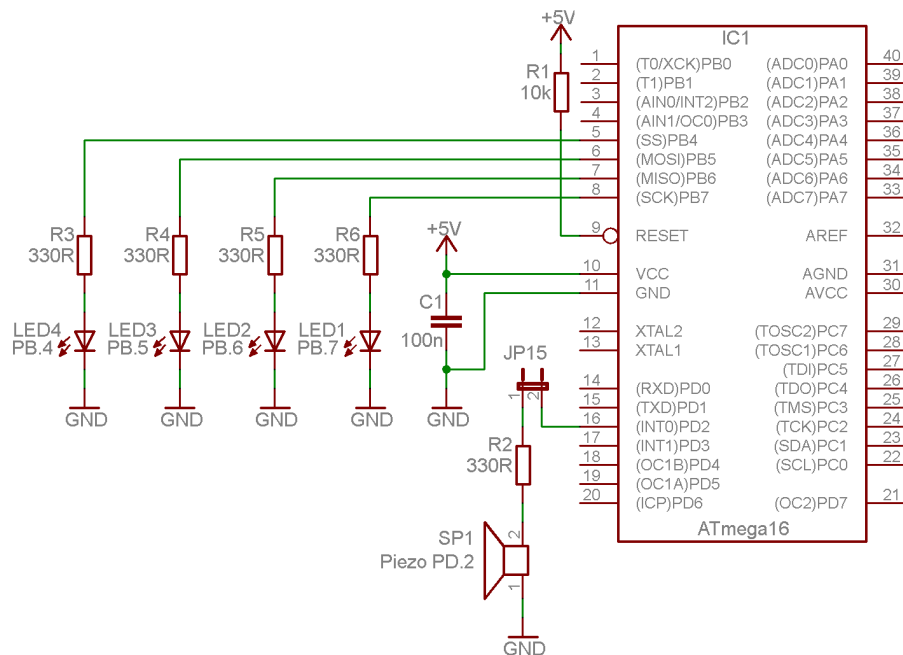
Ako je bit na poziciji  $i$  u registru DDRx jednak 1, tada će pin na poziciji  $i$  biti konfiguriran kao izlazni pin. Pin koji je konfiguriran kao izlaz može biti u dva stanja:

- 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 mikrokontrolera ATmega16



Slika 4.1: Shema spajanja LED dioda i zujalice na mikrokontroler ATmega16



Digitalne izlaze mikrokontrolera ATmega16 testirat ćemo pomoću četiri LED diode i jedne zujalice. Shema spajanja LED dioda i zujalice na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1.

LED diode i zujalicu moguće je spojiti na bilo koje digitalne pinove. Shema sa slike 4.1 usklađena je 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 je strujna zaštita digitalnog pina. Struja digitalnog pina ne smije biti veća od 20 mA. Vrijednost otpora otpornika je 330  $\Omega$ .

Na digitalne izlaze možemo spojiti bipolarne i unipolarne tranzistore, releje s maksimalnom upravljačkom strujom od 20 mA, optičke sprežnike (eng. *Optocoupler*), dijkke, trijke i ostale digitalne aktuatora uz uvjet da se ne premaši maksimalna struja digitalnog pina.

Za detalje oko konfiguracije digitalnih izlaza pogledajte tablicu 20 u literaturi [1]. Programsko razvojno okruženje *Atmel Studio 6* ima definirana imena registara DDRx i PORTx (x = A, B, C, D) te se konfiguracija pinova svodi na dodjeljivanje vrijednosti u definirana imena registara. Na primjer, ako želimo da svi pinovi porta A budu izlazni pinovi u programskom razvojnom okruženju *Atmel Studio 6*, napisali bismo `DDRA = 0xFF`; u heksadecimalnom zapisu ili `DDRA = 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 6*.

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala) skinite datoteku *Digitalni izlazi.zip*. Na radnoj površini stvorite praznu datoteku koju ćete nazvati *Vaše Ime i Prezime* ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Ivica Ivić, datoteka koju ćete stvoriti zvat će se *Ivica Ivic*. Datoteku *Digitalni izlazi.zip* raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite *mikroracunala.atsln* u datoteci `\\Digitalni izlazi\vjezbe`. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju *Digitalni izlazi mikrokontrolera ATmega16*. Vježbe ćemo pisati u datoteke s ekstenzijom `*.c`. 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 4.2):

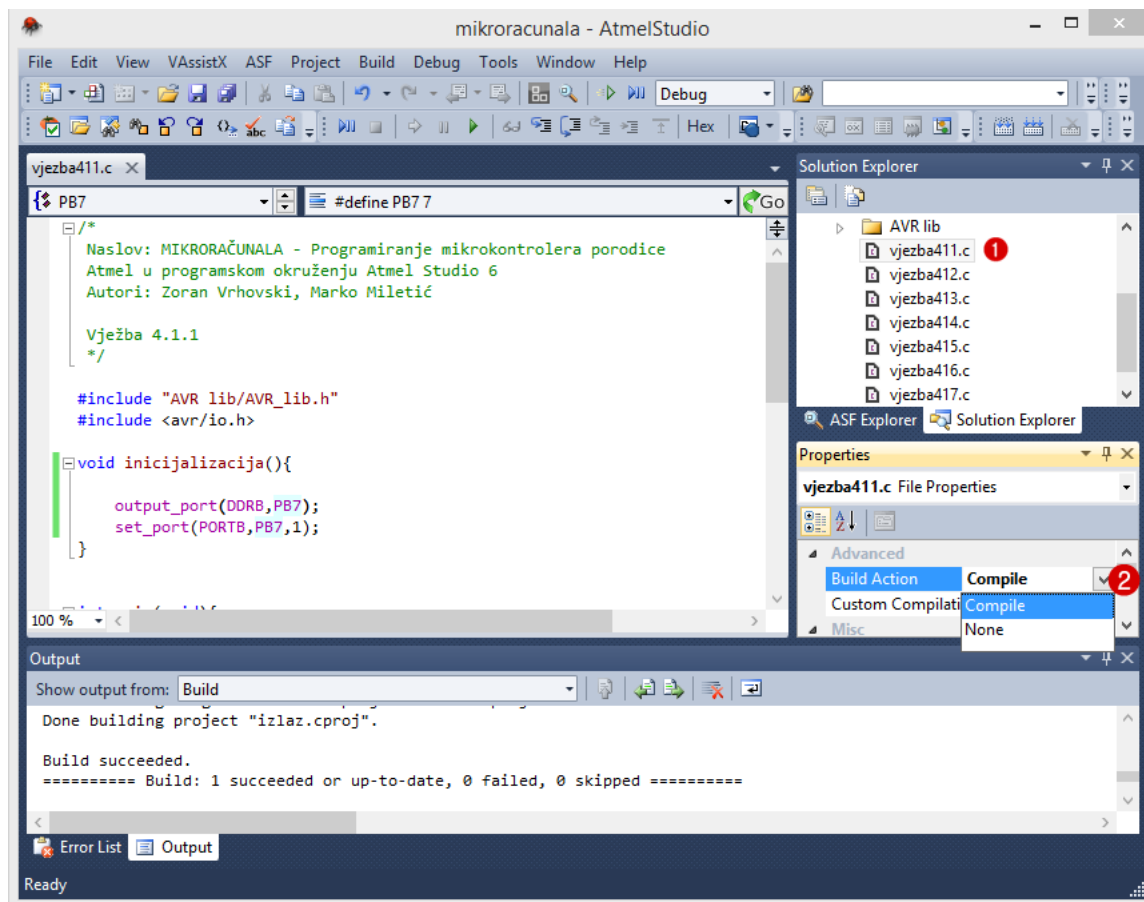
1. u projektnom stablu odaberite datoteku s ekstenzijom `*.c` (npr. `vježbu411.c`) i na njoj pritisnite desni gumb miša te odaberite **Properties**,
2. ispod prozora projektnog stabla otvorit će se prozor sa svojstvima datoteke koju ste odabrali. Za datoteku koju želite prevesti u strojni kod potrebno je u polju *Build Action* odabrati opciju *Compile*, a za sve ostale datoteke u projektnom stablu koje sadrže funkciju `main()` u polju *Build Action* odabrati opciju *None*.

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 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 prevodenje vježbe.

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

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



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



### Vježba 4.1.1

Napravite program koji će uključiti crvenu LED diodu na razvojnom okruženju s mikrokontrolerom ATmega16. Shema spajanja crvene LED diode na digitalni izlaz PB7 mikrokontrolera ATmega16 prikazana je na slici 4.1.

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

- `#include "AVR_lib/AVR_lib.h"` - zaglavlje `AVR_lib.h` s definiranim makronaredbama za konfiguraciju mikrokontrolera. Zaglavlje `AVR_lib.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 mikrokontroler za koji je stvoren projekt u programskom razvojnom okruženju *Atmel Studio 6*. Zaglavlje `io.h` razvijeno je u poduzeću *Atmel* te se u programski kod uključuje naredbom `#include <>`.,
- `void inicijalizacija()` - inicijalizacijska funkcija koju programeri mikrokontrolera često koriste za definiranje i objedinjavanje početnih postavki mikrokontrolera. Naziv inicijalizacijske funkcije proizvoljan je.

Programski kod 4.1: Početni sadržaj datoteke vjezba411.c

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

void inicijalizacija(){
}

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

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

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

Programski kod 4.2: Funkcija inicijalizacije mikrokontrolera - prvi način

```
void inicijalizacija(){
    DDRB = 0b10000000;
    PORTB = 0x80;
}
```

Tijelo funkcije `inicijalizacija()` mora odgovarati programskom kodu 4.2. Sada je potrebno datoteku `vjezba411.c` prevesti u strojni kod sukladno uputama iz poglavlja **Razvojno okruženje Atmel Studio 6**. Strojni kod pomoću softvera *eXtreme Burner – AVR* snimate na mikrokontroler ATmega16 prema uputama iz poglavlja **Snimanje i pokretanje strojnog koda na mikrokontroleru ATmega16**. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Inicijalizacija 4.2 samo je jedan od načina inicijalizacije mikrokontrolera 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 mikrokontrolera - 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.c` putem uključenog zaglavlja `io.h`. U programskom okruženju *Atmel Studio 6* iznad konstante `PB7` pritisnite desni gumb miša te odaberite `Goto Implementation`. Otvorit će se zaglavlje u kojoj su definirane konstante za sve portove. Odvojite vremena i pogledajte ostali sadržaj otvorenog zaglavlja. Opcija `Goto Implementation` vrlo je korisna i preporučuje se njeno 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Programski kod 4.4: Funkcija inicijalizacije mikrokontrolera - treći način

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

Treći način inicijalizacije je najjednostavniji. Autori udžbenika napisali su makronaredbe koje su intuitivne i jednostavne za korištenje. Makronaredbe se nalaze u zaglavlju `AVR_lib.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 kojeg 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. Preporučujemo 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.1.2

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 tri puta uključiti i isključiti crvenu LED diodu u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja crvene LED diode na digitalni izlaz PB7 mikrokontrolera ATmega16 prikazana je na slici 4.1.

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

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

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

void inicijalizacija(){

    DDRB |= (1 << PB7); // pin PB7 postavljen kao izlazni port
    PORTB |= (1 << PB7); // postavljanje pina PB7 u logičku jedinicu
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

return 0;
}
```

U vježbi je zadano da se crvena 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 mikrokontrolera na zadano vrijeme što u principu i nije dobro jer se onemogućuje obrada podataka s vanjskih senzora u tom vremenu. Za sada je ovo jedino rješenje koje će nam omogućiti uključivanje i isključivanje crvene 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, *Fuse* bitova kao što smo to već pokazali. U programskom okruženju *Atmel Studio 6* frekvencija se definira konstantom `F_CPU`. Frekvencija mikrokontrolera namještena je na 8 MHz sukladno prethodnom potrebno je znati frekvenciju rada mikrokontrolera. Na razini hardvera, frekvencija se namješta pomoću poglavlju, a ukoliko to nije tako, namjestite *Fuse* bitove tako da frekvencija rada mikrokontrolera iznosi 8 MHz.

U zaglavlju `AVR_lib.h` nalazi se naredba `#define F_CPU 8000000u1` kojom se prevoditelju ukazuje da frekvencija rada mikrokontrolera iznosi 8 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 mikrokontrolera iznosi 1 MHz. Usklađenost frekvencija na hardverskoj i softverskoj razini lako je testirati na razvojnom okruženju s mikrokontrolerom ATmega16.

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

Prevedite datoteku `vjezba412.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Zašto crvena LED dioda ne izmjenjuje svoje stanje nakon naredbe `(PORTB &= ~(1 << PB7));`?

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



### Vježba 4.1.3

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti beskonačno uključivanje i isključivanje crvene LED diode u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja crvene LED diode na digitalni izlaz PB7 mikrokontrolera ATmega16 prikazana je na slici 4.1.

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

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

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

void inicijalizacija(){

    DDRB |= (1 << PB7); // pin PB7 postavljen kao izlazni port
    PORTB |= (1 << PB7); // postavljanje pina PB7 u logičku jedinicu
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

return 0;
}
```

U vježbi je zadano da se crvena LED dioda mora neprestano uključivati i isključivati u razmaku od jedne sekunde. Uvjet zadatka bit će zadovoljen dok mikrokontroler ima napajanje i dok je RESET pin u visokom stanju. Kako bismo ostvarili beskonačnu izmjenu visokog i niskog stanja na crvenoj 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 mikrokontroler
// 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 crvene 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 prolazu kroz blok naredbi beskonačne `while` petlje prethodne dvije naredbe se ponovno izvode. U `while` petlji čeka se jednu sekundu i nakon toga se stanje mijenja što će dovesti do beskonačne izmjene stanja crvene LED diode.

Prevedite datoteku `vjezba413.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Pokušajte ubrzati izmjenu stanja crvene LED diode.

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



#### Vježba 4.1.4

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 100 ms i to redosljedom crvena → žuta → zelena → bijela → crvena → ... . Za izmjenu stanja LED diode koristite makronaredbu `TOGGLE_PORT`. Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin `PB7`, žuta LED dioda spojena je na digitalni pin `PB6`, zelena LED dioda spojena je na digitalni pin `PB5`, a bijela LED dioda spojena je na digitalni pin `PB4`.

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

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

```
#include "AVR lib/AVR_lib.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 << PB7); // postavljanje PB7 u visoko stanje
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    while (1) // beskonačna petlja
    {
        // blok naredbi za trčanje LED dioda upišite ovdje
    }

    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 bijelu 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 bijelu 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 << PB7);`). U programskom kodu 4.8 u blok naredbi beskonačne `while` petlje upišite sljedeće naredbe:

- `_delay_ms(100);` - kašnjenje od 100 ms.
- `TOGGLE_PORT(PORTB, PB7);` - promijeni stanje crvene LED diode. Prethodno je bila uključena, a nakon ove naredbe bit će isključena.
- `TOGGLE_PORT(PORTB, PB6);` - 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, PB6);` - promijeni stanje žute LED diode. Prethodno je bila uključena, a nakon ove naredbe bit će isključena.
- `TOGGLE_PORT(PORTB, PB5);` - 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, PB4);` - promijeni stanje bijele 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.1.5

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 100 ms i to redoslijedom crvena → žuta → zelena → bijela → crvena → ... . Za izmjenu stanja LED diode koristite `for` petlju. Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4.

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

Programski kod 4.9: Početni sadržaj datoteke `vjezba415.c`

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

void inicijalizacija(){
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    while (1) // beskonačna petlja
    {

        for (char i = 7; i >= 4; i--){

            _delay_ms(100);

            PORTB &= ~(1 << i);

            if (i > 4){
                PORTB |= (1 << (i-1));
            }
            else{
                PORTB |= (1 << PB7);
            }

        }

    }

    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 7, 6, 5 i 4. Primijetite da su to pozicije digitalnih pinova PB7, PB6, PB5 i PB4. 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` veći od 4. Ako je `i` veći od 4, uključujemo sljedeću LED diodu naredbom `PORTB |= (1 << (i-1));`. Onog trenutka kada brojač `for` petlje `i` poprimi vrijednost 4 uključuje se crvena LED dioda naredbom `PORTB |= (1 << PB7);` kao početno stanje za sljedeća četiri prolaza kroz `for` petlju.

Prevedite datoteku `vjezba415.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.1.6

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti 50 ciklusa „trčanja” svih LED dioda svakih 100 ms i to redosljedom crvena → žuta → zelena → bijela → crvena → ... . Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4.

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

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

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

void inicijalizacija(){

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

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    for (char i = 0; i < 50; i++){ // petlja koja se izvodi 50 puta
    // ovdje kopirati blok naredbi while petlje iz datoteke vjezba414.c
    }

    // ovdje ugasiti crvenu LED diodu

    return 0;
}
```

U vježbi je ponovno potrebno ostvariti „trčanje” LED dioda, ali konačni broj puta. U pro-

gramskom kodu 4.10 u blok naredbi `for` petlje kopirajte blok naredbi `while` petlje iz datoteke `vjezba414.c`. U praksi se često ponavljaju dijelovi programskog koda pa je uobičajeno kopirati gotovi programski kod u trenutni projekt koji radite.

Prevedite datoteku `vjezba416.c` 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`.

Prevedite datoteku `vjezba416.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.1.7

Napravite program kojim će razvojno okruženje s mikrokontrolerom ATmega16 reproducirati zvučnu signalizaciju pomoću zujalice i funkcije `BUZZ(double trajanje, int frekvencija)`. Shema spajanja zujalice na digitalni izlaz PD2 mikrokontrolera ATmega16 prikazana je na slici 4.1.

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

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

```
#include "AVR lib/AVR_lib.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 PD2. Na shemi sa slike 4.1 primijetite da je zujalica na mikrokontroler spojena preko kratkospojnika (eng. *Jumper*) JP15. Ako na razvojnom okruženju s mikrokontrolerom ATmega16 nema kratkospojnika JP15, zujalica neće raditi. Zujalicu možemo spojiti na bilo koji digitalni pin.

Definicija funkcije `BUZZ()` nalazi se u datoteci `AVR lib/AVR_lib.c`. Proučite definiciju ove funkcije. U datoteci `AVR lib/AVR_lib.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 PD2 na koji je spojena zujalica

```

#define BUZZER_PORT PORTD
#define BUZZER_DDR  DDRD
#define BUZZER_PIN  PD2

```

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

- `#define BUZZER_PORT PORTD` - podatkovni registar koji uključuje i isključuje digitalni pin PD2,
- `#define BUZZER_DDR DDRD` - registar smjera podataka koji definira digitalni pin PD2 kao izlazni pin,
- `#define BUZZER_PIN PD2` - pozicija digitalnog pina PD2.

Ako bi zujalica bila spojena na digitalni pin PA7, programski kod 4.12 trebalo bi promijeniti na način prikazan u programskom kodu 4.13.

Programski kod 4.13: Definiranje digitalnog pina PA7 na koji je spojena zujalica

```

#define BUZZER_PORT PORTA
#define BUZZER_DDR  DDRA
#define BUZZER_PIN  PA7

```

Prevedite datoteku `vjezba417.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Sada pokušajte napraviti vlastitu zvučnu signalizaciju. Ponovno prevedite datoteku `vjezba417.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba417.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

### 4.1.2 Zadaci - digitalni izlazi mikrokontrolera ATmega16

#### Zadatak 4.1.1

Napravite program koji će uključiti zelenu LED diodu na razvojnom okruženju s mikrokontrolerom ATmega16. Shema spajanja zelene LED diode na digitalni izlaz PB5 mikrokontrolera ATmega16 prikazana je na slici 4.1.

---

#### Zadatak 4.1.2

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 tri puta uključiti i isključiti zelenu LED diodu u razmaku od jedne sekunde između svakog uključivanja. Shema spajanja zelene LED diode na digitalni izlaz PB5 mikrokontrolera ATmega16 prikazana je na slici 4.1.

---

#### Zadatak 4.1.3

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti beskonačno uključivanje i isključivanje zelene LED diode svakih 500 ms. Shema spajanja zelene LED diode na digitalni izlaz PB5 mikrokontrolera ATmega16 prikazana je na slici 4.1.

---

#### Zadatak 4.1.4

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 350 ms i to redoslijedom bijela → zelena → žuta → crvena → bijela → ... . Za izmjenu stanja LED diode koristite makronaredbu `TOGGLE_PORT`. Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4.

---

#### Zadatak 4.1.5

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 350 ms i to redoslijedom bijela → zelena → žuta → crvena → bijela → ... . Za izmjenu stanja LED diode koristite `for` petlju. Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4.

---

#### Zadatak 4.1.6

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 omogućiti

100 ciklusa „trčanja” svih LED dioda svakih 200 ms i to redosljedom bijela → zelena → žuta → crvena → bijela → ... . Shema spajanja LED dioda na digitalne izlaze mikrokontrolera ATmega16 prikazana je na slici 4.1. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4.

---

#### Zadatak 4.1.7

Napravite program kojim će razvojno okruženje s mikrokontrolerom ATmega16 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 PD2 mikrokontrolera ATmega16 prikazana je na slici 4.1.

---

## 4.2 Digitalni ulazi mikrokontrolera ATmega16

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 dva stanja:

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

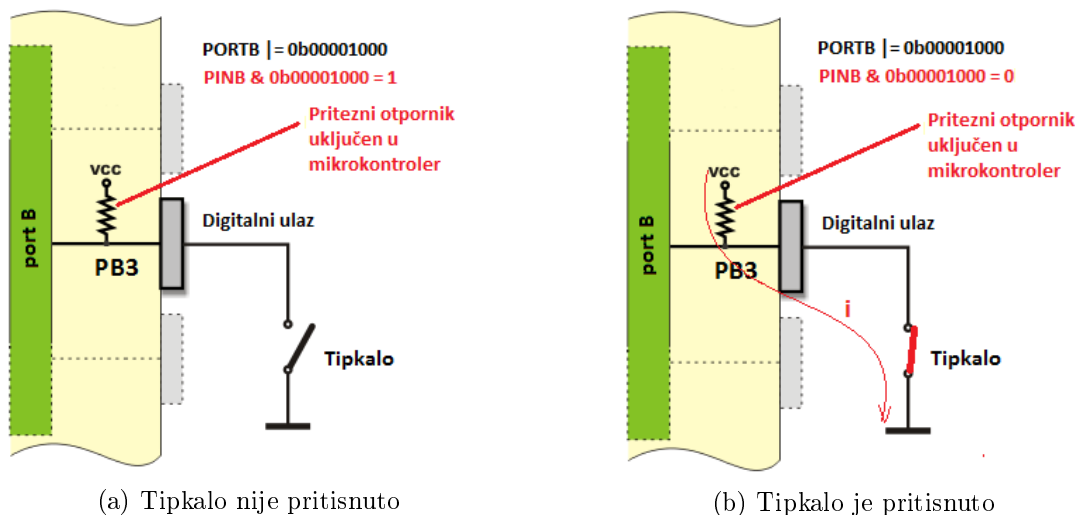
Nisko ili visoko stanje na ulaznom pinu možemo pročitati u registru PINx prema pravilima:

- 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 (eng. *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 4.3.



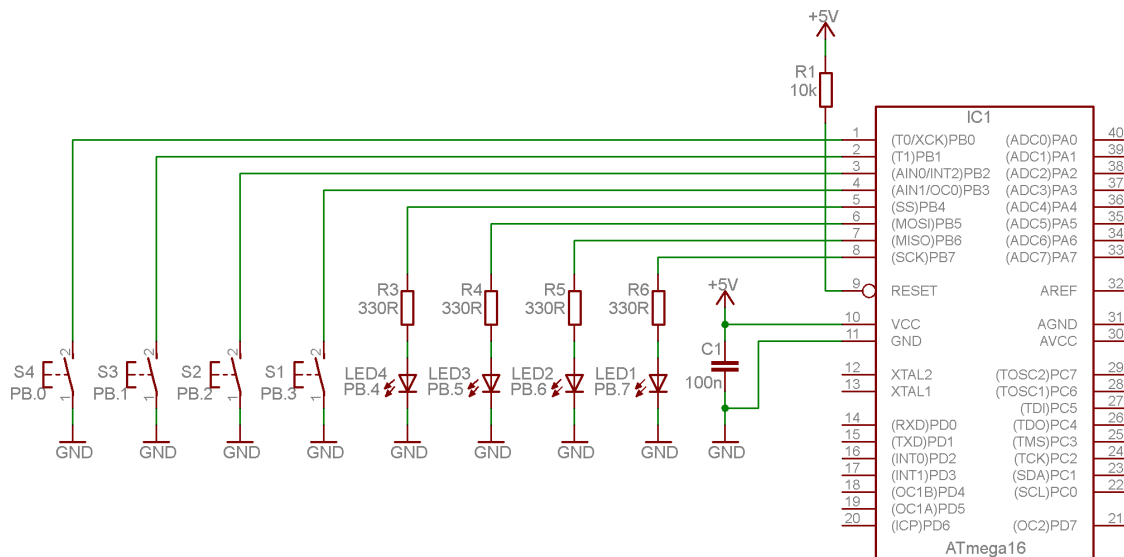
Slika 4.3: Tipkalo spojeno na ulazni pin PB3 mikrokontrolera ATmega16

Pretpostavimo da pritezni otpornik nije uključen. Koliko je potencijal na digitalnom ulazu PB3 ako tipkalo nije pritisnuto (slika 4.3a)? To ne znamo, jer digitalni je ulaz u stanju visoke impedancije te u registru PINB nećemo dobiti ispravno očitavanje. Bilo kakav manji poremećaj može promijeniti stanje bita 3 u registru PINB. Iz tog se razloga pin „pritegne” na potencijal 5 V. Kada tipkalo nije pritisnuto (slika 4.3a), stanje pina je visoko, odnosno 5 V, a stanje bita 3 u registru PINB bit će 1. Kada je tipkalo pritisnuto (slika 4.3b), stanje pina je nisko, odnosno 0 V, a stanje bita 3 u registru PINB bit će 0. Struja će u slučaju pritisnutog tipkala teći iz mikrokontrolera prema masi (slika 4.3b).



Pritezni otpornici uključuju se ako se na mikrokontroler spajaju tipkala i senzori s otvorenim kolektorom. Postoje i senzori koji aktivno na svom izlazu mogu dati i visoko i nisko stanje<sup>3</sup>. Ako na mikrokontroler spajamo takav senzor, pritezni otpornik nije potrebno uključivati<sup>4</sup>. Pritezni otpornik može se spojiti izvana na mikrokontroler, a njegova vrijednost je najčešće 10 k $\Omega$ .

#### 4.2.1 Vježbe - digitalni ulazi mikrokontrolera ATmega16



Slika 4.4: Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16

Digitalne ulaze mikrokontrolera ATmega16 testirat ćemo pomoću četiri tipkala spojena na pinove PB0, PB1, PB2 i PB3. Shema spajanja tipkala na digitalne ulaze mikrokontrolera ATmega16 prikazana je na slici 4.4. Na slici 4.4 nalaze se i LED diode koje ćemo uz tipkala također koristiti u ovoj vježbi.

Za detalje o konfiguraciji digitalnih ulaza pogledajte tablicu 20 u literaturi [1]. Programsko razvojno okruženje *Atmel Studio 6* ima definirana imena registara DDRx, PORTx i PINx (x = A, B, C, D). 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 6*, napisat ćemo `DDRD = 0x00`; u heksadecimalnom zapisu ili `DDRD = 0b00000000`; u binarnom zapisu. Ukoliko na portu D želimo uključiti pritezne otpornike na svim pinovima u programskom razvojnom okruženju *Atmel Studio 6*, 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 6* vrijednost registra `PIND` bila `0x0F = 0b00001111`.

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala) skinite datoteku `Digitalni ulazi.zip`. Na radnoj površini stvorite praznu datoteku koju ćete nazvati **Vaše Ime i Prezime** ne koristeći pritom dijakritičke znakove. Na primjer, ako je Vaše ime Ivica Ivić, datoteka koju ćete stvoriti

<sup>3</sup>Senzori s tzv. *push-pull* izlazom.

<sup>4</sup>Ništa se neće dogoditi ako uključite pritezni otpornik.

zvat će se Ivica Ivic. Datoteku `Digitalni_ulazi.zip` raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite `mikroracunala.atstln` u datoteci `\\Digitalni_ulazi\vjezbe`. U otvorenom projektu nalaze se sve vježbe koje ćemo obraditi u poglavlju `Digitalni_ulazi` mikrokontrolera `ATmega16`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.c`.

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



### Vježba 4.2.1

Napravite program koji će na razvojnom okruženju s mikrokontrolerom `ATmega16` uključiti sve LED diode ako je pritisnuto tipkalo spojeno na pin `PB0`. Shema spajanja tipkala i LED dioda na mikrokontroler `ATmega16` prikazana je na slici 4.4.

U projektnom stablu otvorite datoteku `vjezba421.c`. Omogućite prevođenje samo datoteke `vjezba421.c`. Početni sadržaj datoteke `vjezba421.c` prikazan je programskim kodom 4.14.

Programski kod 4.14: Početni sadržaj datoteke `vjezba421.c`

```
#include "AVR_lib/AVR_lib.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(DDRB,PB0); // PB0 postavljen kao ulazni pin
    set_port(PORTB,PB0,1); // uključenje priteznog otpornika na PB0
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    while (1)
    {
        if((PINB & 0x01) == 0x00){ // ako je pin PB0 u logičkoj nuli
            PORTB |= 0xF0; // uključiti sve LED diode
        }
        else{
            PORTB &= ~0xF0; // inače ih isključiti
        }
    }

    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 kojeg 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 PB0. Kada je tipkalo pritisnuto, na pinu PB0 je nisko stanje, a vrijednost bita 0 u registru `PINB` je 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 PB0 u programskom razvojnom okruženju *Atmel Studio 6* ispituje se naredbom `if((PINB & 0x01)== 0x00)`. Rezultat bitovne operacije `PINB & 0x01` bit će jednak `0x00` ako je tipkalo pritisnuto (tablica 4.3). Ako tipkalo nije pritisnuto, rezultat bitovne operacije `PINB & 0x01` bit će jednak `0x01` (tablica 4.4).

Tablica 4.3: Ispitivanje stanja pina PB0 - 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>PINB</b>	x	x	x	x	x	x	x	0
<b>0x01</b>	0	0	0	0	0	0	0	1
<b>PINB &amp; 0x01</b>	0	0	0	0	0	0	0	0

Tablica 4.4: Ispitivanje stanja pina PB0 - 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>PINB</b>	x	x	x	x	x	x	x	1
<b>0x01</b>	0	0	0	0	0	0	0	1
<b>PINB &amp; 0x01</b>	0	0	0	0	0	0	0	1

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

Prevedite datoteku `vjezba421.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Mikrokontroler se može inicijalizirati i funkcijom `inicijalizacija()` prikazanom programskim kodom 4.15.

Programski kod 4.15: Funkcija inicijalizacije mikrokontrolera - drugi način

```
void inicijalizacija(){
    DDRB = 0xF0;
    PORTB = 0x01;
}
```

Ovaj način inicijalizacije koristi se samo onda kada su konfiguracije svih pinova na nekom portu poznate. Kao što smo već i prije rekli, najčešće nisu poznate sve konfiguracije pinova ili se one mijenjaju tijekom rada mikrokontrolera. Promijenite funkciju `inicijalizacija()` u programskom razvojnom okruženju *Atmel Studio 6* sukladno programskom kodu 4.15. Prevedite datoteku `vjezba421.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Treći način inicijalizacije mikrokontrolera je pomoću bitovnih operatora (programski kod 4.16). Ovo je najsloženiji način inicijalizacija, ali daje jasnu sliku o konfiguraciji pinova. Promijenite funkciju `inicijalizacija()` u programskom razvojnom okruženju *Atmel Studio 6* sukladno programskom kodu 4.16. Prevedite datoteku `vjezba421.c` u strojni kod i snimite ga

na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Programski kod 4.16: Funkcija inicijalizacije mikrokontrolera - treći način

```
void inicijalizacija(){
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);
    DDRB &= ~(1 << PB0);

    PORTB |= (1 << PB0);
}
```

Izbrišite naredbu `PORTB |= (1 << PB0);` u funkciji `inicijalizacija()`. Prevedite datoteku `vjezba421.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Pokušajte prstima dodirivati pinove porta B mikrokontrolera ATmega16 na razvojnom okruženju. Primijetite da se LED diode neprestano uključuju i isključuju iako niste pritisnuli tipkalo spojeno na pin PB0. Onog trenutka kada smo u funkciji `inicijalizacija()` izbrisali naredbu `PORTB |= (1 << PB0);`, isključili smo pritezni otpornik na pinu PB0. Kada tipkalo nije pritisnuto, pin PB0 nalazi se u stanju visoke impedancije te svaki vanjski poremećaj utječe na promjenu stanja bita 0 u registru `PINB`.

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



## Vježba 4.2.2

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

U projektnom stablu otvorite datoteku `vjezba422.c`. Omogućite prevođenje samo datoteke `vjezba422.c`. Početni sadržaj datoteke `vjezba422.c` prikazan je programskim kodom 4.17.

Programski kod 4.17: Početni sadržaj datoteke `vjezba422.c`

```
#include "AVR lib/AVR_lib.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);
    // PB3,PB2,PB1,i PBO postavljeni kao izlazni pinovi
    DDRB &= ~(1 << PB3) | (1 << PB2) | (1 << PB1) | (1 << PBO));
    // pritezni otpornici uključeni na pinovima PB3,PB2,PB1 i PBO
    PORTB |= (1 << PB3) | (1 << PB2) | (1 << PB1) | (1 << PBO);
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera
```

```
while (1)
{
    if((PINB & 0x08) == 0x00){ // ako je pin PB3 u logičkoj nuli
        PORTB |= (1 << PB7); // uključi crvenu LED diodu
    }
    else{
        PORTB &= ~(1 << PB7); // inače je isključi
    }

    // nastavite za ostala tipkala
}

return 0;
}
```

U vježbi je potrebno uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB3, žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PB2, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB1 i bijelu LED diodu ako je pritisnuto tipkalo spojeno na pin PB0.

U programskom kodu 4.17 prikazano je tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PB3, PB2, PB1 i PB0 konfigurirani kao ulazni. Uključeni su pritezni otpornici za pinove PB3, PB2, PB1 i PB0. Pri pokretanju strojnog koda u mikrokontroleru svi DDRx registri inicijalno su u stanju 0x00 pa smo stoga mogli izostaviti dio tijela funkcije `inicijalizacija()` koji konfigurira ulazne pinove. Preporučujemo da unatoč tome ne izostavljate konfiguraciju ulaznih pinova zbog preglednosti programskog koda.

U `while` petlji prikazanoj u programskom kodu 4.17 nalazi se niz naredbi koje će uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB3. Proširite niz naredbi `while` petlje tako da uključite žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PB2, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB1 i bijelu LED diodu ako je pritisnuto tipkalo spojeno na pin PB0. Maska za naredbu `if((PINB & maska)== 0x00)` formira se na sljedeći način:

- `maska = 0x02` - ako želite provjeriti stanje pina PB1 (0x02 na poziciji bita 1 ima vrijednost 1, ostali bitovi imaju vrijednost 0),
- `maska = 0x04` - ako želite provjeriti stanje pina PB2 (0x04 na poziciji bita 2 ima vrijednost 1, ostali bitovi imaju vrijednost 0),
- `maska = 0x08` - ako želite provjeriti stanje pina PB3 (0x08 na poziciji bita 3 ima vrijednost 1, ostali bitovi imaju vrijednost 0).

Prevedite datoteku `vjezba422.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.2.3

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Za provjeru stanja ulaznog pina koristite makronaredbu `get_pin`. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

U projektnom stablu otvorite datoteku `vjezba423.c`. Omogućite prevođenje samo datoteke `vjezba423.c`. Početni sadržaj datoteke `vjezba423.c` prikazan je programskim kodom 4.18.

Programski kod 4.18: Početni sadržaj datoteke `vjezba423.c`

```
#include "AVR lib/AVR_lib.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(DDRB ,PB3); // PB3 postavljen kao ulazni pin
    input_port(DDRB ,PB2); // PB2 postavljen kao ulazni pin
    input_port(DDRB ,PB1); // PB1 postavljen kao ulazni pin
    input_port(DDRB ,PB0); // PB0 postavljen kao ulazni pin

    set_port(PORTB ,PB3 ,1); // uključivanje priteznog otpornika na PB3
    set_port(PORTB ,PB2 ,1); // uključivanje priteznog otpornika na PB2
    set_port(PORTB ,PB1 ,1); // uključivanje priteznog otpornika na PB1
    set_port(PORTB ,PB0 ,1); // uključivanje priteznog otpornika na PB0
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    while (1)
    {
        if(get_pin(PINB ,PB3) == 0){ // ako je pin PB3 u niskom stanju
            set_port(PORTB , PB7 , 1); // uključi crvenu LED diodu
        }
        else{
            set_port(PORTB , PB7 , 0); // isključi crvenu LED diodu
        }

        // nastavite za ostala tipkala
    }

    return 0;
}
```

U ovoj vježbi cilj je isti kao i u prethodnoj vježbi. U programskom kodu 4.18 prikazano je tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PB3, PB2, PB1 i PB0 konfigurirani kao ulazni. Uključeni su pritezni

otpornici za pinove PB3, PB2, PB1 i PB0. Za navedene radnje korištene su makronaredbe iz zaglavlja `AVR_lib.h`.

U `while` petlji prikazanoj u programskom kodu 4.18 nalazi se niz naredbi koje će uključiti crvenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB3. 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 PB3 visoko stanje, makronaredba `get_pin(PINB,PB3)` vratit će vrijednost 1. Ako je na ulaznom pinu PB3 nisko stanje, makronaredba `get_pin(PINB,PB3)` vratit će vrijednost 0. Kada je tipkalo pritisnuto, stanje pina je nisko. Prema tome, logički uvjet (`get_pin(PINB,PB3)== 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 PB2, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB1 i bijelu LED diodu ako je pritisnuto tipkalo spojeno na pin PB0 korištenjem makronaredbe `get_pin`.

Prevedite datoteku `vjezba423.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



#### Vježba 4.2.4

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Za provjeru stanja ulaznog pina koristite funkciju `debounce`. Funkcija `debounce` može filtrirati smetnje koje se javljaju kada pritisnemo tipkalo. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

U projektnom stablu otvorite datoteku `vjezba424.c`. Omogućite prevođenje samo datoteke `vjezba424.c`. Početni sadržaj datoteke `vjezba424.c` prikazan je programskim kodom 4.19.

Programski kod 4.19: Početni sadržaj datoteke `vjezba424.c`

```
#include "AVR_lib/AVR_lib.h"
#include <avr/io.h>

void inicijalizacija(){
    // budući da znamo funkciju svih digitalnih pinova na portu B,
    // inicijalizacija se može izvesti na sljedeći način:
    DDRB = 0xF0;
    PORTB = 0x0F;
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    while (1)
    {
        // funkcija debounce filtrira smetnje
```

```

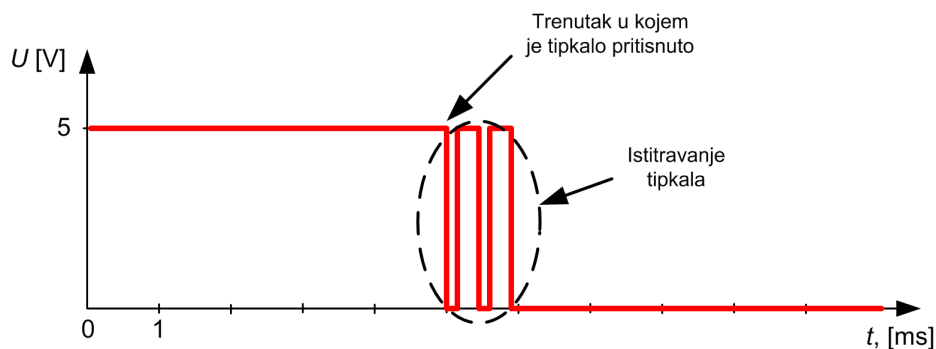
    if(debounce(&PINB, PB3, 0) == 0){
        set_port(PORTB, PB7, 1);
    }
    else{
        set_port(PORTB, PB7, 0);
    }

    // nastavite za ostala tipkala
}

return 0;
}

```

U trenutku kada pritisnemo tipkalo na razvojnom okruženju s mikrokontrolerom ATmega16, brid signala tipkala neće odmah pasti na nisku razinu, već dolazi do istitravanja (slika 4.5).



Slika 4.5: Istitravanje koje se javlja kod tipkala

Istitravanje može biti nezgodno ako npr. moramo brojati koliko puta je tipkalo pritisnuto. Pogledajmo signal tipkala na slici 4.5. 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 je uzastopni prijelaz iz visokog u nisko stanje zbog mehaničkih karakteristika tipkala.

Ovaj problem može se riješiti uvođenjem vremenskog okvira unutar kojeg ćemo pratiti koliko vremena je tipkalo bilo pritisnuto. U zaglavlju `AVR_lib.h` nalazi se deklaracija funkcije `debounce` koja rješava problem istitravanja tipkala.

Funkcija `debounce` prima tri argumenta:

- adresu registra `PINx` (uz registar koji kao prvi argument upisujete u funkciju `debounce` potrebno je staviti adresni operator `&`, npr. `&PINA`),
- poziciju ulaznog pina čije stanje želite ispitati, npr. `PA1`,
- stanje koje želite ispitati na poziciji ulaznog pina, npr. `1`.

Funkcija `debounce` kao povratnu vrijednost vraća stanje ulaznog pina. Uz funkciju `debounce` potrebno je definirati vremensku konstantu `DEBOUNCE_TIME` koja se nalazi u zaglavlju `AVR_lib.h`. Ova konstanta definira se u milisekundama, a predstavlja vremenski okvir unutar kojeg će se svake milisekunde ispitivati koje je trenutno stanje ulaznog pina. Pretpostavimo da smo pozvali funkciju `debounce(&PIND, PD4, 0)`. Ova će funkcija vratiti vrijednost `0` ako je ulazni pin `PD4` bio 90 % vremena definiranog vremenskom konstantom `DEBOUNCE_TIME` u niskom stanju. Na ovaj način kratkotrajni će poremećaji biti filtrirani zbog činjenice da ulazni pin `PD4` neće 90 % vremena definiranog vremenskom konstantom `DEBOUNCE_TIME` biti u niskom stanju. Vremenska konstanta `DEBOUNCE_TIME` podešava se iskustveno, a njena inicijalna vrijednost je 25 ms.



U programskom kodu 4.19 prikazano je tijelo funkcije `inicijalizacija()`. Pinovi PB7, PB6, PB5 i PB4 konfigurirani su kao izlazni pinovi, dok su pinovi PB3, PB2, PB1 i PB0 konfigurirani kao ulazni. Uključeni su pritezni otpornici za pinove PB3, PB2, PB1 i PB0.

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 PB3. Za provjeru stanja ulaznog pina korištena je funkcija `debounce`. Proširite niz naredbi `while` petlje tako da uključite žutu LED diodu ako je pritisnuto tipkalo spojeno na pin PB2, zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB1 i bijelu LED diodu ako je pritisnuto tipkalo spojeno na pin PB0 korištenjem funkcije `debounce`.

Prevedite datoteku `vjezba424.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Proizvoljno promijenite vremensku konstantu `DEBOUNCE_TIME` koja se nalazi u zaglavlju `AVR_lib.h`. Ponovno prevedite datoteku `vjezba424.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 4.2.5

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 promijeniti stanje crvene LED diode na padajući brid signala tipkala spojenog na pin PB3, žute LED diode na padajući brid signala tipkala spojenog na pin PB2, zelene LED diode na padajući brid signala tipkala spojenog na pin PB1 i bijele LED diode na padajući brid signala tipkala spojenog na pin PB0. Za provjeru stanja ulaznog pina koristite funkciju `debounce`. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

U projektnom stablu otvorite datoteku `vjezba425.c`. Omogućite prevođenje samo datoteke `vjezba425.c`. Početni sadržaj datoteke `vjezba425.c` prikazan je programskim kodom 4.20.

Programski kod 4.20: Početni sadržaj datoteke `vjezba425.c`

```
#include "AVR_lib/AVR_lib.h"
#include <avr/io.h>

void inicijalizacija(){

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

    input_port(DDRB,PB3); // PB3 postavljen kao ulazni pin

    set_port(PORTB,PB3,1); // uključen pritezni otpornik na PB3
}

int main(void){

    inicijalizacija(); // inicijalizacija mikrokontrolera

    int pinb3 = 1; // stara vrijednost ulaznog pina PB3

    while (1)
    {
        // na padajući brid pina PB3 promijeni stanje crvene LED diode
        if(pinb3 == 1 && debounce(&PINB, PB3, 0) != pinb3){
            TOGGLE_PORT(PORTB,PB7);
            pinb3 = 0;
        }
    }
}
```

```

    }

    if(get_pin(PINB,PB3))
        pinb3 = 1;

    // nastavite za ostala tipkala
}

return 0;
}

```

Često se u praktičnoj primjeni koriste gumbi na dodir (eng. *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 4.20 prikazano je tijelo funkcije `inicijalizacija()`. Konfigurirani su pinovi PB7 kao izlazni i PB3 kao ulazni pin. Napravite konfiguraciju ostalih pinova koji će se koristiti u ovoj vježbi.

U `while` petlji prikazanoj u programskom kodu 4.20 nalazi se niz naredbi koje će promijeniti stanje crvene LED diode ako se na pinu PB3 pojavi padajući brid signala. Padajući brid signala na pinu PB3 javit će se u trenutku kada pritisnemo tipkalo. Tada će stanje iz visokog prijeći u nisko.

U funkciju `main()` deklarirali smo cjelobrojnu varijablu `pinb3` i inicijalno je postavili u visoko stanje. Varijabla `pinb3` čuva prošlo stanje pina PB3. U `while` petlji nalazi se naredba `if(pinb3 == 1 && debounce(&PINB, PB3, 0) != pinb3)`. Objasnimo ovu naredbu. Kao uvjet `if` naredbe provjerava se je li varijabla `pinb3` u visokom stanju, što je zapravo provjera je li pin PB3 u prošlom prolazu kroz `while` petlju bio u visokom stanju. Istovremeno se provjerava i je li trenutno stanje pina PB3 jednako 0 i je li različito od prošlog stanja sačuvanog u varijabli `pinb3`. Ako su oba uvjeta zadovoljena, pojavio se padajući brid. U varijablu `pinb3` postavlja se novo stanje pina PB3 koje će se koristiti u sljedećem prolazu kroz `while` petlju kao prošlo stanje. Makronaredbom `TOGGLE_PORT` mijenjamo stanje crvene LED diode. U sljedećem prolazu kroz `while` petlju, budući da je `pinb3 = 0`, prvi uvjet u `if` naredbi neće biti zadovoljen te se promjena stanja crvene LED diode neće dogoditi. Kada otpustimo tipkalo, moramo osigurati ponovno detektiranje padajućeg brida. To smo osigurali naredbom `if(get_pin(PINB,PB3))pinb3 = 1;`.

Proširite niz naredbi `while` petlje tako da se stanje žute LED diode promijeni na padajući brid signala tipkala spojenog na pin PB2, zelene LED diode na padajući brid signala tipkala spojenog na pin PB1 i bijele LED diode na padajući brid signala tipkala spojenog na pin PB0. Ne zaboravite deklarirati cjelobrojne varijable `pinb0`, `pinb1` i `pinb2` koje će čuvati prošla stanja ulaznih pinova PB0, PB1 i PB2.

Prevedite datoteku `vjezba425.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba425.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

### 4.2.2 Zadaci - digitalni ulazi mikrokontrolera ATmega16

#### Zadatak 4.2.1

Napravite program koji će na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti crvenu i zelenu LED diodu ako je pritisnuto tipkalo spojeno na pin PB2. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

---

#### Zadatak 4.2.2

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

---

#### Zadatak 4.2.3

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Za provjeru stanja ulaznog pina koristite makronaredbu `get_pin`. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

---

#### Zadatak 4.2.4

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 uključiti bijela LED dioda ako je pritisnuto tipkalo spojeno na pin PB3, zelena LED dioda ako je pritisnuto tipkalo spojeno na pin PB2, žuta LED dioda ako je pritisnuto tipkalo spojeno na pin PB1 i crvena LED dioda ako je pritisnuto tipkalo spojeno na pin PB0. Za provjeru stanja ulaznog pina koristite funkciju `debounce`. Funkcija `debounce` može filtrirati smetnje koje se javljaju kada pritisnemo tipkalo. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

---

#### Zadatak 4.2.5

Napravite program kojim će se na razvojnom okruženju s mikrokontrolerom ATmega16 promijeniti stanje crvene LED diode na rastući brid signala tipkala spojenog na pin PB3, žute LED diode na rastući brid signala tipkala spojenog na pin PB2, zelene LED diode na rastući brid signala tipkala spojenog na pin PB1 i bijele LED diode na rastući brid signala tipkala spojenog na pin PB0. Za provjeru stanja ulaznog pina koristite funkciju `debounce`. Shema spajanja tipkala i LED dioda na mikrokontroler ATmega16 prikazana je na slici 4.4.

---

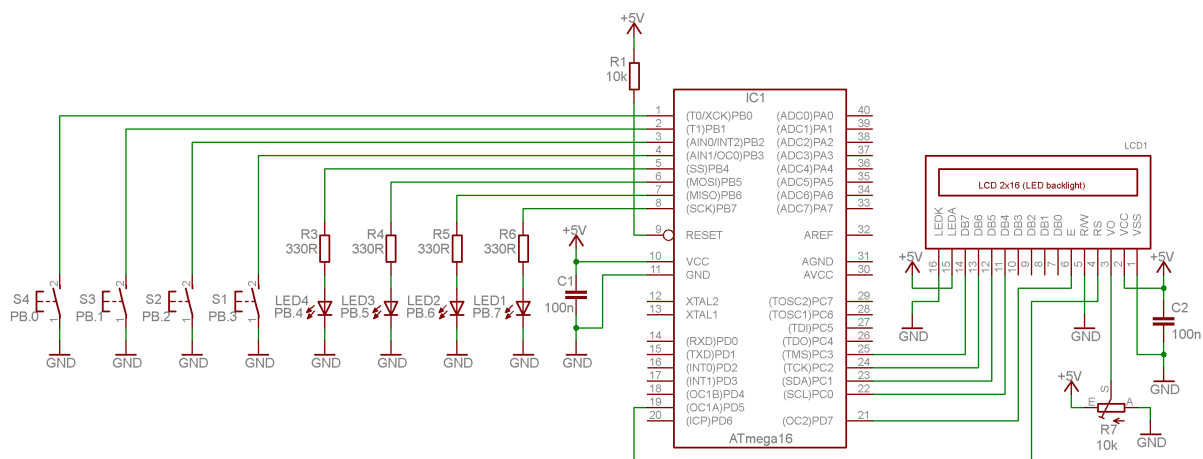


## Poglavlje 5

# 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 mikrokontrolerom ATmega16 prikazanom na slici 3.1 nalazi se LCD displej GDM1602E. On ima mogućnost prikaza dva retka 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 mikrokontroler koji se brine o dekodiranju podataka koji pristižu na njegove ulazne pinove. Znakovi koje LCD displej prikazuje nalaze se u njegovom 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 LED dioda, tipkala i LCD displeja na mikrokontroler ATmega16

Na slici 5.1 prikazana je shema spajanja LED dioda, tipkala i LCD displeja na mikrokontroler ATmega16. 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 mikrokontroler je potrebno spojiti na pinove LCD displeja DB4, DB5, DB6 i DB7.

Bilo koja četiri pina mikrokontrolera možemo spojiti na pinove DB4, DB5, DB6 i DB7. Na razvojnom okruženju s mikrokontrolerom ATmega16 4-bitna podatkovna sabirnica LCD displeja spojena je na pinove PC0, PC1, PC2 i PC3. Pinovi RS i E na LCD displeju upravljački su pinovi te su spojeni na pinove PD5 i PD7 (slika 5.1). Na LCD displej moguće je zapisivati podatke i čitati podatke. Na razvojnom okruženju s mikrokontrolerom ATmega16 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 DB4, DB5, DB6, DB7, RS i E mogu se spojiti na bilo koji dostupni digitalni pin mikrokontrolera što ovisi o zauzeću ostalih pinova. U većini slučajeva odluka o spajanju LCD displeja na digitalne pinove mikrokontrolera dolazi na kraju kada se na mikrokontroler spoje senzori i aktuatori koji koriste namjenske pinove mikrokontrolera.

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

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

Neposredno prije nego što počnemo ispisivati tekst na LCD displej pomoću mikrokontrolera ATmega16, potrebno je konfigurirati pinove mikrokontrolera 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*

```
// konfiguracija LCD displeja

// port za 4 bitnu komunikaciju - podatkovni bit 0 //D4
#define LCD_DATA0_PORT PORTC
// port za 4 bitnu komunikaciju - podatkovni bit 1 //D5
#define LCD_DATA1_PORT PORTC
// port za 4 bitnu komunikaciju - podatkovni bit 2 //D6
#define LCD_DATA2_PORT PORTC
// port za 4 bitnu komunikaciju - podatkovni bit 3 //D7
#define LCD_DATA3_PORT PORTC
#define LCD_DATA0_PIN PC0 // pin za 4-bitnu komunikaciju //D4
#define LCD_DATA1_PIN PC1 // pin za 4-bitnu komunikaciju //D5
#define LCD_DATA2_PIN PC2 // pin za 4-bitnu komunikaciju //D6
#define LCD_DATA3_PIN PC3 // pin za 4-bitnu komunikaciju //D7
#define LCD_RS_PORT PORTD // port za odabir registra //RS
#define LCD_RS_PIN PC5 // pin za odabir registra
#define LCD_E_PORT PORTD // port za odobrenje upisa //EN
#define LCD_E_PIN PC7 // pin za odobrenje upisa

#define LCD_LINES 2 // broj vidljivih redaka na LCD displeju
#define LCD_DISP_LENGTH 16 // broj vidljivih znakova u retku

// kraj konfiguracije LCD displeja
```

U programskom kodu 5.1 prikazana je konfiguracija pinova LCD displeja u zaglavlju *lcd.h*. Konfiguracija pinova mikrokontrolera 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 mikrokontrolera

i LCD displeja je 4-bitna. Podatkovna sabirnica konfigurira se na sljedeći način:

- `#define LCD_DATA0_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera na kojem je spojen pin LCD displeja DB4,
- `#define LCD_DATA1_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera na kojem je spojen pin LCD displeja DB5,
- `#define LCD_DATA2_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera na kojem je spojen pin LCD displeja DB6,
- `#define LCD_DATA3_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera na kojem je spojen pin LCD displeja DB7,
- `#define LCD_DATA0_PIN Pxi`, ( $x = A, B, C, D; i = 0,1,\dots,7$ ) - pozicija pina mikrokontrolera koji je spojen na pin LCD displeja DB4,
- `#define LCD_DATA1_PIN Pxi`, ( $x = A, B, C, D; i = 0,1,\dots,7$ ) - pozicija pina mikrokontrolera koji je spojen na pin LCD displeja DB5,
- `#define LCD_DATA2_PIN Pxi`, ( $x = A, B, C, D; i = 0,1,\dots,7$ ) - pozicija pina mikrokontrolera koji je spojen na pin LCD displeja DB6,
- `#define LCD_DATA3_PIN Pxi`, ( $x = A, B, C, D; i = 0,1,\dots,7$ ) - pozicija pina mikrokontrolera koji je spojen na pin LCD displeja DB7.

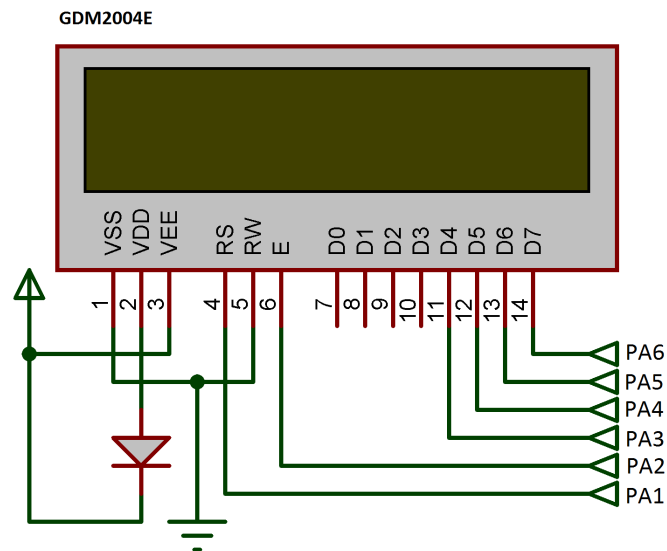
Upravljački pinovi konfiguriraju se na sljedeći način:

- `#define LCD_RS_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera 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 mikrokontrolera koji je spojen na pin LCD displeja RS,
- `#define LCD_E_PORT PORTx`, ( $x = A, B, C, D$ ) - port mikrokontrolera 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 mikrokontrolera 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 GDM2004 na mikrokontroler ATmega16.



Slika 5.2: Primjer LCD displeja GDM2004 spojenog na ATmega16

Ovaj LCD displej ima četiri vidljiva retka i 20 vidljivih znakova u jednom retku. Svi pinovi LCD displeja (podatkovni i upravljački) spojeni su na port A mikrokontrolera ATmega16 prema slici 5.2. Za ovako spojen LCD displej na mikrokontroler ATmega16 potrebno je napraviti konfiguracija pinova u zaglavlju `lcd.h`. Konfiguracija pinova mikrokontrolera 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`

```
// konfiguracija LCD displeja

// port za 4 bitnu komunikaciju - podatkovni bit 0 //D4
#define LCD_DATA0_PORT PORTA
// port za 4 bitnu komunikaciju - podatkovni bit 1 //D5
#define LCD_DATA1_PORT PORTA
// port za 4 bitnu komunikaciju - podatkovni bit 2 //D6
#define LCD_DATA2_PORT PORTA
// port za 4 bitnu komunikaciju - podatkovni bit 3 //D7
#define LCD_DATA3_PORT PORTA
#define LCD_DATA0_PIN PA3 // pin za 4-bitnu komunikaciju //D4
#define LCD_DATA1_PIN PA4 // pin za 4-bitnu komunikaciju //D5
#define LCD_DATA2_PIN PA5 // pin za 4-bitnu komunikaciju //D6
#define LCD_DATA3_PIN PA6 // pin za 4-bitnu komunikaciju //D7
#define LCD_RS_PORT PORTA // port za odabir registra //RS
#define LCD_RS_PIN PA1 // pin za odabir registra
#define LCD_E_PORT PORTA // port za odobrenje upisa //EN
#define LCD_E_PIN PA2 // pin za odobrenje upisa

#define LCD_LINES 4 // broj vidljivih redaka na LCD displeju
#define LCD_DISP_LENGTH 20 // broj vidljivih znakova u retku

// kraj konfiguracije LCD displeja
```





### Vježba 5.1.1

Napravite program u `while` petlji koji će na LCD displeju na početku prvog retka ispisati Vaše ime, a na početku drugog retka LCD displeja ispisati Vaše prezime bez diakritičkih znakova. Postavite kašnjenje u programu od dvije sekunde pa ispišite tekst `Visoka tehnicka skola u Bj` pravilno raspoređen u dva retka LCD displeja. Ponovno postavite kašnjenje u programu od dvije sekunde. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.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(0,0);
        lcd_print("Visoka tehnicka");
        lcd_gotoxy(1,3);
        lcd_print("skola u Bj");
        _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 je `#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 mikrokontrolera 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 mikrokontrolera uvijek ćemo pozivati u funkciji `inicijalizacija` u kojoj se nalazi i funkcija za inicijalizaciju LCD displeja `lcd_init`.

Prevedite datoteku `vjezba511.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite ispis imena i prezimena na sredinu LCD displeja, a tekst *Visoka tehnicka skola* u Bj poravnajte desno na LCD displeju.

Prevedite datoteku `vjezba511.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 5.1.2

Napravite program u `while` petlji koji će na LCD displeju na početku prvog retka ispisati proizvoljni cijeli broj tipa `int`, a na početku drugog retka ispisati proizvoljni realan broj tipa `float` na dva decimalna mjesta. Postavite kašnjenje u programu od dvije sekunde, a zatim na početku prvog retka ispišite proizvoljni cijeli broj tipa `int32_t`, a na početku drugog retka ispišite proizvoljni cijeli broj tipa `uint32_t`. Ponovno postavite kašnjenje u programu od dvije sekunde. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.h"
#include "LCD/lcd.h"
#include <avr/io.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: %f ", 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 6* (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 u programskom razvojnom okruženju *Atmel Studio 6* širine 16 bitova. U programskom razvojnom okruženju *Atmel Studio 6* postoje definirani tipovi podataka koji su izvedeni iz standardnih tipova podataka (tablica 5.1). Na primjer, `int8_t` cijeli je broj s predznakom širine osam 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 6*

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 proizvoljni 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 proizvoljni cijeli broj tipa `int32_t`, a na početku drugog retka ispisati proizvoljni 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite tipove podataka za ispis te ponovno prevedite datoteku `vjezba512.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba512.c` 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 mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.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: "); // popravite argument funkcije lcd_print

        _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 displejem.

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.c` u strojni kod i snimite ga na mikrokontroler ATmega16.

Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba513.c` 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 LCD displeja ispisivati sinus tog kuta na tri decimalna mjesta. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.h"
#include "LCD/lcd.h"
#include <avr/io.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 potrebno kut u stupnjevima pretvoriti u radijane. Varijablu `kut` deklarirali smo kao cjelobrojnu varijablu bez predznaka širine osam bitova. Ispod funkcije `lcd_home` u programskom kodu 5.6 upišite sljedeće tri naredbe:

- `lcd_print("Kut:%u%c", kut, 178);` - ispis kuta od 0 do 180 °. U argumentu funkcije `lcd_print` pojavljuje se varijabla `kut` i broj 178. Broj 178 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  $\frac{\pi \cdot \text{kut}}{180}$ .

Prevedite datoteku `vjezba514.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite preciznost ispisa na više decimalnih mjesta (ne više od sedam). Ograničite promjenu kuta od 30 do 90 °. Prevedite datoteku `vjezba514.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 5.1.5

Napravite program koji će na LCD displeju prikazivati sekunde, minute i sate od trenutka uključivanja mikrokontrolera. Svake sekunde promijenite stanje crvene LED diode. Kada se pritisne tipkalo spojeno na pin PB0, neka se isključi crvena LED dioda te sekunde, minute i sati neka se postave na nulu. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.h"
#include "LCD/lcd.h"
#include <avr/io.h>

void inicijalizacija(){

    output_port(DDRB,PB7); //pin PB7 postavljen kao izlaz
    input_port(DDRB,PB0); //pin PB0 postavljen kao ulaz
    set_port(PORTB,PB0,1); // uključen pritezni otpornik na PBO

    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(debounce(&PINB,PB0,0) == 0){
        sec = 0;
        min = 0;
        sat = 0;
        set_port(PORTB,PB7,0);
    }

}

return 0;
}

```

Cilj je ove vježbe napraviti prikaz vremena rada mikrokontrolera u satima, minutama i sekundama. U vježbi se koriste crvena LED dioda i tipkalo spojeno na pin PB0 pa je u programskom kodu 5.7 u funkciji `inicijalizacija()` napravljena konfiguracija pinova. Na LCD displeju potrebno je prikazivati sekunde, minute i sate. Proračun sekunda, 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 crvene 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 PB0 provjeravamo funkcijom `debounce`. Ako funkcija vrati vrijednost 0 (slučaj kada je tipkalo pritisnuto), tada će se isključiti crvena LED dioda, a varijable `sec`, `min` i `sat` postaviti u vrijednost nula.

Prevedite datoteku `vjezba515.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba515.c` 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.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

Programski kod 5.8: Početni sadržaj datoteke vjezba516.c

```

#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.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 8 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.c` 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

```

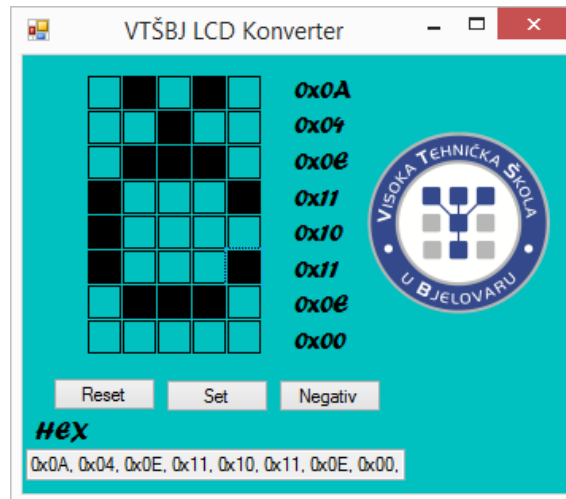
const uint8_t symbol[8][8] = { //definirajte 8 znakova

    /* 0x00 */ 0x0A, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00, //č
    /* 0x01 */ 0x02, 0x04, 0x0E, 0x11, 0x10, 0x11, 0x0E, 0x00, //ć
    /* 0x02 */ 0x02, 0x07, 0x02, 0x0E, 0x12, 0x12, 0x0E, 0x00, //đ
    /* 0x03 */ 0x0A, 0x04, 0x0E, 0x10, 0x0E, 0x01, 0x1E, 0x00, //š
    /* 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
};

```

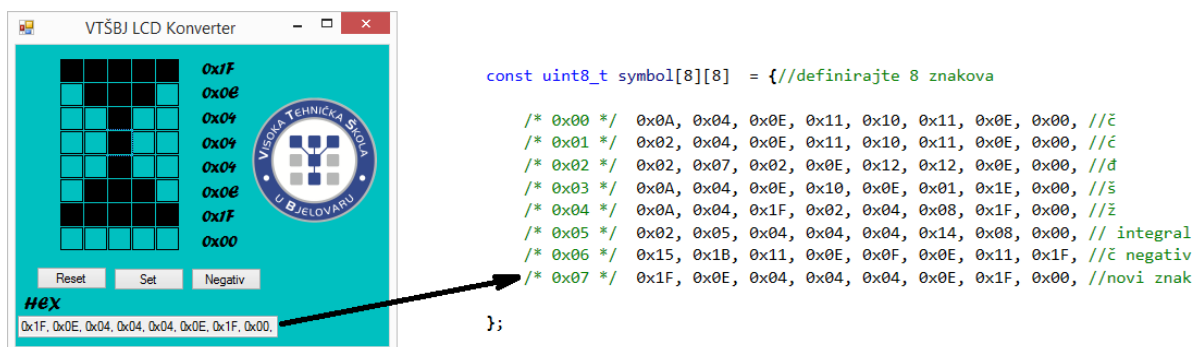
U programskom kodu 5.9 blok komentarima označene su memorijske lokacije definiranih znakova. Svaki znak je jedan redak u dvodimenzionalnom polju. Ako želite definirati vlastiti znak, potrebno je koristiti aplikaciju LCD konverter dostupnu na stranici [www.vtstbj.hr/mikroracunala](http://www.vtstbj.hr/mikroracunala). Skinite aplikaciju i pokrenite je. Aplikacija LCD konverter prikazana je na slici 5.3.





Slika 5.3: Aplikacija LCD konverter

Matrica za izradu znakova na aplikaciji LCD konverter dimenzijom odgovara znaku na LCD displeju. Aplikacija LCD konverter stvara osam bajtova podataka u heksadecimalnom zapisu koje je potrebno kopirati u dvodimenzionalno polje koje se nalazi u tijelu funkcije `lcd_define_char` (slika 5.4) 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 5.4: Definiranje novog znaka

Definirane znakove možemo ispisivati na LCD displej pomoću dvije funkcije:

- `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 prosleđuju na LCD displej. U programskom jeziku C svaki je niz znakova (tekst (eng. *string*)) „zaključan” s tzv. `null` znakom čiji je ASCII kod jednak `0x00`. Iz ovog 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 kojeg 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("Mikrora%cunala", 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Definirajte osam vlastitih znakova, pohranite ih u dvodimenzionalno polje u funkciji `lcd_define_char` te ponovno prevedite datoteku `vjezba516.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



### Vježba 5.1.7

Napravite program koji će na LCD displeju svake dvije sekunde izmjenjivati tekstove **Visoka tehnička škola u Bjelovaru** i **Stručni studij Mehatronika**. Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Za definiranje znakova potrebno je koristiti aplikaciju LCD konverter dostupnu na stranici [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

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

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.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 **Visoka tehnička škola u Bjelovaru** i **Stručni studij Mehatronika** 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 **Visoka tehnička škola u Bjelovaru** i **Stručni studij**

Mehatronika.

Prevedite datoteku `vjezba517.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Proizvoljno ispišite tekst koji sadrži dijakritičke znakove na LCD displej te ponovno prevedite datoteku `vjezba517.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba517.c` 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.vtsbj.hr/mikroracunala`. Tekst je potrebno svaku sekundu pomaknuti za jedno mjesto ulijevo. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

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

Programski kod 5.11: Početni sadržaj datoteke `vjezba518.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.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.vtsbj.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.vtsbj.hr/mikroracunala` imaju više od 16 znakova što izlazi iz vidnog područja LCD displeja GMD1602E. Svaki redak može pohraniti do 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba518.c` i onemogućite prevodenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 5.2 Zadaci - LCD displej

### Zadatak 5.2.1

Napravite program u `while` petlji koji će na LCD displeju u prvom retku desno poravnato ispisati Vaše ime, a u drugom retku desno poravnato ispisati Vaše prezime bez dijakritičkih znakova. Postavite kašnjenje u programu od tri sekunde pa ispišite tekst `Strucni studij Mehatronika` pravilno raspoređen u dva retka LCD displeja. Ponovno postavite kašnjenje u programu od tri sekunde. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

### Zadatak 5.2.2

Napravite program u `while` petlji koji će na LCD displeju na početku prvog retka ispisati proizvoljni cijeli broj tipa `int8_t`, a na početku drugog retka ispisati proizvoljni realan broj tipa `double` na četiri decimalna mjesta. Postavite kašnjenje u programu od dvije sekunde, a zatim na početku prvog retka ispisati proizvoljni cijeli broj tipa `int16_t`, a na početku drugog retka ispisati proizvoljni cijeli broj tipa `uint16_t`. Ponovno postavite kašnjenje u programu od dvije sekunde. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

### Zadatak 5.2.3

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 s `printf` formatom `%u`. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

### Zadatak 5.2.4

Napravite program koji će u prvom retku LCD displeja svake sekunde ispisivati kut od 0 do 180 °, a u drugom retku LCD displeja ispisivati kosinus tog kuta na dva decimalna mjesta. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

### Zadatak 5.2.5

Napravite program koji će na LCD displeju prikazivati desetinke sekunde, sekunde, minute i sate od trenutka uključanja mikrokontrolera. Svake sekunde promijenite stanje zelene LED diode. Ukoliko se pritisne tipkalo spojeno na pin PB4, neka se isključi zelena LED dioda te desetinke sekunde, sekunde, minute i sati neka se postave na nulu. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

** Zadatak 5.2.6**

Napravite program koji će na LCD displeju ispisivati proizvoljne znakove definirane pomoću aplikacije LCD konverter dostupne na mrežnoj stranici [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Znakovi se na LCD displeju definiraju pozivom funkcije `lcd_define_char`. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

** Zadatak 5.2.7**

Napravite program koji će na LCD displeju svake dvije sekunde izmjenjivati tekstove MIKRORAČUNALO ATmega16 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.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala). Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

** Zadatak 5.2.8**

Napravite program koji će na LCD displeju u gornjem retku ispisati tekst *Visoka tehnička škola u Bjelovaru*, a u donjem retku *Stručni studij Mehatronika*. Tekst je potrebno svake sekunde pomaknuti za jedno mjesto ulijevo. Shema spajanja LCD displeja na razvojno okruženje s mikrokontrolerom ATmega16 prikazana je na slici 5.1.

---

## Poglavlje 6

# EEPROM memorija

EEPROM memorija (eng. *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 mikrokontroleru ATmega16 nalazi se EEPROM memorija veličine 512 bajtova. Radni vijek EEPROM memorije je do najviše 100000 ciklusa pisanja i čitanja. EEPROM memorija u mikrokontroleru ATmega16 odvojena je memorija koja nije u sklopu arhitekture mikrokontrolera ATmega16 te se zbog toga za adresiranje podataka koristi poseban EEPROM registar [1].

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.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 `mikroracunala.atsln` 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 `*.c`.

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 PB0. Nad navedenim brojevima potrebno je izvršiti proizvoljne jednostavne matematičke operacije. Brojeve je potrebno prikazivati na LCD displeju. Prilikom uključivanja mikrokontrolera brojeve tipa `int8_t`, `int16_t`, `int32_t` i `float` inicijalizirajte čitanjem iz EEPROM memorije.

---

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

Programski kod 6.1: Početni sadržaj datoteke vjezba611.c

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include <avr/eeprom.h>

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

void inicijalizacija(){

    input_port(DDRB,PB0);
    set_port(PORTB,PB0,1);

    cijeli8_t = eeprom_read_byte(0);
    cijeli16_t = eeprom_read_word(1);
    cijeli32_t = eeprom_read_dword(3);
    realan = eeprom_read_float(7);

    lcd_init();
}

int main(void){

    inicijalizacija();

    uint8_t pinb0 = 1;

    while(1)
    {

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

        // ispitivanje padajućeg brida na PB0
        if(pinb0 == 1 && debounce(&PINB, PB0, 0) != pinb0){
            eeprom_write_byte(0,cijeli8_t);
            eeprom_write_word(1,cijeli16_t);
            eeprom_write_dword(3,cijeli32_t);
            eeprom_write_float(7,realan);
            pinb0 = 0;
        }

        if(get_pin(PINB,PB0))
            pinb0 = 1;

        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 makronaredbe:

- `eeeprom_read_byte(addr)` - makronaredba koja čita cijeli broj širine osam bitova s adrese `addr`.
- `eeeprom_read_word(addr)` - makronaredba koja čita cijeli broj širine 16 bitova s adrese `addr`.
- `eeeprom_read_dword(addr)` - makronaredba koja čita cijeli broj širine 32 bita s adrese `addr`.
- `eeeprom_read_float(addr)` - makronaredba koja čita realan broj širine 32 bita s adrese `addr`.
- `eeeprom_write_byte(addr, data)` - makronaredba koja zapisuje cijeli broj `data` širine osam bitova na adresu `addr`.
- `eeeprom_write_word(addr, data)` - makronaredba koja zapisuje cijeli broj `data` širine 16 bitova na adresu `addr`.
- `eeeprom_write_dword(addr, data)` - makronaredba koja zapisuje cijeli broj `data` širine 32 bita na adresu `addr`.
- `eeeprom_write_float(addr, data)` - makronaredba koja zapisuje realan broj `data` širine osam bitova na adresu `addr`.

Navedene makronaredbe definirane su u standardnom zaglavlju `eeeprom.h`. Zaglavlje `eeeprom.h` u programski je kod 6.1 uključeno naredbom `#include <avr/eeeprom.h>`.

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.c`. U funkciji `inicijalizacija()` iz EEPROM memorije čitamo podatke.

Na padajući brid signala tipkala spojenog na pin PB0 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 kojeg čitamo, odnosno zapisujemo. Jedna memorijska lokacija EEPROM memorije širine je 8 bitova (jedan 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 je `0x03`. Ako na memorijsku lokaciju `0x03` zapisujemo cijeli broj širine 32 bitova, tada taj broj zauzima memorijske lokacije `0x03`, `0x04`, `0x05` i `0x06` (slika 6.1). Sljedeća slobodna lokacija za zapisivanje podataka je `0x07`. Podatke moramo čitati s onih memorijskih lokacija na koje smo zapisali podatke. Pri tome je potrebno voditi brigu o veličini podatka kojeg 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 tako da u jednom trenutku pritisnete tipkalo spojeno na pin PB0. Nakon toga isključite napajanje mikrokontrolera ATmega16 te ga ponovno uključite. Jesu li podaci ostali sačuvani bez obzira na nestanak napajanja?

Zatvorite datoteku `vjezba611.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 6.2 Zadaci - EEPROM memorija

### Zadatak 6.2.1

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 PB3. Nad navedenim brojevima potrebno je izvršiti proizvoljne jednostavne matematičke operacije. Brojeve je potrebno prikazivati na LCD displeju. Prilikom uključivanja mikrokontrolera brojeve tipa `uint8_t`, `uint16_t`, `int8_t`, `int16_t`, `uint32_t` i `double` inicijalizirajte čitanjem iz EEPROM memorije.

## Poglavlje 7

# Analogno-digitalna pretvorba

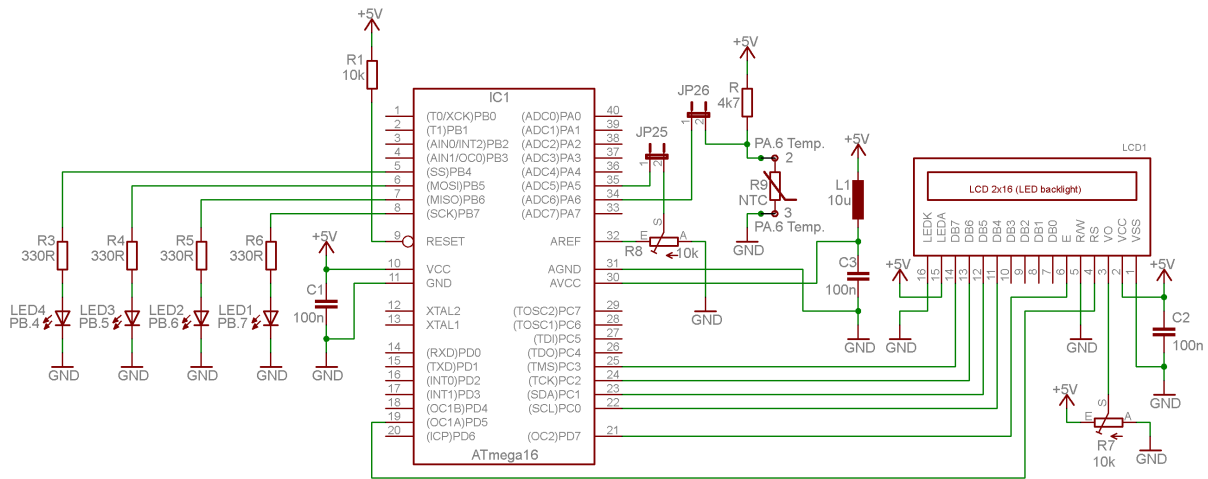
Analogno-digitalna pretvorba koristi se za prezentaciju analognih veličina s analognih senzora u mikrokontroleru. Analogni senzori mjerni su uređaji koji najčešće neelektrične veličine pretvaraju u električne (npr. temperaturu u napon, tlak u napon, vlagu u napon i drugo). Izlazni signal s analognih senzora često je napon u rasponu 0 - 5 V ili 0 - 10 V. Taj je napon potrebno mjeriti i prezentirati u mikrokontroleru. Podatak o mjerenoj veličini koristi se pri upravljanju i nadzoru sustava. Mikrokontroleri 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

Mikrokontroler ATmega16 ima osam analognih ulaza na portu A. Kao što možemo primijetiti, pinovi na portu A višenamjenski su jer osim što se mogu koristiti kao digitalni, mogu se koristiti i kao analogni. Analogno-digitalni pretvornik u mikrokontroleru ATmega16 pretvara analognu vrijednost napona u digitalnu riječ širine 10 bitova metodom sukcesivne aproksimacije [3]. Brzina analogno-digitalne pretvorbe je od 13 do 260  $\mu$ s. Shema spajanja LED dioda, LCD displeja i analognih senzora na mikrokontroler ATmega16 prikazana je na slici 7.1. Analogni senzori spojeni na mikrokontroler ATmega16 su potenciometar i NTC otpornik. Potenciometar je preko kratkospojnika JP25 spojen na pin PA5, a NTC otpornik je preko kratkospojnika JP26 spojen na pin PA6.



Slika 7.1: Shema spajanja LED dioda, LCD displeja i analognih senzora na mikrokontroler ATmega16

Analogno-digitalni pretvornik ima odvojeno napajanje od mikrokontrolera. Napajanje za njega dovodi se na pinove mikrokontrolera AGND i AVCC preko LC filtra (slika 7.1). Zavojnica LC filtra uklanja propade struje napajanja, a kondenzator LC filtra uklanja propade napona napajanja. Vrijednost induktiviteta zavojnice LC filtra je  $10 \mu\text{H}$ , a vrijednost kapaciteta kondenzatora LC filtra je  $100 \text{ nF}$ . Razlika napajanja na AVCC pinu i VCC pinu (napajanje mikrokontrolera) ne smije biti veća od  $\pm 0,3 \text{ V}$ . Na razvojnom okruženja sa slike 3.1 napajanje mikrokontrolera i analogno-digitalnog pretvornika je isto i iznosi  $5 \text{ V}$ .

Analogno-digitalni pretvornik za pretvaranje napona u digitalnu riječ koristi referentni naponski izvor napona  $V_{REF}$ . Prema tome, analogna vrijednost napona u rasponu  $[0, V_{REF} \cdot (1 - 2^{-10})]$  V pretvara se u digitalnu riječ u rasponu  $[0, 1023]$ . Za referentni naponski izvor  $V_{REF}$  može biti izabran:

- napon napajanja analogno-digitalnog pretvornika AVCC,
- unutarnji napon  $2,56 \text{ V}$ ,
- vanjski napon na AREF pinu mikrokontrolera ATmega16.

Ako je referentni naponski izvor izabran kao AVCC ili unutarnji  $2,56 \text{ V}$ , tada se napon referentnog naponskog izvora prosljeđuje na AREF pin mikrokontrolera ATmega16. Referentni naponski izvor odabire se u registru **ADMUX** prema tablici 83 u literaturi [1].

Rezultat analogno-digitalne pretvorbe na pinu  $PA_i$  prikazan je relacijom 7.1:

$$ADC_i = \frac{V_{PA_i} \cdot 1024}{V_{REF}} \quad (7.1)$$

gdje je:

- $ADC_i$  - cjelobrojni rezultat analogno-digitalne pretvorbe na pinu  $PA_i$ ,
- $V_{PA_i}$  - napon na pinu  $PA_i$ ,
- $V_{REF}$  - referentni naponski izvor.

Pretpostavimo da napon napajanja analogno-digitalnog pretvornika iznosi  $5 \text{ V}$  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 PA2 iznosi 2,14 V ( $V_{PAi} = 2,14$  V). Prema relaciji (7.1), cjelobrojni rezultat analogno-digitalne pretvorbe na pinu PA2 iznosi:

$$ADC2 = \frac{V_{PA2} \cdot 1024}{5} = 438. \quad (7.2)$$

Analogni sensor mjeri neku fizikalnu veličinu i na svom izlazu daje analognu vrijednost napona kojeg mjeri pin PAi. Napon na pinu PAi na temelju rezultata analogno-digitalne pretvorbe na pinu PAi možemo dobiti pomoću relacije 7.3:

$$V_{PAi} = \frac{ADCi \cdot V_{REF}}{1024}. \quad (7.3)$$

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 PA2 iznosi 765 ( $ADC2 = 765$ ). Prema relaciji (7.3), napon analognog senzora spojenog na pin PA2 iznosi:

$$V_{PA2} = \frac{ADC2 \cdot 5 \text{ V}}{1024} = 3.735 \text{ V}. \quad (7.4)$$

Kada znamo napon  $V_{PAi}$ , vrijednost fizikalne veličine možemo izračunati na temelju tehničkih specifikacija analognog senzora. Frekvencija rada analogno-digitalne pretvorbe odabire se u registru `ADCSRA` prema tablici 85 u literaturi [1].

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.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 `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 `*.c`.

U datoteci s vježbama nalaze se i rješenja vježbi koje 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      0x00    // AREF pin
#define ADC_REFERENCE_AVCC     0x01    // AVCC pin
#define ADC_REFERENCE_RSVD     0x02    // Rezervirana
#define ADC_REFERENCE_256V    0x03    // Unutarnjih 2,56V

// 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 mikrokontrolera. Maske se

dodjeljuju konstanti `ADC_PRESCALE`. Što je veći djelitelj frekvencije, frekvencija rada analogno-digitalne pretvorbe je manja, a rezultat analogno-digitalne pretvorbe je 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      0x00    // F_CPU/2
#define ADC_PRESCALE_DIV4      0x02    // F_CPU/4
#define ADC_PRESCALE_DIV8      0x03    // F_CPU/8
#define ADC_PRESCALE_DIV16     0x04    // F_CPU/16
#define ADC_PRESCALE_DIV32     0x05    // F_CPU/32
#define ADC_PRESCALE_DIV64     0x06    // F_CPU/64
#define ADC_PRESCALE_DIV128    0x07    // F_CPU/128

// odabir djelitelja frkevcnije
#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 PA5, a na početku drugog retka ispisati napon koji se trenutno mjeri na pinu PA5. Analogno-digitalnu pretvorbu provodite jednom u sekundi. Na pinu PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je 10 kΩ.

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

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

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija(){

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

int main(void){

    inicijalizacija();

    uint16_t ADC5; // rezultat AD pretvorbe
    float VPA5; // napon na pinu PA5
    const float VREF = 5.0; // AVCC

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

```

    ADC5 = adc_read_10bit(5); // rezultat AD pretvorbe
    VPA5 = ADC5 * VREF / 1024.0; // napon na pinu PA5

    lcd_print("ADC5 = %d", ADC5);
    lcd_print("\nVPA5 = %.2fV", VPA5);
    _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 je `#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`.
- `adc_read_10bit(uint8_t PAi)` - funkcija koja kao argument prima pin `PAi` na kojem želite napraviti analogno-digitalnu pretvorbu. Povratna vrijednost funkcije `adc_read_10bit` je vrijednost analogno-digitalne pretvorbe na pinu `PAi`. Povratna vrijednost je širine 10 bitova.

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

- `uint16_t ADC5` - varijabla koja se koristi za spremanje rezultata analogno-digitalne pretvorbe,
- `float VPA5` - napon na pinu `PA5`,
- `const float VREF` - referentni iznos napona `AVCC = 5 V`.

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

Funkcija `adc_read_10bit(5)` vraća rezultat analogno-digitalne pretvorbe na pinu `PA5`. Vrijednost napona na pinu `PA5` 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 `PA5`.

Prevedite datoteku `vjezba711.c` u strojni kod i snimite ga na mikrokontroler `ATmega16`. Testirajte program na razvojnom okruženju s mikrokontrolerom `ATmega16`. Što je sve potrebno napraviti ako želimo da referentni iznos napona bude unutarnjih 2,56 V?

Zatvorite datoteku `vjezba711.c` 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 PA5 uzetih u vremenu od jedne sekunde. Na početku drugog retka LCD displeja ispišite srednju vrijednost napona na pinu PA5. Na pinu PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je 10 k $\Omega$ .

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

Programski kod 7.4: Početni sadržaj datoteke vjezba712.c

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"

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

int main(void){
    inicijalizacija();

    uint16_t ADC5 ; // rezultat AD pretvorbe
    float VPA5; // napon na pinu PA5
    const float VREF = 5.0; // AVCC

    while (1)
    {
        ADC5 = 0;

        for(uint8_t i = 0; i < 10; i++)
        {
            // zbroj deset mjerenja u jednoj sekundi
        }

        ADC5 = ADC5 / 10;

        // napon na pinu PA5

        lcd_clrscr();
        lcd_home();

        lcd_print("ADC5 = %d", ADC5);
        lcd_print("\nVPA5 = %.2fV", VPA5);

    }

    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 zaglavlje u kojem su definirane sve funkcije za rad s analogno-digitalnom pretvorbom.

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 PA5 u jednoj sekundi. Prema tome, u jednoj iteraciji `for` petlje potrebno je ostvariti kašnjenje od 100 ms. Početna vrijednost varijable `ADC5` u koju će se sumirati 10 mjerenja je nula.

U `for` petlju upišite naredbu `ADC5 = ADC5 + adc_read_10bit(5)`; te naredbu koja će u `for` petlji ostvariti kašnjenje od 100 ms. Nakon što `for` petlja izvrši 10 iteracija, u varijabli `ADC5` nalazi se suma od 10 mjerenja u digitalnom zapisu. Srednju vrijednost 10 mjerenja dobit ćemo tako da vrijednost varijable `ADC5` podijelimo s 10. Prema relaciji (7.3) izračunajte srednju vrijednost napona na pinu PA5 ispod naredbe `ADC5 = ADC5 / 10`; . Vrijednosti varijable `ADC5` i srednje vrijednosti napona ispisuju se na LCD displej.

Prevedite datoteku `vjezba712.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite broj mjerenih uzoraka u jednoj sekundi te ponovno prevedite datoteku `vjezba712.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba712.c` 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:

$$y[k] = 0.2y[k-1] + 0.8ADC5[k]$$

gdje je:

- $y[k]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k$ ,
- $y[k-1]$  - filtrirana vrijednost analogno-digitalne pretvorbe u koraku  $k-1$ ,
- $ADC5[k]$  - rezultat analogno-digitalne pretvorbe na pinu PA5 u koraku  $k$ .

Početna vrijednost digitalnog filtra prvog reda je  $y[-1] = 0$ . 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 PA5. Dodatno, napravite signalizaciju LED diodama na sljedeći način:

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

Na pinu PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je 10 k $\Omega$ .

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

Programski kod 7.5: Početni sadržaj datoteke vjezba713.c

```
#include "AVR_lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija(){

    // inicijalizacija izlaza

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

int main(void){

    inicijalizacija();

    uint16_t ADC5; // rezultat AD pretvorbe
    float VPA5; // napon na pinu PA5
    const float VREF = 5.0; // AVCC

    float yk; // y[k]
    float yk_1 = 0; // y[k-1]

    while(1)
    {
        ADC5 = adc_read_10bit(5);

        yk = 0.2 * yk_1 + 0.8 * ADC5;
        VPA5 = yk * VREF / 1023.0;

        // ispišite vrijednosti yk i VPA5

        _delay_ms(100);
        yk_1 = yk;

        if(VPA5 < 1.25){

            set_port(PORTB, PB4, 1);
            set_port(PORTB, PB5, 0);
            set_port(PORTB, PB6, 0);
            set_port(PORTB, PB7, 0);
        }

        if(VPA5 >= 1.25 && VPA5 < 2.5){

            set_port(PORTB, PB4, 1);
            set_port(PORTB, PB5, 1);
            set_port(PORTB, PB6, 0);
            set_port(PORTB, PB7, 0);
        }

        // nastavite za napon veći i jednak 2,5 V

    }
    return 0;
}
```

Digitalni filter prvog reda može filtrirati poremećaje koji se pojavljuju na izlazu mjernog senzora spojenog na analogni ulaz mikrokontrolera. 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 PA5. Rezultat pretvorbe sprema se u varijablu `ADC5`. 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 izračunajte pomoću relacije (7.3) i spremite ga u varijablu `VPA5`. Na LCD displej ispišite filtriranu vrijednost analogno-digitalne pretvorbe  $y[k]$  i napon `VPA5`.

Ostvarite kašnjenje programa od 100 ms. Nakon ovog kašnjenja korak  $k$  se povećao za jedan te vrijednost  $y[k]$  postaje  $y[k-1]$ . U programskom kodu 7.5 to znači da varijabla `yk_1` poprima prethodnu vrijednost varijable `yk`.

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.

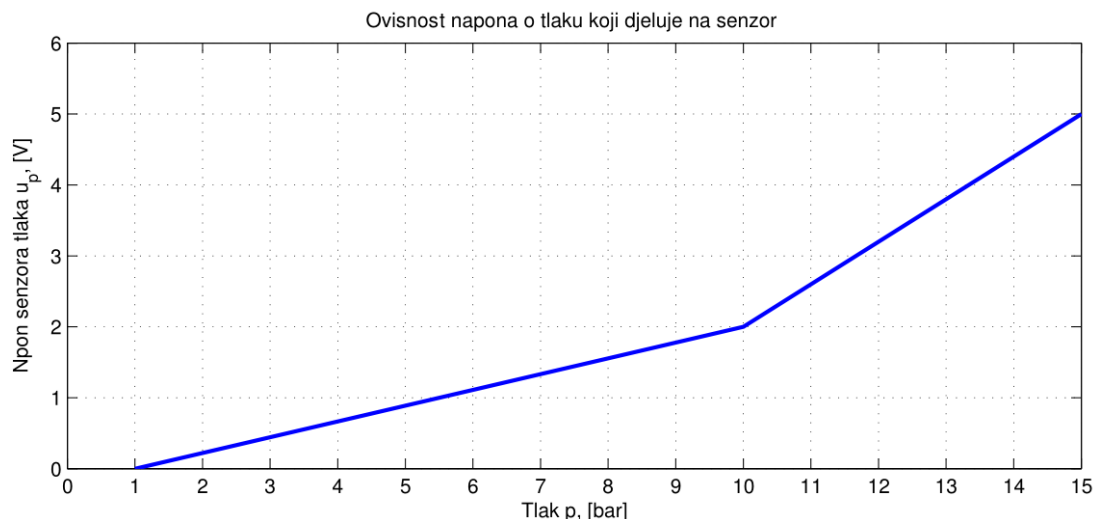
Prevedite datoteku `vjezba713.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

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



#### Vježba 7.1.4

Potenciometrom spojenim na pin PA5 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 LCD displeja ispisati tlak koji se trenutno mjeri senzorom tlaka. Na pinu PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je 10 k $\Omega$ .

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

Programski kod 7.6: Početni sadržaj datoteke vjezba714.c

```

#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija(){

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

int main(void){

    inicijalizacija();

    uint16_t ADC5; // rezultat AD pretvorbe
    float VPA5; // napon na pinu PA5
    const float VREF = 5.0; // AVCC
    float u, p; // napon i tlak

    while(1){

        // odredite ADC5 i VPA5

        u = VPA5;

        if (u <= 2.0)
        {
            p = 9 * u / 2 + 1;
        }
        else
        {
            p = 5 * u / 3 + 20.0 / 3;
        }

        // ispišite napon i tlak na tri decimalna mjesta
    }

    return 0;
}

```

Ovisnost napona o tlaku koji djeluje na senzor prikazan 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 1. 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 ono što nas zanima je tlak. Napon je samo posredna veličina preko koje analogno-digitalnom pretvorbom izračunavamo tlak. U programskom kodu 7.6 odredite rezultat analogno-digitalne pretvorbe na pinu PA5 (varijabla ADC5) pozivom funkcije `adc_read_10bit`. Odredite napon na pinu PA5 (varijabla VPA5) prema relaciji 7.1.

U uvjetnom `if` bloku provjeravamo je li napon na PA5 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba714.c` 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  $T$ , [Ω],
- $R_N = 10 \text{ k}\Omega$  - otpor NTC otpornika na nazivnoj temperaturi  $T_N$ , [Ω],

- $T$  - temperatura okoline NTC otpornika, [K],
- $T_N = 25\text{ }^\circ\text{C} = 298,15\text{ K}$  - nazivna temperatura okoline NTC otpornika, [K],
- $B = 3450\text{ K}$  - konstanta ovisna o materijalu NTC otpornika, [K].

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

- ako je temperatura manja od  $25\text{ }^\circ\text{C}$ , neka je uključena samo bijela LED dioda,
- ako je temperatura veća i jednaka  $25\text{ }^\circ\text{C}$  i manja od  $27,5\text{ }^\circ\text{C}$ , neka su uključene bijela i zelena LED dioda,
- ako je temperatura veća i jednaka  $27,5\text{ }^\circ\text{C}$  i manja od  $30\text{ }^\circ\text{C}$ , neka su uključene bijela, zelena i žuta LED dioda,
- ako je temperatura veća i jednaka  $30\text{ }^\circ\text{C}$ , 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 6.1. Pad napona na NTC otporniku mjeri se pomoću pina PA6.

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

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

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"

void inicijalizacija(){

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

int main(void){

    inicijalizacija();

    uint16_t ADC6; // rezultat AD pretvorbe
    float T;

    while(1)
    {
        ADC6 = adc_read_10bit(6);

        // relacija za temperaturu

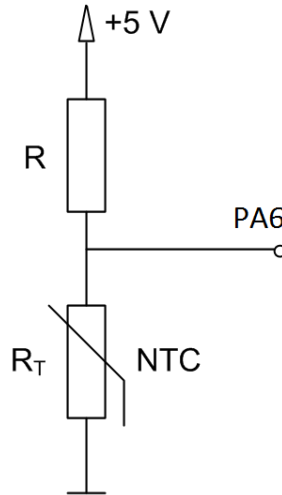
        lcd_clrscr();
        lcd_home();
        lcd_print("T = %.2f°C", T, 178);

        // kod za signalizaciju

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

Svojstvo NTC (eng. *Negative Temperature Coefficient*) otpornika je da se porastom temperature u okolini NTC otpornika smanjuje njegov otpor, a smanjenjem temperature povećava se njegov otpor. 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  $V_{PA6}$  na pinu PA6 prema slici 7.3 jednak je:

$$V_{PA6} = \frac{R_T}{R_T + R} 5 \quad (7.12)$$

Iz relacije (7.11) potrebno je izraziti temperaturu  $T$  preko otpora NTC otpornika. Ako relaciju (7.11) logaritmiramo s 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 je 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):

$$\begin{aligned} U_{PA6} &= \frac{R_T}{R_T + R} 5 / \cdot (R_T + R) \Rightarrow (R_T + R) U_{PA6} = R_T 5 \Rightarrow \\ R_T &= \frac{U_{PA6}}{5 - U_{PA6}} R. \end{aligned} \quad (7.15)$$

Napon  $U_{PA6}$  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:

$$R_T = \frac{ADC6}{1024 - ADC6} R. \quad (7.16)$$

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

$$T = \frac{B}{\ln\left(\frac{ADC6}{1024 - ADC6} \frac{R}{R_N}\right) + \frac{B}{T_N}}. \quad (7.17)$$

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

$$T = \frac{B}{\ln\left(\frac{ADC6}{1024 - ADC6}\right) + \ln\left(\frac{R}{R_N}\right) + \frac{B}{T_N}}. \quad (7.18)$$

Pojednostavljena relacija (7.19) manje je zahtjevna za izračun u mikrokontroleru jer smo dio proračuna napravili ručno. U praksi se preporučuje pojednostavljenje matematičkih relacija kako bi mikrokontroler 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 konstantu  $B = 3450 \text{ K}$ :

$$T = \frac{3450}{\ln\left(\frac{ADC6}{1024 - ADC6}\right) + 10,861}. \quad (7.19)$$

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

$$T = \frac{3450}{\ln\left(\frac{ADC6}{1024 - ADC6}\right) + 10,861} - 273,15 \quad (7.20)$$

Relacija (7.20) konačna je i potrebno ju je zapisati u sintaksi programskog jezika C u programski kod 7.7. Funkcija koja u programskom jeziku C računa prirodni logaritam  $\ln()$  je `log(double)`. U programskom kodu je napisana naredba za analogno-digitalnu pretvorbu na pinu PA6, a rezultat pretvorbe sprema se u varijablu ADC6.

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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba715.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

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



## 7.2 Zadaci - analogno-digitalna pretvorba

### Zadatak 7.2.1

Napravite program koji će na početku prvog retka LCD displeja ispisati rezultat analogno-digitalne pretvorbe na pinu PA5, a na početku drugog retka ispisati otpor potencijometra spojenog na pin PA5. Nazivna vrijednost otpora potencijometra je 10 kΩ. Analogno-digitalnu pretvorbu provodite jednom u sekundi. Na pinu PA5 spojen je potencijometar prema shemi prikazanoj na slici 6.1.

---

### Zadatak 7.2.2

Napravite program koji će na početku prvog retka LCD displeja ispisati srednju vrijednost šest uzoraka analogno-digitalne pretvorbe na pinu PA5 uzetih u vremenu od 900 ms. Na početku drugog retka LCD displeja ispišite srednju vrijednost napona na pinu PA5. Na pinu PA5 spojen je potencijometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potencijometra je 10 kΩ.

---

### Zadatak 7.2.3

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 PA5 u koraku  $k$ .

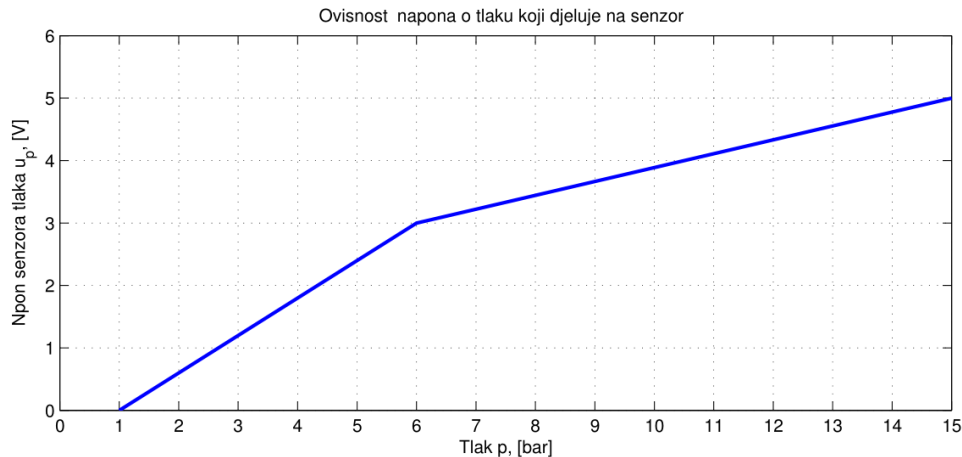
Početne vrijednosti digitalnog filtra drugog reda su  $y[-1] = 0$  i  $y[-2] = 0$ . Vrijeme između uzimanja uzoraka  $k$  i  $k-1$  neka bude 200 ms. Na početku drugog retka ispišite napon dobiven na temelju filtrirane vrijednosti analogno-digitalne pretvorbe na pinu PA5. Dodatno, napravite signalizaciju 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 PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je  $10\text{ k}\Omega$ .

### Zadatak 7.2.4

Potenciometrom spojenim na pin PA5 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 LCD displeja ispisati tlak koji se trenutno mjeri senzorom tlaka. Na pinu PA5 spojen je potenciometar prema shemi prikazanoj na slici 6.1. Nazivna vrijednost otpora potenciometra je  $10\text{ k}\Omega$ .

### Zadatak 7.2.5

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,  $[\text{K}]$ ,
- $B = 3250\text{ K}$  - 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 6.1. Pad napona na NTC otporniku mjeri se pomoću pina PA6.

## Poglavlje 8

# Tajmeri i brojači

Tajmer (eng. *timer*) važan je dio svakog mikrokontrolera i većina mikrokontrolera ima jedan ili više tajmera s rezolucijom od 8 i/ili 16 bitova [4]. Primjena tajmera je široka, a najčešće se koriste za sinkronizaciju događaja u mikrokontroleru, generiranje periodičnih valnih oblika, uzimanje mjernih uzoraka u točnim vremenskim trenucima i u digitalnoj izvedbi PID regulatora. Tajmer je u osnovi brojač (eng. *counter*) koji povećava ili smanjuje svoj iznos za 1 na svaki  $k$ -ti<sup>1</sup> impuls radnog takta. Tajmer i brojač su na hardverskoj 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.

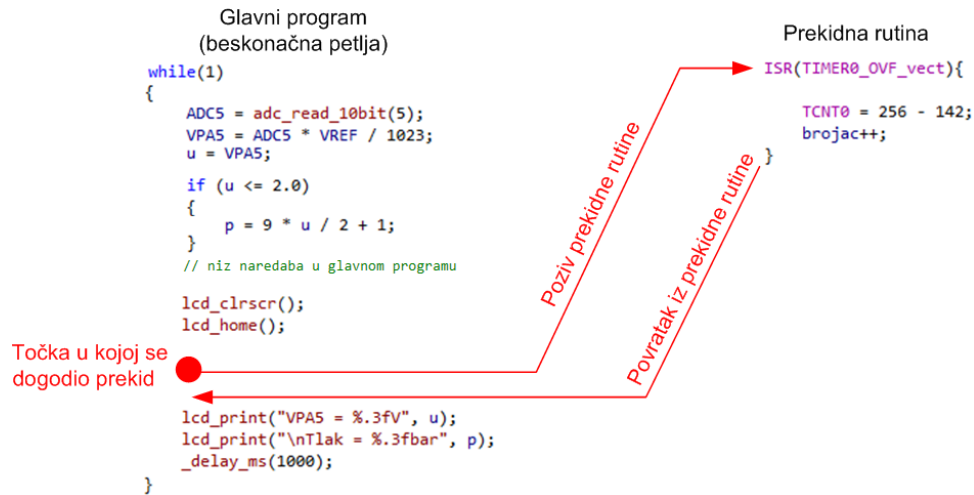
Rezolucija tajmera i brojača je broj bitova u registru koji se koriste za brojanje impulsa s izvora impulsa. Ako je rezolucija tajmera i brojača  $n$ , tada se vrijednosti registra u kojem se broje impulsi kreću u rasponu  $[0, 2^n - 1]$ . Na primjer, ako je rezolucija tajmera i brojača 8, tada se vrijednosti njihovog registra kreću u rasponu od  $[0, 255]$ . Na svaki impuls s izvora impulsa vrijednost u registru se 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 (eng. *overflow*) jer broj 256 ne stane u registar širine 8 bitova. Tajmeri u većini mikrokontrolera generiraju prekid u trenutku kada se dogodi preljev. Prekid (eng. *interrupt*) je mehanizam mikrokontrolera 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, mikrokontroler nastavlja izvoditi programski kod na mjestu gdje je prekidna rutina pozvana (slika 8.1). Mikrokontroleri imaju više izvora prekida. Ukoliko se istovremeno dogodi više prekida s različitim izvorima, prekidi se izvode prema prioritetu.

Tajmeri u mikrokontroleru mogu mjeriti vrijeme jer broje impulse dobivene na temelju radnog takta čija je frekvencija poznata. Radni takt moguće je sklopovski podijeliti s brojem  $k$ , ( $k = 1, 2, 8, 32, 64, 128, 256, 1024$ ). Ukoliko je radni takt sklopovski podijeljen s 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 djeljitelj frekvencije radnog takta (eng. *prescaler*).

---

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



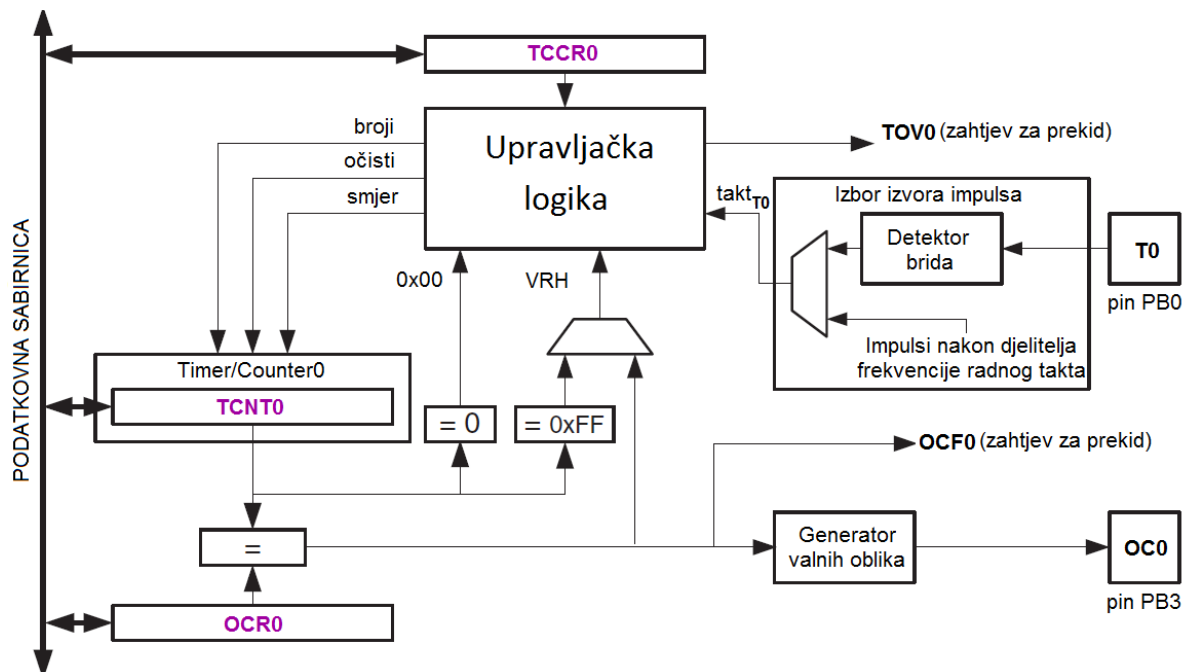
Slika 8.1: Poziv prekidne rutine u trenutku kada se dogodio prekid

## 8.1 Vježbe - tajmeri i brojači

Mikrokontroler ATmega16 ima tri tajmera i brojača:

- *Timer/Counter0* - tajmer i brojač rezolucije 8 bitova s rasponom brojanja od [0, 255],
- *Timer/Counter1* - tajmer i brojač rezolucije 16 bitova s rasponom brojanja od [0, 65535],
- *Timer/Counter2* - tajmer i brojač rezolucije 8 bitova s rasponom brojanja od [0, 255].

Rad tajmera i brojača 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.

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

Sklop *Timer/Counter0* ima tri registra sa sljedećim funkcijama [1]:

- **TCNT0** - (eng. *Timer/Counter Register*), registar širine 8 bitova u kojem se vrijednost uvećava ili smanjuje za 1. Raspon vrijednosti u registru su od [0, 255]. Moguće ga je programski čitati i u njega zapisivati vrijednosti putem podatkovne sabirnice.
- **OCRO** - (eng. *Output Compare Register*), registar širine 8 bitova čija se vrijednost stalno uspoređuje s registrom **TCNT0**. Raspon vrijednosti u registru su od [0, 255]. Moguće ga je programski čitati i u njega zapisivati vrijednosti putem podatkovne sabirnice.
- **TCCRO** - (eng. *Timer/Counter Control Register*), registar kojim se konfigurira rad sklopa *Timer/Counter0*.

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 smanjaj za 1,
- **takt<sub>T0</sub>** - izvor impulsa na temelju kojih se vrijednost registra **TCNT0** uvećava ili smanjuje za 1,
- **0x00** - signalizira da je vrijednost registra **TCNT0** 0x00, a koristi se za generiranje prekida kada se dogodi preljev,
- **VRH** - signalizira da je vrijednost registra **TCNT0** postigla maksimalnu vrijednost koja može biti 255 ili vrijednost koja se nalazi u registru **OCRO**.

Sklop *Timer/Counter0* generira dvije vrste prekida:

- **TOV0** - prekid koji se generira kada registar **TCNT0** prelazi iz vrijednosti 255 u 0,
- **OCF0** - prekid koji se generira kada je vrijednost registra **TCNT0** jednaka vrijednosti registra **OCRO**.

Registar kojim se konfigurira rad sklopa *Timer/Counter0* prikazan je na slici 8.3. Svaki bit u registru **TCCRO** ima svoju funkciju i određuje rad sklopa *Timer/Counter0*.

Bit	7	6	5	4	3	2	1	0
Registar <b>TCCRO</b>	<b>FOCO</b>	<b>WGM00</b>	<b>COM01</b>	<b>COM00</b>	<b>WGM01</b>	<b>CS02</b>	<b>CS01</b>	<b>CS00</b>

Slika 8.3: Registar **TCCRO**

Na slici 8.2 prikazan je blok **Izbor izvora impulsa**. Izvor impulsa mogu biti ili impulsi nakon djelitelja frekvencije radnog takta ili impulsi koji se dovode na pin PB0 (T0). Ponovno možemo primijetiti da pin PB0 ima višestruku funkciju. Izvor impulsa odabire se u registru **TCCRO** 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]

CS02	CS01	CS00	Izvor impulsa
0	0	0	Nema impulsa, <i>Timer/Counter0</i> isključen
0	0	1	F_CPU
0	1	0	F_CPU/8
0	1	1	F_CPU/64
1	0	0	F_CPU/256
1	0	1	F_CPU/1024
1	1	0	Na padajući brid signala pina PB0, uvećaj TCNT0 za 1
1	1	1	Na rastući brid signala pina PB0, uvećaj TCNT0 za 1

Četiri su osnovna načina rada tajmera:

- normalan način rada,
- *CTC* (eng. *Clear Timer on Compare Match*) način rada,
- *Fast PWM* (eng. *Pulse Width Modulation*) način rada,
- *Phase Correct PWM* način rada.

U vježbama ćemo koristiti normalan način rada i *Fast PWM* način rada te ćemo ih detaljnije opisati u nastavku. Opis preostala dva načina rada nalazi se u literaturi [1] na stranicama 76 i 79. Način rada sklopa *Timer/Counter0* odabire se pomoću bitova **WGMO1** i **WGMO0** u registru **TCCRO** prema tablici 8.2.

Tablica 8.2: Odabir načina rada tajmera

WGMO1	WGMO0	Način rada
0	0	Normalni način rada
0	1	<i>Phase Correct</i> način rada
1	0	<i>CTC</i> način rada
1	1	<i>Fast PWM</i> način rada

### 8.1.1 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 6* pozivaju makronaredbom **ISR(vector)** (eng. *Interrupt Service Routines*). Ova makronaredba zahtjeva uključenje zaglavlja **#include <avr/interrupt.h>**. Makronaredba **ISR** kao argument prima prekidni vektor koji je definiran tablicom 18 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()**. Generiranje prekida za preljev u registru **TCNT0**

možemo omogućiti u registru `TIMSK` tako da bit 0 (`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
TIMSK |= (1 << TOIE0); // omogućenje prekida za timer0
TCCR0 |= (0 << WGM01) | (0 << WGM00); // Normalan način rada
TCCR0 |= (1 << CS02) | (0 << CS01) | (0 << CS00); // F_CPU/256
```

Makronaredbom `sei()` globalno smo omogućili prekide, a naredbom `TIMSK |= (1 << TOIE0)`; prekid koji se generira kada se u registru `TCNT0` dogodi preljev. Normalan način rada tajmera omogućili smo naredbom `TCCR0 |= (0 << WGM01) | (0 << WGM00)`; prema tablici 8.2. Pretpostavimo da je frekvencija radnog takta 8 MHz. Kada ne bi bilo dijeljenja frekvencije radnog takta, registar `TCNT0` bi 8000000 puta svoju vrijednost uvećao za 1 u jednoj sekundi i generirao 31250 preljeva. U registar `TCCR0` na mjesto zadnja tri bita upisana je vrijednost 0x04 naredbom `TCCR0 |= (1 << CS02) | (0 << CS01) | (0 << CS00)`; Prema tablici 8.1, 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  $8000000/256 = 31250$  puta u jednoj sekundi i generirati 122 prekida. Vrijeme između dva prekida bilo bi  $1 \text{ s}/122 = 0.008 \text{ s} = 8 \text{ ms}$ , a ono se općenito može izračunati pomoću sljedeće relacije:

$$t_{T0} = \frac{PRESCALER}{F\_CPU} \cdot (256 - TCNT0_0) \quad (8.1)$$

gdje je:

- $t_{T0}$  - vrijeme između dva prekida, [s],
- *PRESCALER* - djelitelj frekvencije radnog takta,
- *F\_CPU* - frekvencija radnog takta mikrokontrolera,
- $256 - 2^8$  gdje broj 8 predstavlja rezoluciju sklopa *Timer/Counter0* (za *Timer/Counter1* 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 zadan 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 dva 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 dva prekida traje 256 impulsa.

Primjena ove vrste prekida je pozivanje prekidne rutine svakih  $t_{T0}$  vremena (npr. potrebno je mjeriti temperaturu peći na kruta goriva svakih 20 ms). Varijable u relaciji (8.1) koje možemo mijenjati su *PRESCALER*, *F\_CPU* i  $TCNT0_0$ . Frekvenciju radnog takta u pravilu ne mijenjamo, već je fiksiramo na samom početku razvoja programskog rješenja. Ono što možemo mijenjati je varijabla *PRESCALER* i  $TCNT0_0$ . Kako bismo dobili željeno vrijeme između dva 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 najmanjeg djelitelja frekvencije radnog takta za kojeg će vrijediti da početna vrijednost registra `TCNT0` ne bude manja od 0. Pokušajmo u relaciju (8.2) uvrstiti  $PRESCALER = 256$  ( $t_{T0} = 20$  ms):

$$TCNT0_0 = 256 - 0,02 \cdot \frac{8000000}{256} = -369. \quad (8.3)$$

Početna vrijednost registra `TCNT0` je prema (8.3) manja od 0 pa je potrebno pokušati s većim djeliteljem frekvencije radnog takta (npr.  $PRESCALER = 1024$ ):

$$TCNT0_0 = 256 - 0,02 \cdot \frac{8000000}{1024} = 99,75 \approx 100. \quad (8.4)$$

Početna vrijednost registra `TCNT0` je prema (8.4) veća od 0 pa je proračun gotov. U programskom kodu 8.1 sada moramo zamijeniti naredbu za namještanje djelitelja frekvencije radnog takta s naredbom `TCCR0 |= (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:

$$t_{T0} = \frac{1024}{8000000} \cdot (256 - 100) = 19,97ms. \quad (8.5)$$

Stvarno vrijeme između dva prekida neće biti 20 ms, već 19,97 ms. Razlog tome je 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, 100, 101, ...

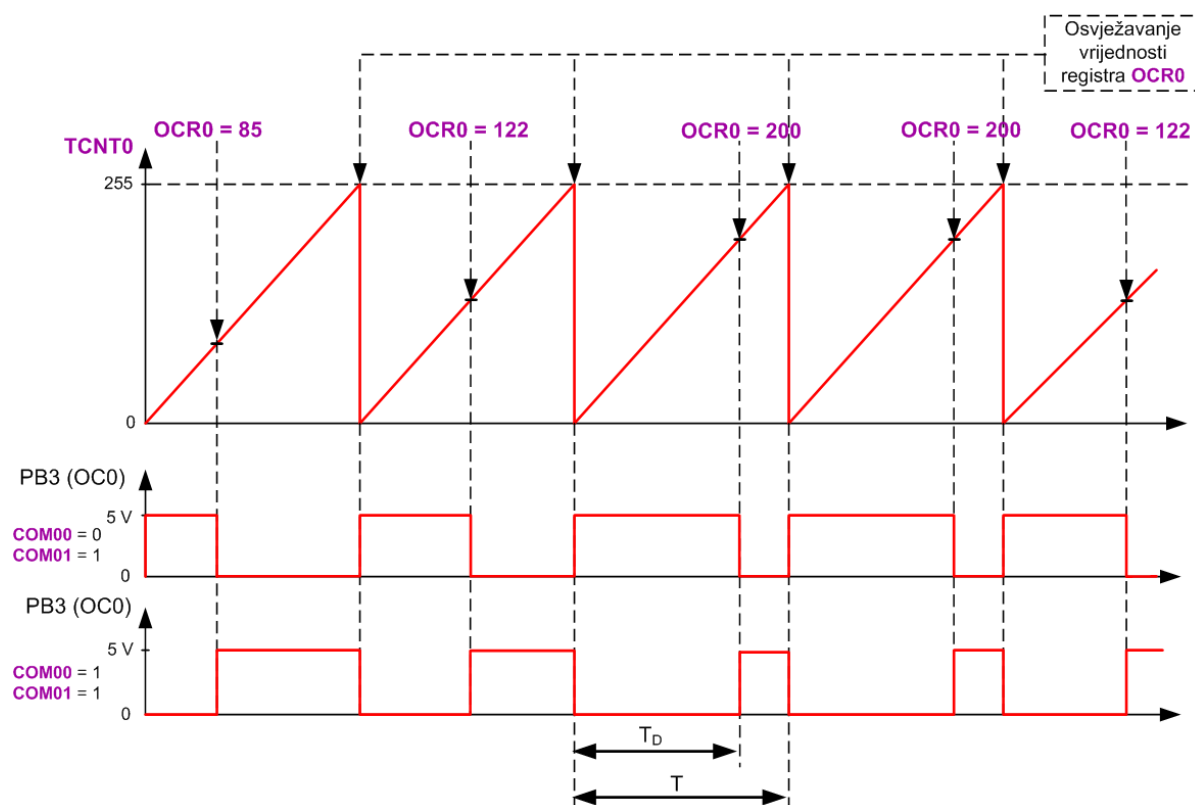
Programski kod 8.2: Prekidna rutina za prekidni vektor `TIMERO_OVF_vect`

```
ISR(TIMERO_OVF_vect){
    TCNT0 = 100;
    // niz naredbi koje mjere temperaturu peći na kruta goriva
}
```

### 8.1.2 Fast PWM način rada tajmera

U *Fast PWM* načinu rada tajmera na temelju vrijednosti u registrima `TCNT0` i `OCR0` generira se pravokutni valni oblik frekvencije  $F_{fPWM}$  na pinu PB3 (`OC0`). *PWM* se najčešće koristi kod upravljanja brzinom vrtnje istosmjernog motora, ispravljača, izmjenjivača i digitalno analogne pretvorbe. Vremenski dijagram *Fast PWM* načina rada prikazan je na slici 8.4. Vrijednost registra `TCNT0` cijelo se vrijeme uvećava za 1 i kreće se od 0 do 255. Nakon što registar `TCNT0` postigne vrijednost 255, ponovno se postavlja u 0. U registar `OCR0` možemo upisati vrijednosti u rasponu od [0, 255]. Vrijednost u registru `OCR0` osvježava se svaki put kada vrijednost registra `TCNT0` prelazi iz 255 u 0. U ovisnosti o konfiguraciji bitova `COM01` i `COM00` prema tablici 8.3 na pinu PB3 generira se pravokutni valni oblik prikazan na slici 8.4.



Slika 8.4: Vremenski dijagram *Fast PWM* načina radaTablica 8.3: Postavke bitova **COM01** i **COM00** u *Fast PWM* načina rada [1]

<b>COM01</b>	<b>COM00</b>	<b>Postavke <i>Fast PWM</i> načina rada</b>
0	0	Pin PB3 (OC0) isključen
0	1	Rezervirano
1	0	Pin PB3 (OC0) postavlja se u nisko stanje kada je vrijednost registra <b>TCNT0</b> jednaka vrijednosti registra <b>OCR0</b> . Pin PB3 postavlja se u visoko stanje kada vrijednost registra <b>TCNT0</b> postane jednaka 0.
1	1	Pin PB3 (OC0) postavlja se u visoko stanje kada je vrijednost registra <b>TCNT0</b> jednaka vrijednosti registra <b>OCR0</b> . Pin PB3 postavlja se u nisko stanje kada vrijednost registra <b>TCNT0</b> postane jednaka 0.

Ako je konfiguracija bitova **COM01** = 1 i **COM00** = 0, tada se pin PB3 (OC0) postavlja u nisko stanje kada je vrijednost registra **TCNT0** jednaka vrijednosti registra **OCR0**, a u visoko stanje kada vrijednost registra **TCNT0** postane jednaka 0. Ovakva vrsta konfiguracije generira neinvertirajući *PWM* signal (slika 8.4).

Ako je konfiguracija bitova **COM01** = 1 i **COM00** = 1, tada se pin PB3 (OC0) postavlja u visoko stanje kada je vrijednost registra **TCNT0** jednaka vrijednosti registra **OCR0**, a u nisko stanje kada vrijednost registra **TCNT0** postane jednaka 0. Ovakva vrsta konfiguracije generira invertirajući *PWM* signal (slika 8.4).

Blokovska shema na slici 8.2 prikazuje sklopovski dio koji provjerava jednakost registara **TCNT0** i **OCR0** te dio koji generira valni oblik na pinu PB3.

Frekvencija *PWM* signala u *Fast PWM* načina rada je:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256}, \quad (8.6)$$

a period *PWM* signala je:

$$T = \frac{1}{F_{fPWM}} = \frac{PRESCALER \cdot 256}{F_{CPU}}. \quad (8.7)$$

Frekvencija i period *PWM* signala ovisi 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. Širina impulsa neinvertirajućeg *PWM* signala određena je vremenom visokog stanja  $T_D$  (eng. *Duty Cycle*) u odnosu na vrijeme perioda  $T$  (slika 8.4) prema relaciji:

$$\frac{T_D}{T} = \frac{OCR0}{255}. \quad (8.8)$$

Širinu impulsa mijenjamo promjenom vrijednosti registra *OCR0* prema relaciji:

$$OCR0 = \frac{T_D}{T} \cdot 255. \quad (8.9)$$

Omjer  $\frac{T_D}{T}$  predstavlja postotak aktivnog stanja *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 PB3 upravlja radom sklopke u izvedbi unipolarnog tranzistora. Označimo izlazni napon na silaznom pretvaraču  $U_i$ . Izlazni napon može se izračunati prema relaciji:

$$U_i = \frac{T_D}{T} \cdot U_{DC} = \frac{OCR0}{255} \cdot U_{DC}. \quad (8.10)$$

Napon na izlazu iz silaznog pretvarača mijenjamo promjenom vrijednosti registra *OCR0* prema relaciji:

$$OCR0 = \frac{U_i}{U_{DC}} \cdot 255. \quad (8.11)$$

U programskom kodu 8.3 prikazana je konfiguracija tajmera u *Fast PWM* načinu rada.

Programski kod 8.3: Konfiguracija tajmera u *Fast PWM* načinu rada

```
TCCR0 |= (0 << CS02) | (1 << CS01) | (0 << CS00); //F_CPU/8
TCCR0 |= (1 << WGM01) | (1 << WGM01); // Fast PWM način rada
TCCR0 |= (1 << COM01) | (0 << COM00); // Neinvertirajući PWM
OCR0 = 100;
```

Naredbom `TCCR0 |= (0 << CS02) | (1 << CS01) | (0 << CS00);` frekvenciju radnog takta podijelili smo s 8 prema tablici 8.1. Frekvencija *PWM* signala je prema relaciji 8.6:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256} = \frac{8000000}{8 \cdot 256} = 3906,25 \text{ Hz}. \quad (8.12)$$

*Fast PWM* način rada tajmera odabran je naredbom `TCCR0 |= (1 << WGM01) | (1 << WGM01);` prema tablici 8.2. Neinvertirajući način rada odabran je naredbom `TCCR0 |= (1 << COM01) | (0 << COM00);`. U tom načinu rada vrijednost registra *OCR0* 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 8.3 može se izračunati prema relaciji (8.10):

$$U_i = \frac{OCR0}{255} \cdot U_{DC} = \frac{100}{255} \cdot U_{DC} = 11.76V. \quad (8.13)$$

Ako na izlazu želimo postići vrijednost izlaznog napona 25 V, tada je u registar `OCR0` potrebno upisati vrijednost dobivenu relacijom (8.11):

$$OCR0 = \frac{U_i}{U_{DC}} \cdot 255 = \frac{25}{30} \cdot 255 = 213. \quad (8.14)$$

Za sklopove *Timer/Counter1* i *Timer/Counter2* konfiguracija načina rada slična je prethodno provedenoj konfiguraciji sklopa *Timer/Counter0*. Za sve detalje o konfiguraciji sklopova *Timer/Counter1* i *Timer/Counter2* pogledajte literaturu [1]. U narednim vježbama frekvencija radnog takta mikrokontrolera bit će 8 MHz.

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.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 `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 `*.c`.

U datoteci s vježbama nalaze se i rješenja vježbi koje 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 25 ms. Crvena LED dioda spojena je na pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba811.c`. Omogućite prevođenje samo datoteke `vjezba811.c`. Početni sadržaj datoteke `vjezba811.c` prikazan je programskim kodom 8.4.

Programski kod 8.4: Početni sadržaj datoteke `vjezba811.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include <avr/interrupt.h>

ISR(TIMERO_OVF_vect){ // prekidna rutina za timer0
    TCNT0 = 0; // početna vrijednost registra
    TOGGLE_PORT(PORTB, PB7);
}

void inicijalizacija(){

    output_port(DDRB, PB7); // pin PB7 postavljen kao izlazni
    set_port(PORTB, PB7, 1); // početno stanje crvene LED diode
```

```

    lcd_init();
    timer0_init();
    //sei(); // globalno omogućavanje prekida
    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 od 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 je `#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>`.

U programskom kodu 8.4 u funkciji `inicijalizacija()` nalazi se funkcija `timer0_init()`. Ova funkcija konfigurira djelitelj frekvencije radnog takta i omogućuje prekid sklopa *Timer/Counter0* kada dođe do preljeva registra `TCNT0`. Prije korištenja normalnog rada tajmera, potrebno je u datoteci `timer.h` podesiti djelitelj frekvencije radnog takta. Na temelju tablice 8.1 u datoteci `timer.h` napisane su maske koje se koriste u konfiguraciji djelitelja frekvencije radnog takta. U programskom kodu 8.5 nalazi se dio sadržaja datoteke `timer.h` koji se odnosi samo na sklop *Timer/Counter0*. Kako bismo definirali djelitelj frekvencije radnog takta, potrebno je konstanti `TIMERO_PRESCALER` dodijeliti definiranu masku. U slučaju programskog koda 8.4 odabran je djelitelj frekvencije radnog takta 1024.

Programski kod 8.5: Odabir djelitelja frekvencije radnog takta u datoteci `timer.h`

```

//maske za djelitelja frekvencije
#define TIMERO_PRESCALER_OFF      0x0
#define TIMERO_PRESCALER_1       0x1
#define TIMERO_PRESCALER_8       0x2
#define TIMERO_PRESCALER_64      0x3
#define TIMERO_PRESCALER_256     0x4
#define TIMERO_PRESCALER_1024    0x5
#define TIMERO_EXTERNAL_FALL_EDGE 0x6
#define TIMERO_EXTERNAL_RISI_EDGE 0x7

// korisnik odabire djelitelj frekvencije za timer0
#define TIMERO_PRESCALER TIMERO_PRESCALER_1024

```

U funkciji `inicijalizacija()` potrebno je pozvati makronaredbu `sei()` koja globalno omogućuje prekide. Ispod poziva makronaredbe `sei()` potrebno je postaviti početnu vrijednost registra `TCNT0`. Trenutna početna vrijednost je 0, što ćemo nešto kasnije promijeniti.

Uvijek kada se dogodi preljev registra `TCNT0`, poziva se prekidna rutina prikazana programskim kodom 8.6.

Programski kod 8.6: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT0`

```
ISR(TIMER0_OVF_vect){ // prekidna rutina za timer0
    TCNT0 = 0;
    TOGGLE_PORT(PORTB, PB7);
}
```

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 25 ms pa vrijedi da je  $t_{T0} = 0,025$  s. Početnu vrijednost registra `TCNT0` možemo izračunati prema relaciji (8.2):

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F_{CPU}}{PRESCALER} = 256 - 0,025 \cdot \frac{8000000}{1024} = 61. \quad (8.15)$$

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 mikrokontrolera u minutama. Primitite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikrokontroler 60 sekundi ništa ne radi u `while` petlji. Prekidna se rutina poziva bez obzira na kašnjenje u `while` petlji i to je velika korist prekidnih rutina uopće.

Prevedite datoteku `vjezba811.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Ukoliko crvena LED dioda ne izmjenjuje stanje, izbrišite komentar kod makronaredbe `sei()` 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.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

U nastavku vježbe izbrisat ćemo funkciju `timer0_init()` iz funkcije `inicijalizacija()` te napisati niz naredbi iz programskog koda 8.7.

Programski kod 8.7: Konfiguracija sklopa *Timer/Counter0* bez funkcije `timer0_init()`

```
TCCR0 |= (0 << WGM01) | (0 << WGM00); //normalan način rada
TCCR0 |= (1 << CS02) | (0 << CS01) | (1 << CS00); // F_CPU / 1024
TIMSK |= (1 << TOIE0); // omogući prekid TOV0
```

U programskom kodu direktno smo konfigurirali bitove u registru `TCCR0` prema tablicama 8.1 i 8.2. U registru `TIMSK` omogućili smo prekid koji se događa prilikom preljeva registra `TCNT0`. Prevedite datoteku `vjezba811.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

U zaglavlju `AVR_lib.h` dodane su dvije nove 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 8.8 konfiguraciju sklopa *Timer/Counter0* napravili smo pomoću makronaredbi `set_bit_reg` i `reset_bit_reg` prema tablicama 8.1 i 8.2. Ovaj način konfiguracije sklopa *Timer/Counter0* jednostavan je, a daje jasan pregled stanja bitova u registrima sklopa *Timer/Counter0*.

Programski kod 8.8: Konfiguracija sklopa *Timer/Counter0* pomoću makronaredbi

`set_bit_reg` i `reset_bit_reg`

```
//normalan način rada
reset_bit_reg(TCCR0,WGM00); //WGM00 = 0
reset_bit_reg(TCCR0,WGM01); //WGM01 = 0
// F_CPU / 1024
set_bit_reg(TCCR0,CS00); // CS00 = 1
reset_bit_reg(TCCR0,CS01); // CS01 = 0
set_bit_reg(TCCR0,CS02); // CS02 = 1
// omogući prekid TOV0
set_bit_reg(TIMSK,TOIE0); // TOIE0 = 1
```

Konfigurirajte sklop *Timer/Counter0* pomoću niza naredbi u programskom kodu 8.8, a zatim prevedite datoteku `vjezba811.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba811.c` i onemogućite prevođenje ove datoteke.



## Vježba 8.1.2

Napravite program u kojem ćete pomoću sklopa *Timer/Counter1* mijenjati stanje crvene LED diode svakih 200 ms. Crvena LED dioda spojena je na pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektom stablu otvorite datoteku `vjezba812.c`. Omogućite prevođenje samo datoteke `vjezba812.c`. Početni sadržaj datoteke `vjezba812.c` prikazan je programskim kodom 8.9.

Programski kod 8.9: Početni sadržaj datoteke `vjezba812.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include <avr/interrupt.h>

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 0; // početna vrijednost registra
    TOGGLE_PORT(PORTB, PB7);
}

void inicijalizacija(){

    output_port(DDRB,PB7); // pin PB7 postavljen kao izlazni
    set_port(PORTB,PB7,1); // početno stanje crvene LED diode

    lcd_init();

    //normalan način rada - 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
    // F_CPU / 64
    set_bit_reg(TCCR1B,CS10); // CS10 = 1
    set_bit_reg(TCCR1B,CS11); // CS11 = 1
    reset_bit_reg(TCCR1B,CS12); // CS12 = 0
    // omogući prekid TOV1
    set_bit_reg(TIMSK,TOIE1); // TOIE1 = 1

    sei(); // globalno omogućavanje prekida
    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 od 60 s
    }

    return 0;
}

```

U ovoj vježbi koristit ćemo sklop *Timer/Counter1* kao tajmer u normalnom načinu rada. Rezolucija sklopa *Timer/Counter1* je 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 broju (0 je zamijenjena s 1).

Budući da sklop *Timer/Counter1* pruža više mogućnosti od sklopa *Timer/Counter0*, konfiguracija sklopa *Timer/Counter1* izvršava se pomoću dva registra **TCCR1A** i **TCCR1B**. Detalje o konfiguraciji sklopa *Timer/Counter1* pogledajte u literaturi [1] u tablicama 47 i 48.

Prema tablicama 47 i 48 u literaturi [1] u funkciji **inicijalizacija()** konfiguriran je sklop *Timer/Counter1* kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 64. Prekid kojeg izaziva preljev u registru **TCNT1** omogućuje se upisivanjem broja 1 na mjesto bita **TOIE1** u registru **TIMSK**. Za globalno omogućavanje prekida pozvana je makronaredba **sei()**.

Vrijeme između dva poziva prekidne rutine mora biti 200 ms ( $t_{T1} = 0,2$  s). Početnu vrijednost registra **TCNT1** izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 0,2 \cdot \frac{8000000}{64} = 40536. \quad (8.16)$$

Spomenuli smo ranije da je kod sklopa *Timer/Counter1* u relaciji (8.2) broj 256 potrebno zamijeniti s brojem 65536. Početna vrijednost registra **TCNT1** je 40536. Upišite tu vrijednost na odgovarajuća mjesta u programskom kodu 8.9.

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 crvene LED diode.

U **while** petlji na LCD displej ispisuje se vrijeme rada mikrokontrolera u minutama. Primi-

jetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikrokontroler 60 sekundi ništa ne radi u `while` petlji. Prekidna rutina se poziva bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba812.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite vrijeme između poziva prekidne rutine u 500 ms. Ponovno prevedite datoteku `vjezba812.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Izbrišite u funkciji `inicijalizacija()` sve makronaredbe `set_bit_reg` i `reset_bit_reg`. Pokušajte konfigurirati sklop *Timer/Counter1* pomoću funkcije `timer1_init()`. Promijenite vrijeme između poziva prekidne rutine na 50 ms, a djelitelj frekvencije radnog takta u zaglavlju `timer.h` postavite na vrijednost 8. Prevedite datoteku `vjezba812.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba812.c` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.3

Napravite program u kojem ćete pomoću sklopa *Timer/Counter2* mijenjati stanje crvene LED diode svakih 100 ms. Crvena LED dioda spojena je na pin PB7. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba813.c`. Omogućite prevođenje samo datoteke `vjezba813.c`. Početni sadržaj datoteke `vjezba813.c` prikazan je programskim kodom 8.10.

Programski kod 8.10: Početni sadržaj datoteke `vjezba813.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"

void inicijalizacija(){

    output_port(DDRB,PB7); // pin PB7 postavljen kao izlazni
    set_port(PORTB,PB7,1); // početno stanje crvene LED diode

    lcd_init();

    //normalan način rada
    reset_bit_reg(TCCR2,WGM20); //WGM20 = 0
    reset_bit_reg(TCCR2,WGM21); //WGM21 = 0
    // F_CPU / 1024
    set_bit_reg(TCCR2,CS20); // CS20 = 1
    set_bit_reg(TCCR2,CS21); // CS21 = 1
    set_bit_reg(TCCR2,CS22); // CS22 = 1
    // omogući prekid TOV2
    set_bit_reg(TIMSK,TOIE2); // TOIE2 = 1

    // globalno omogućite prekide

    TCNT2 = 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 od 60 s
    }

    return 0;
}

```

U programski kod 8.10 uključite zaglavlje `interrupt.h` pomoću naredbe `#include <avr/interrupt.h>` te globalno omogućite prekide. U ovoj vježbi koristit ćemo sklop *Timer/Counter2* kao tajmer u normalnom načinu rada. Rezolucija sklopa *Timer/Counter2* je 8 bitova pa se raspon vrijednosti u registru u kojem se broje impulsi kreće od [0, 255]. Registar u kojem se broje impulsi zove se `TCNT2`. U programskom kodu 8.10 u funkciji `inicijalizacija()` konfiguriran je sklop *Timer/Counter2* kao tajmer u normalnom načinu rada s djeljiteljem frekvencije radnog takta 1024 prema tablicama 50 i 54 u literaturi [1]. Prekid kojeg generira preljev u registru `TCNT2` omogućuje se upisivanjem broja 1 na mjesto bita `TOIE2` u registru `TIMSK`. Vrijeme između dva poziva prekidne rutine mora biti 100 ms ( $t_{T2} = 0,1$  s). Početnu vrijednost registra `TCNT2` izračunat ćemo pomoću relacije (8.2):

$$TCNT2_0 = 256 - t_{T2} \cdot \frac{F_{CPU}}{PRESCALER} = 256 - 0,1 \cdot \frac{8000000}{1024} = -525. \quad (8.17)$$

Početna vrijednost registra `TCNT2` dobivena relacijom (8.17) manja je od 0. To znači da s najvećim djeljiteljem frekvencije radnog takta nije moguće ostvariti vrijeme između dva poziva prekidne rutine od 100 ms. Za ovaj problem postoji rješenje koje ćemo prikazati u nastavku. Na primjer, poziv između dvije prekidne rutine u vremenskom razmaku od 100 ms možemo podijeliti na četiri poziva prekidne rutine u vremenskom razmaku od 25 ms. Stanje crvene LED diode u prekidnoj rutini tada ćemo mijenjati u svakom četvrtom pozivu prekidne rutine. Izračunajmo novu početnu vrijednost registra `TCNT2` za četvrtinu vremena od 100 ms:

$$TCNT2_0 = 256 - \frac{t_{T2}}{4} \cdot \frac{F_{CPU}}{PRESCALER} = 256 - 0,025 \cdot \frac{8000000}{1024} = 61. \quad (8.18)$$

Uvijek kada se dogodi preljev registra `TCNT2`, poziva se prekidna rutina prikazana programskim kodom 8.11. U programski kod 8.10 upišite prekidnu rutinu `ISR(TIMER2_OVF_vect)` prikazanu programskim kodom 8.11.

Programski kod 8.11: Prekidna rutina koja se poziva kada se dogodi preljev registra `TCNT2`

```

uint8_t prazan_hod = 0;

ISR(TIMER2_OVF_vect){ // prekidna rutina za timer2
    TCNT2 = 61; // početna vrijednost registra
    prazan_hod++;
    if (prazan_hod == 4)
    {
        TOGGLE_PORT(PORTB, PB7);
        prazan_hod = 0;
    }
}

```

U programskom kodu 8.11 deklarirana je varijabla `prazan_hod`. Ova varijabla koristi se kao brojač poziva prekidne rutine `ISR(TIMER2_OVF_vect)`. U svakom četvrtom pozivu prekidne rutine mijenjamo stanje crvene LED diode. Vrijeme između dva poziva prekidne rutine je 25 ms te se stanje crvene LED diode mijenja svakih 100 ms ( $25 \text{ ms} \cdot 4 = 100 \text{ ms}$ ).

Navedeni postupak isti je za sva vremena koja se ne mogu ostvariti s 256 impulsa registara `TCNT0` i `TCNT2`, odnosno sa 65536 impulsa registra `TCNT1`. Zadano vrijeme podijelite s najmanjim cijelim brojem za koji će početna vrijednost registara `TCNT0`, `TCNT1` i `TCNT2` biti veća i jednaka 0.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikrokontrolera u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikrokontroler 60 sekundi ništa ne radi u `while` petlji. Prekidna rutina se poziva bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba813.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite vrijeme između poziva prekidne rutine u 200 ms. Ponovno prevedite datoteku `vjezba813.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba813.c` i onemogućite prevođenje ove datoteke.



#### Vježba 8.1.4

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje crvene LED diode svake 22 ms, pomoću sklopa *Timer/Counter1* mijenjati stanje žute LED diode svakih 180 ms, a pomoću sklopa *Timer/Counter2* mijenjati stanje zelene LED diode svake 32 ms. Crvena LED dioda spojena je na pin PB7, žuta LED dioda spojena je na pin PB6, a zelena LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba814.c`. Omogućite prevođenje samo datoteke `vjezba814.c`. Početni sadržaj datoteke `vjezba814.c` prikazan je programskim kodom 8.12.

Programski kod 8.12: Početni sadržaj datoteke `vjezba814.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include <avr/interrupt.h>

ISR(TIMER0_OVF_vect){ // prekidna rutina za timer0
    TCNT0 = 0; // početna vrijednost registra
    TOGGLE_PORT(PORTB, PB7);
}

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 0; // početna vrijednost registra
    TOGGLE_PORT(PORTB, PB6);
}

ISR(TIMER2_OVF_vect){ // prekidna rutina za timer2
    TCNT2 = 0; // početna vrijednost registra
    TOGGLE_PORT(PORTB, PB5);
}
```

```

void inicijalizacija(){

    output_port(DDRB,PB7); // pin PB7 postavljen kao izlazni
    set_port(PORTB,PB7,1); // početno stanje crvene LED diode
    output_port(DDRB,PB6); // pin PB6 postavljen kao izlazni
    set_port(PORTB,PB6,1); // početno stanje žute LED diode
    output_port(DDRB,PB5); // pin PB5 postavljen kao izlazni
    set_port(PORTB,PB5,1); // početno stanje zelene LED diode

    lcd_init();

    // inicijalizacija svih tajmera
    timer0_init();
    timer1_init();
    timer2_init();

    sei(); // globalno omogućavanje prekida

    TCNT0 = 0;
    TCNT1 = 0;
    TCNT2 = 0;
}

int main(void){

    inicijalizacija();

    uint16_t minute = 0;

    while(1)
    {
        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje od 60 s
    }

    return 0;
}

```

U programskom kodu 8.12 u funkciji `inicijalizacija()` nalaze se funkcije `timer0_init()`, `timer1_init()` i `timer2_init()`. Ove funkcije konfiguriraju normalan način rada i djelitelja frekvencije radnog takta svih tajmera te omogućuje prekide koje generiraju svi tajmeri. U datoteci `timer.h` podesite djelitelje frekvencije radnog takta svih tajmera.

Vrijeme između dva poziva prekidne rutine `ISR(TIMERO_OVF_vect)` mora biti 22 ms ( $t_{T0} = 0,022$  s). Početnu vrijednost registra `TCNT0` izračunat ćemo pomoću relacije (8.2):

$$TCNT0_0 = 256 - t_{T0} \cdot \frac{F_{CPU}}{PRESCALER_0} = 256 - 0,022 \cdot \frac{8000000}{1024} = 84. \quad (8.19)$$

Vrijeme između dva poziva prekidne rutine `ISR(TIMER1_OVF_vect)` mora biti 180 ms ( $t_{T1} = 0,18$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65535 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER_1} = 65535 - 0,18 \cdot \frac{8000000}{64} = 43036. \quad (8.20)$$

Vrijeme između dva poziva prekidne rutine `ISR(TIMER2_OVF_vect)` mora biti 32 ms

( $t_{T2} = 0,032$  s). Početnu vrijednost registra `TCNT2` izračunat ćemo pomoću relacije (8.2):

$$TCNT2_0 = 256 - t_{T2} \cdot \frac{F_{CPU}}{PRESCALER\_2} = 256 - 0,032 \cdot \frac{8000000}{1024} = 6. \quad (8.21)$$

Upišite početne vrijednosti registara `TCNT0`, `TCNT1` i `TCNT2` u programski kod 8.12 na sva predviđena mjesta.

U `while` petlji na LCD displej ispisuje se vrijeme rada mikrokontrolera u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikrokontroler 60 sekundi ništa ne radi u `while` petlji. Prekidna rutina se poziva bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba814.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba814.c` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.5

Napravite program koji će omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 400 ms i to redosljedom crvena → žuta → zelena → bijela → crvena → ... . Izmjenu stanja LED dioda svakih 400 ms ostvarite pomoću sklopa *Timer/Counter1*. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

U projektnom stablu otvorite datoteku `vjezba815.c`. Omogućite prevođenje samo datoteke `vjezba815.c`. Početni sadržaj datoteke `vjezba815.c` prikazan je programskim kodom 8.13.

Programski kod 8.13: Početni sadržaj datoteke `vjezba815.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include <avr/interrupt.h>

uint8_t brojac = 0;

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 0;
    brojac++;
    switch (brojac)
    {
    case 1:
        set_port(PORTB,PB7,1);
        set_port(PORTB,PB4,0);
        break;
    case 2:
        set_port(PORTB,PB6,1);
        set_port(PORTB,PB7,0);
        break;
    case 3:
        set_port(PORTB,PB5,1);
        set_port(PORTB,PB6,0);
        break;
    case 4:
```

```

        set_port(PORTB,PB4,1);
        set_port(PORTB,PB5,0);
        brojac = 0;
        break;
    default:
        break;
}
}

void inicijalizacija(){

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

    lcd_init();

    timer1_init();

    sei(); // globalno omogućavanje prekida
    TCNT1 = 0;
}

int main(void){

    inicijalizacija();

    uint16_t minute = 0;

    while(1)
    {
        lcd_clrscr();
        lcd_home();
        lcd_print("0N : %u min", minute++);
        _delay_ms(60000); // kašnjenje od 60 s
    }

    return 0;
}

```

U ovoj vježbi koristit ćemo sklop *Timer/Counter1* kao tajmer u normalnom načinu rada. Konfiguracija je ostvarena pomoću funkcije `timer1_init()`. U datoteci `timer.h` potrebno je podesiti djelitelj frekvencije radnog takta za sklop *Timer/Counter1*. U vježbi je potrebno ostvariti trčee LED diode redosljedom crvena → žuta → zelena → bijela → crvena → ... . Izmjenu stanja LED dioda svakih 400 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 trčanje LED dioda.

Vrijeme između dva poziva prekidne rutine mora biti 400 ms ( $t_{T1} = 0,4$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65535 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65535 - 0,4 \cdot \frac{8000000}{64} = 15536. \quad (8.22)$$

Upišite početnu vrijednost registra `TCNT1` u programski kod 8.13 na za to predviđeno mjesto. U `while` petlji na LCD displej ispisuje se vrijeme rada mikrokontrolera u minutama. Primijetite da smo u `while` petlji postavili kašnjenje od 60 s. Zbog tog kašnjenja mikrokontroler 60 sekundi ništa ne radi u `while` petlji. Prekidna rutina se poziva bez obzira na kašnjenje u `while` petlji.

Prevedite datoteku `vjezba815.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite smjer trčanja LED dioda te ponovno prevedite datoteku `vjezba815.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba815.c` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.6

Napravite program u kojem će se, ako je pritisnuto tipkalo spojeno na pin `PB0`, uzimati uzorci s analognog senzora spojenog na pin `PA5`. Vrijeme uzorkovanja neka iznosi 500 ms. Uzimanje uzoraka ostvarite u prekidnoj rutini `ISR(TIMER1_OVF_vect)`. Vrijednost napona na pinu `PA5` ispišite na LCD displej u `while` petlji.

U projektnom stablu otvorite datoteku `vjezba816.c`. Omogućite prevođenje samo datoteke `vjezba816.c`. Početni sadržaj datoteke `vjezba816.c` prikazan je programskim kodom 8.14.

Programski kod 8.14: Početni sadržaj datoteke `vjezba816.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
// uključite zaglavlja

uint16_t ADC5; // rezultat AD pretvorbe
float VPA5; // napon na pinu PA5
const float VREF = 5.0; // AVCC

uint8_t uzorak_procitan = 0; // logička varijabla

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 0;

    ADC5 = adc_read_10bit(5);
    VPA5 = ADC5 * VREF / 1023;
    uzorak_procitan = 1;
}

void inicijalizacija(){

    input_port(DDRB,PB0); // pin PB0 pstavljen kao izlazni
    set_port(PORTB,PB0,1); // uključen pritezni otpornik na PB0

    lcd_init();
    adc_init();

    //normalan način rada - 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
```

```

// F_CPU / 64
set_bit_reg(TCCR1B,CS10); // CS10 = 1
set_bit_reg(TCCR1B,CS11); // CS11 = 1
reset_bit_reg(TCCR1B,CS12); // CS12 = 0

sei(); // globalno omogućavanje prekida
TCNT1 = 0;
}

int main(void){

    inicijalizacija();

    while(1)
    {

        if (uzorak_procitan == 1)
        {
            lcd_clrscr();
            lcd_home();
            lcd_print("Napon : %.2f V", VPA5);
            uzorak_procitan = 0;
        }

        if (get_pin(PINB,PB0) == 0)
        {
            set_bit_reg(TIMSK,TOIE1); // TOIE1 = 1
        }
        else
        {
            reset_bit_reg(TIMSK,TOIE1); // TOIE1 = 0
        }

    }

    return 0;
}

```

U programskom kodu 8.14 prvo je potrebno uključiti sva zaglavlja koja nedostaju. Primijetite da se u vježbi koristi LCD displej, analogno-digitalna pretvorba, tajmer i prekidi. U vježbi je potrebno svakih 500 ms očitati uzorak s analognog senzora spojenog na pin PA5. U praksi se to uvijek radi s tajmerima jer uvijek postoji zahtjev za preciznost uzimanja uzoraka. U funkciji `inicijalizacija()` konfiguriran je izlazni pin PB0 kojim ćemo omogućiti uzimanje uzoraka.

Prema tablicama 47 i 48 u literaturi [1] u funkciji `inicijalizacija()` konfiguriran je sklop *Timer/Counter1* kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 64. Za globalno omogućavanje prekida pozvana je makronaredba `sei()`. Na trenutak ćemo izostaviti omogućavanje prekida kojeg izaziva preljev u registru `TCNT1`.

Mjerenje analognim senzorom provodi se pomoću prekidne rutine `ISR(TIMER1_OVF_vect)`. Uzorke je potrebno uzimati svakih 500 ms ( $t_{T1} = 0,5$  s) pa je početna vrijednost registra `TCNT1` izračunata prema korigiranoj relaciji (8.2):

$$TCNT1_0 = 65535 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER_1} = 65535 - 0,5 \cdot \frac{8000000}{64} = 3036. \quad (8.23)$$

Upišite početnu vrijednost registra `TCNT1` na odgovarajuća mjesta u programskom kodu 8.14. Prekidna rutina poziva se samo ako je omogućen prekid kojeg izaziva preljev u registru `TCNT1`.

Taj prekid omogućuje se upisivanjem broja 1 na mjesto bita `TOIE1` u registru `TIMSK`. U vježbi je zadano da se uzorci uzimaju samo ako je pritisnuto tipkalo PB0. U `while` petlji provjeravat ćemo stanje tipkala spojenog na pin PB0 makronaredbom `get_pin`. Ako je tipkalo pritisnuto tada ćemo makronaredbom `set_bit_reg(TIMSK, TOIE1)`; omogućiti prekid kojeg izaziva preljev u registru `TCNT1`. Ako tipkalo nije pritisnuto, tada ćemo makronaredbom `reset_bit_reg(TIMSK, TOIE1)`; onemogućiti prekid kojeg izaziva preljev u registru `TCNT1`. Na taj smo način omogućili, odnosno onemogućili uzimanje uzoraka.

Vrijednost napona na analognom ulazu PA5 ispisujemo na LCD displeju ako je zadovoljen uvjet `if (uzorak_procitan == 1)`. Varijablu `uzorak_procitan` koristimo kao indikaciju da nam je na raspolaganju novi uzorak kojeg 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(TIMER1_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.

Prevedite datoteku `vjezba816.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba816.c` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.7

Napravite program koji će brojati rastuće bridove signala na pinu PB0 (T0) pomoću sklopa *Timer/Counter0*. Broj rastućih bridova potrebno je ispisati na LCD displeju. Maksimalni broj impulsa koji može pristići na pin PB0 je 1000000.

U projektnom stablu otvorite datoteku `vjezba817.c`. Omogućite prevođenje samo datoteke `vjezba817.c`. Početni sadržaj datoteke `vjezba817.c` prikazan je programskim kodom 8.15.

Programski kod 8.15: Početni sadržaj datoteke `vjezba817.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include <avr/interrupt.h>

uint32_t brojac_prekida = 0;

ISR(TIMER0_OVF_vect){ // prekidna rutina za timer0
    brojac_prekida++;
}

void inicijalizacija(){

    input_port(DDRB,PB0); // pin PB0 postavljen kao ulazni
    set_port(PORTB,PB0,1); // uključen pritezni otpornik na PB0

    lcd_init();

    //normalan način rada
    reset_bit_reg(TCCRO,WGM00); //WGM00 = 0
    reset_bit_reg(TCCRO,WGM01); //WGM01 = 0
```



```

// broji rastuće bridove
set_bit_reg(TCCR0, CS00); // CS00 = 1
set_bit_reg(TCCR0, CS01); // CS01 = 1
set_bit_reg(TCCR0, CS02); // CS02 = 1
// omogući prekid TOV0
set_bit_reg(TIMSK, TOIE0); // TOIE0 = 1

sei(); // globalno omogućavanje prekida
}

int main(void){

    inicijalizacija();

    while(1)
    {
        lcd_clrscr();
        lcd_home();
        lcd_print("Brojac: %lu", 256 * brojac_prekida + TCNT0);
        _delay_ms(1000);
    }

    return 0;
}

```

U ovoj vježbi sklop *Timer/Counter0* koristit ćemo kao brojač rastućih bridova na pinu PB0 (T0). Prema tablici 8.1, bitove **CS00**, **CS01** i **CS02** potrebno je sve redom postaviti na vrijednost 1 kako bi se sklop *Timer/Counter0* koristio kao brojač rastućih bridova impulsa. I u ovom slučaju generira se prekid kada se u registru **TCNT0** dogodi preljev. Prekid kojeg generira preljev u registru **TCNT0** omogućuje se upisivanjem broja 1 na mjesto bita **TOIE1** u registru **TIMSK**. U tu svrhu koristili smo makronaredbu **set\_bit\_reg**. Za globalno omogućavanje prekida pozvana je makronaredba **sei()**.

Sklop *Timer/Counter0* koristimo kao brojač kada brojimo impulse s vanjskih digitalnih senzora. Primjer takvih senzora su induktivni senzor, kapacitivni senzor, optički senzor, tipkala, krajnji prekidači i enkoderi. U vježbi je potrebno omogućiti prikaz do maksimalno 1000000 impulsa na LCD displeju. Registar **TCNT0** ima raspon vrijednosti od [0, 255] pa je pomoću samo njega nemoguće osigurati brojanje do maksimalno 1000000 impulsa. Iskoristit ćemo činjenicu da se prilikom preljeva registra **TCNT0** poziva prekidna rutina **ISR(TIMERO\_OVF\_vect)**. Deklarirali smo globalnu varijablu imena **brojac\_prekida**. U ovoj varijabli bit će pohranjen broj poziva prekidne rutine. Ako se prekidna rutina poziva uvijek kada registar **TCNT0** prelazi iz 255 u 0, odnosno kada je izbrojio 256 impulsa, ukupni broj impulsa  $N$  računa se prema relaciji:

$$N = 256 \cdot \text{brojac\_prekida} + \text{TCNT0}. \quad (8.24)$$

Ovu relaciju upisali smo kao argument funkcije **lcd\_print**. Generator impulsa na razvojnom okruženju s mikrokontrolerom ATmega16 bit će tipkalo spojeno na pin PB0. Zbog toga je pin PB0 potrebno konfigurirati kao ulazni pin i potrebno je uključiti pritezni otpornik. U praksi će uključivanje priteznog otpornika na pinu PB0 ovisiti o vrsti digitalnog senzora kojeg spajate na njega. Senzori s otvorenim kolektorom zahtijevaju uključen pritezni otpornik na pinu PB0, dok senzori s tzv. *push-pull* izlazom ne zahtijevaju uključen pritezni otpornik na pinu PB0.

Prevedite datoteku **vjezba817.c** u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 tako da pritisnete tipkalo spojeno na pin PB0.

Zatvorite datoteku **vjezba817.c** i onemogućite prevođenje ove datoteke.



### Vježba 8.1.8

Napravite program koji će na pinu PB3 (OC0) generirati neinvertirajući *PWM* signal pomoću sklopa *Timer/Counter0*. Frekvenciju *PWM* signala namjestite tako da bude veća od 3 kHz, a manja od 10 kHz. Visoko stanje impulsa *PWM* signala postavite na 70 % perioda *T*. Osciloskopom je potrebno pratiti *PWM* signal na PB3 (OC0) pinu.

U projektnom stablu otvorite datoteku `vjezba818.c`. Omogućite prevođenje samo datoteke `vjezba818.c`. Početni sadržaj datoteke `vjezba818.c` prikazan je programskim kodom 8.16.

Programski kod 8.16: Početni sadržaj datoteke `vjezba818.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "timer/timer.h"
#include <util/delay.h>

void inicijalizacija(){

    //Fast PWM način rada
    set_bit_reg(TCCRO,WGM00); //WGM00 = 1
    set_bit_reg(TCCRO,WGM01); //WGM01 = 1

    // F_CPU/8
    reset_bit_reg(TCCRO,CS00); // CS00 = 0
    set_bit_reg(TCCRO,CS01); // CS01 = 1
    reset_bit_reg(TCCRO,CS02); // CS02 = 0

    //neinvertirajući PWM
    reset_bit_reg(TCCRO,COM00); // COM00 = 0
    set_bit_reg(TCCRO,COM01); // COM01 = 1

    output_port(DDRB,PB3); // pin PB3 postavljen kao izlazni pin

    OCRO = 0; // duty cycle
}

int main(void){

    inicijalizacija();

    while(1)
    {
        _delay_ms(1000); // kašnjenje od 1 s
    }

    return 0;
}
```

U ovoj vježbi sklop *Timer/Counter0* koristit ćemo u *Fast PWM* načinu rada. U ovom načinu rada sklop *Timer/Counter0* generira *PWM* signal na pinu PB3 (OC0). Prema tablici 8.2 vrijednosti bitova `WGM00` i `WGM01` u `TCCRO` registru moraju se postaviti u vrijednost 1 kako bi sklop *Timer/Counter0* bio konfiguriran u *Fast PWM* načinu rada. Frekvencija *PWM* signala mora biti veća od 3 kHz, a manja od 10 kHz. Prema relaciji (8.6) frekvencija *PWM* signala ovisi o frekvenciji radnog takta `F_CPU` i o djelitelju frekvencije radnog takta. Frekvencija radnog takta je 8 MHz pa je potrebno odabrati djelitelj frekvencije radnog takta za koji će vrijediti da frekvencija bude između 3 i 10 kHz. Za djelitelja frekvencije radnog takta odabrat ćemo 8.

Prema relaciji (8.6) frekvencija *PWM* signala bit će:

$$F_{fPWM} = \frac{F_{CPU}}{PRESCALER \cdot 256} = \frac{8000000}{8 \cdot 256} = 3906,25\text{Hz} = 3,91\text{kHz}. \quad (8.25)$$

Dobivena frekvencija *PWM* signala u relaciji (8.25) veća je od 3 kHz i manja od 10 kHz. Pokušajte u relaciju (8.6) uvrstiti ostale vrijednosti djelitelja frekvencije radnog takta kako bismo ustanovili sve moguće frekvencije *PWM* signala pomoću sklopa *Timer/Counter0*. U programskom kodu 8.16 u funkciji `inicijalizacija()` napravljena je konfiguracija bitova `CS00`, `CS01` i `CS02` u registru `TCCR0` koju ostvaruje djelitelj frekvencije radnog takta vrijednosti 8.

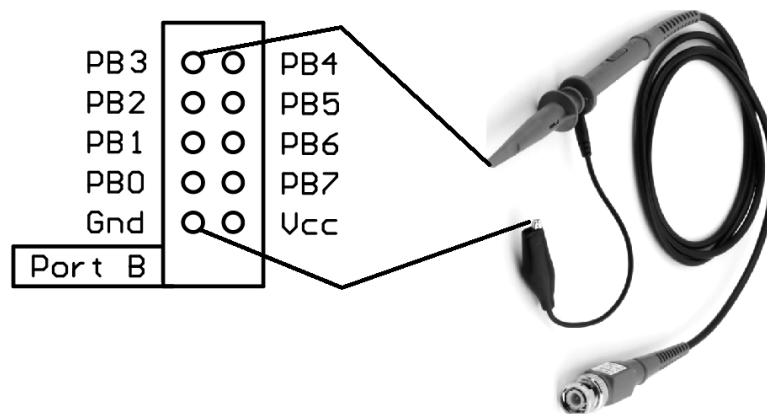
*PWM* signal mora biti neinvertirajući. Prema tablici 8.3 to se postiže postavljanjem bitova `COM00` u vrijednost 0 i `COM01` u vrijednost 1. Ovi bitovi nalaze se u registru `TCCR0` i konfigurirani su u funkciji `inicijalizacija()`. *PWM* signal generira se na pinu PB3 (OC0) pa ga je potrebno konfigurirati kao izlazni pin. Ukoliko zaboravite pin PB3 konfigurirati kao izlazni, na njega se neće generirati *PWM* signal.

Nakon konfiguracije neinvertirajućeg *PWM* signala potrebno je odrediti vrijednost registra `OCR0` za koju će vrijediti da visoko stanje *PWM* signala traje 70 % perioda  $T$ . Omjer  $\frac{T_D}{T}$  u ovom slučaju iznosi 0,7. Prema relaciji (8.9) vrijednost `OCR0` registra mora biti:

$$OCR0 = \frac{T_D}{T} \cdot 255 = 0,7 \cdot 255 = 179. \quad (8.26)$$

U programski kod 8.16 upišite izračunatu vrijednost registra `OCR0`. U `while` petlji nalazi se samo jedna naredba, odnosno kašnjenje od 100 ms. Na prvi pogled čini se da mikrokontroler u ovom slučaju ništa ne radi, no hardverski smo konfigurirali sklop *Timer/Counter0* koji generira *PWM* signal. Ako mikrokontroler koristite samo za generiranje *PWM* signala, tada u funkciji `main` obavezno morate napraviti beskonačnu `while` petlju. Kada ne biste napravili beskonačnu petlju, mikrokontroler ne bi generirao *PWM* signal jer bi završio s izvođenjem `main` funkcije.

Prevedite datoteku `vjezba818.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. *PWM* signal pratit ćemo pomoću osciloskopa. Spajanje sonde osciloskopa na razvojno okruženje s mikrokontrolerom ATmega16 prikazano je na slici 8.5.



Slika 8.5: Spajanje sonde osciloskopa na razvojno okruženje s mikrokontrolerom ATmega16

Na razvojnom okruženju s mikrokontrolerom ATmega16 pinovi pojedinih portova izvučeni su na priključnice s imenom Port A, Port B i Port D. Na slici 8.5 prikazana je priključnica s rasporedom pinova porta B na kojoj se nalazi i napajanje (masa (Gnd) i 5V (Vcc)). Sondu osciloskopa potrebno je spojiti između mase i pina PB3 (slika 8.5). Na osciloskopu prikazite tri perioda *PWM* signala te frekvenciju *PWM* signala.

Zatvorite datoteku `vjezba818.c` i onemogućite prevođenje ove datoteke.



### Vježba 8.1.9

Napravite program koji će na pinu PB3 (OC0) 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 PA5. Na LCD displeju ispišite frekvenciju *PWM* signala i postotak popunjenosti *PWM* signala. Osciloskopom je potrebno pratiti *PWM* signal na PB3 (OC0) pinu.

U projektnom stablu otvorite datoteku `vjezba819.c`. Omogućite prevođenje samo datoteke `vjezba819.c`. Početni sadržaj datoteke `vjezba819.c` prikazan je programskim kodom 8.17.

Programski kod 8.17: Početni sadržaj datoteke `vjezba819.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "timer/timer.h"
#include "ADC/adc.h"

void inicijalizacija(){
    lcd_init();
    adc_init();

    // konfigurirati neinvertirajući PWM signal
}

int main(void){
    inicijalizacija();

    uint32_t ADC5;

    while(1)
    {
        ADC5 = adc_read_10bit(5);

        OCR0 = ADC5 * 255 / 1023;

        lcd_clrscr();
        lcd_home();
        lcd_print("PWM freq:%luHz\n", F_CPU / 8 / 256);
        lcd_print("TD/T:%u%%", ADC5 * 100 / 1023);
        _delay_ms(500);
    }

    return 0;
}
```

U programskom kodu 8.17 u funkciji `inicijalizacija()` napravite konfiguraciju sklopa *Timer/Counter0* u *Fast PWM* načinu rada sukladno uputama. Širinu *PWM* signala potrebno je mijenjati pomoću potenciometra na pinu PA5. Zbog toga je u funkciji `inicijalizacija()` pozvana funkcija `adc_init()` kojom se konfigurira analogno-digitalna pretvorba. Širinu *PWM* signala mijenjat ćemo pomoću registra `OCR0`. Vrijednost analogno-digitalne pretvorbe na pinu PA5 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 PA5 na sljedeći način:

$$\frac{T_D}{T} = \frac{ADC5}{1023}. \quad (8.27)$$

Prema relaciji (8.9), vrijednost `OCR0` registra je:

$$OCR0 = \frac{T_D}{T} \cdot 255 = \frac{ADC5}{1023} \cdot 255. \quad (8.28)$$

Vrijednost `OCR0` registra računat ćemo u `while` petlji svakih 500 ms. Na LCD displeju prikazuje se frekvencija *PWM* signala izračunata prema relaciji (8.6) te postotak popunjenosti *PWM* signala prema relaciji:

$$\frac{T_D}{T} [\%] = \frac{ADC5}{1023} \cdot 100 [\%]. \quad (8.29)$$

Prevedite datoteku `vjezba819.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Na slici 8.5 prikazana je priključnica s rasporedom pinova porta B na kojoj se nalazi i napajanje (masa (`Gnd`) i 5V (`Vcc`)). Sondu osciloskopa potrebno je spojiti između mase i pina PB3 (slika 8.5). 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?

Zatvorite datoteku `vjezba819.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 8.2 Zadaci - tajmeri i brojači

### Zadatak 8.2.1

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje zelene LED diode svakih 15 ms. Zelena LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

### Zadatak 8.2.2

Napravite program u kojem ćete pomoću sklopa *Timer/Counter1* mijenjati stanje zelene LED diode svakih 400 ms. Zelena LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

### Zadatak 8.2.3

Napravite program u kojem ćete pomoću sklopa *Timer/Counter2* mijenjati stanje zelene LED diode svakih 150 ms. Zelena LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

### Zadatak 8.2.4

Napravite program u kojem ćete pomoću sklopa *Timer/Counter0* mijenjati stanje crvene LED diode svakih 15 ms, pomoću sklopa *Timer/Counter1* mijenjati stanje žute LED diode svakih 450 ms, a pomoću sklopa *Timer/Counter2* mijenjati stanje zelene LED diode svakih 30 ms. Crvena LED dioda spojena je na pin PB7, žuta LED dioda spojena je na pin PB6, a zelena LED dioda spojena je na pin PB5. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

### Zadatak 8.2.5

Napravite program koji će omogućiti tzv. beskonačno „trčanje” svih LED dioda svakih 360 ms i to redoslijedom bijela → zelena → žuta → crvena → bijela → ... . Izmjenu stanja LED dioda svakih 360 ms ostvarite pomoću sklopa *Timer/Counter1*. Crvena LED dioda spojena je na digitalni pin PB7, žuta LED dioda spojena je na digitalni pin PB6, zelena LED dioda spojena je na digitalni pin PB5, a bijela LED dioda spojena je na digitalni pin PB4. Istovremeno je na LCD displeju potrebno ispisivati vrijeme rada mikrokontrolera u minutama. Minute generirajte tako da postavite kašnjenje `while` petlje od 60 s.

---

**📌 Zadatak 8.2.6**

Napravite program u kojem će se, ako je pritisnuto tipkalo spojeno na pin PB2, uzimati uzorci s analognog senzora spojenog na pin PA5. Vrijeme uzorkovanja neka iznosi 340 ms. Uzimanje uzoraka ostvarite u prekidnoj rutini `ISR(TIMER1_OVF_vect)`. Vrijednost napona na pinu PA5 ispišite na LCD displej u `while` petlji.

---

**📌 Zadatak 8.2.7**

Napravite program koji će brojati padajuće bridove signala na pinu PB0 (T0) pomoću sklopa *Timer/Counter0*. Broj padajućih bridova potrebno je ispisati na LCD displeju. Maksimalni broj impulsa koji može pristići na pin PB0 je 50000.

---

**📌 Zadatak 8.2.8**

Napravite program koji će na pinu PB3 (OC0) generirati neinvertirajući *PWM* signal pomoću sklopa *Timer/Counter0*. Frekvenciju *PWM* signala namjestite tako da bude veća od 10 kHz. Visoko stanje impulsa *PWM* signala postavite na 20 % perioda *T*. Osciloskopom je potrebno pratiti *PWM* signal na PB3 (OC0) pinu.

---

**📌 Zadatak 8.2.9**

Napravite program koji će na pinu PB3 (OC0) generirati invertirajući *PWM* signal pomoću sklopa *Timer/Counter0*. Frekvenciju *PWM* signala namjestite tako da bude veća od 10 kHz. Omogućite promjenu širine impulsa *PWM* signala pomoću potencijometra spojenog na pin PA5. Na LCD displeju ispišite frekvenciju *PWM* signala i postotak popunjenosti *PWM* signala. Osciloskopom je potrebno pratiti *PWM* signal na PB3 (OC0) pinu.

---

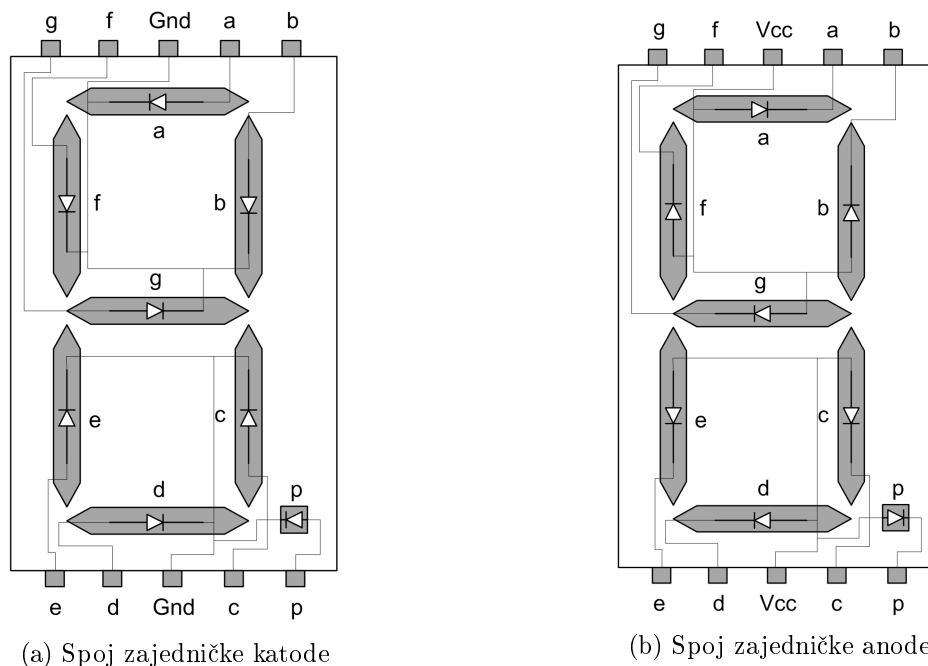




## Poglavlje 9

# Numerički displej

Numerički displej najčešće se koristi za prikaz brojeva pomoću segmenata. Svaki segment je 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.



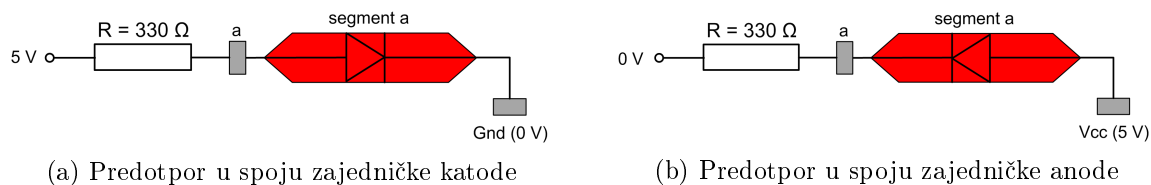
Slika 9.1: Numerički displej sa sedam segmenata za prikaz brojeva i jednim segmentom za prikaz decimalne točke

Imena segmenata za prikaz brojeva su **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).

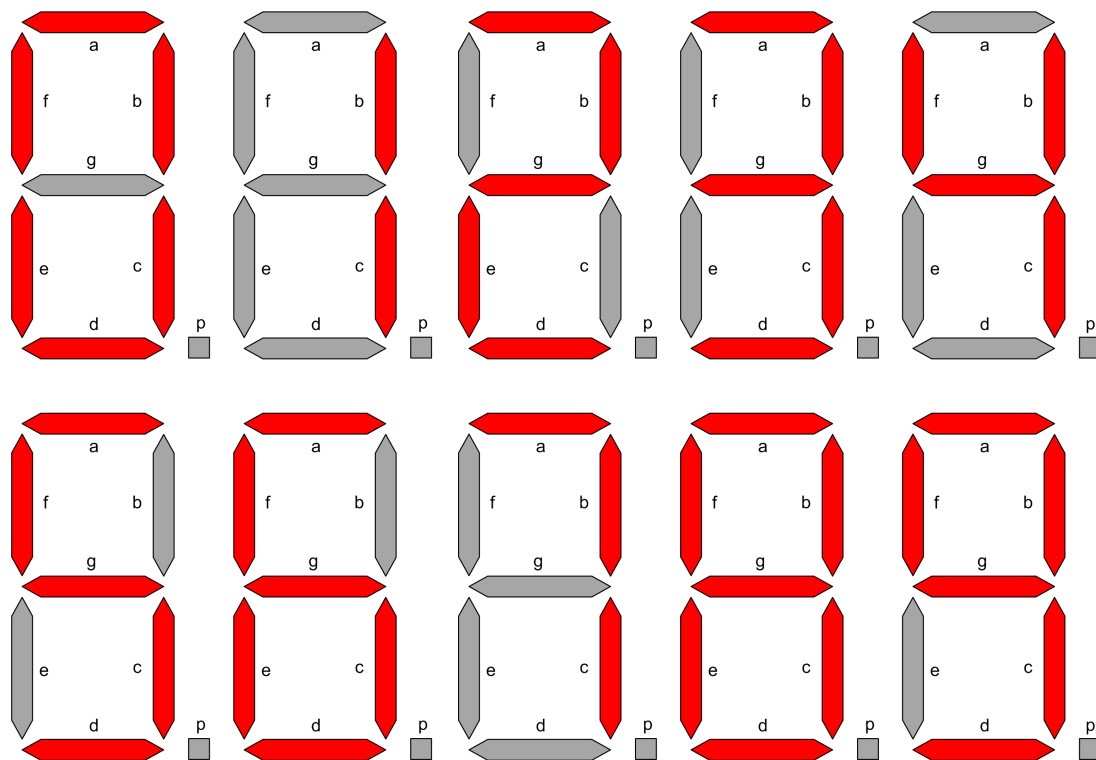
U spoju zajedničke katode sve katode LED dioda koje čine segmente spojene su na zajednički potencijal  $Gnd$  (0 V). U tom slučaju jedan segment uključujemo tako da na pin tog segmenta preko predotpora od  $330 \Omega$  dovedemo potencijal od 5 V (slika 9.2a).

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 mikrokontrolera 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 mikrokontrolera uključuje segment, a nisko stanje isključuje segment. U spoju zajedničke anode nisko stanje na digitalnom pinu mikrokontrolera uključuje segment, a visoko stanje isključuje segment.



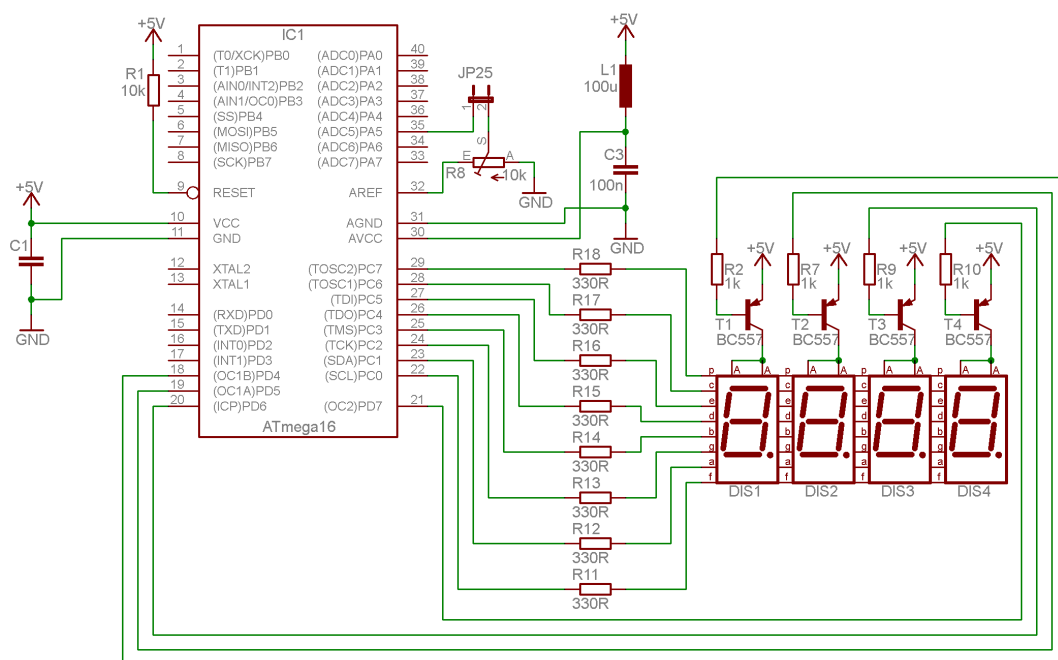
Slika 9.3: Prikaz uključenih i isključenih segmenata za znamenke dekadskog sustava

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

## 9.1 Vježbe - numerički displej

U narednim vježbama ćemo prezentirati četveroznamenkaste brojeve na četiri numerička displeja. Za prikaz znamenke na jednom numeričkom displeju potrebno je osam digitalnih pinova. Za prikaz četiri znamenke na četiri numerička displeja potrebna su 32 digitalna pina mikrokontrolera. Na početku ovog udžbenika naveli smo kako mikrokontroler ATmega16 ima 32 digitalna pina. Kada bi za prikaz četiri znamenke na četiri numerička displeja koristili 32 digitalna pina mikrokontrolera, mikrokontroler ne bi mogao obavljati nikakve druge funkcije osim prikaza četveroznamenkastog broja. Postoji bolje rješenje za spajanje numeričkih displeja na mikrokontroler s manjom potrošnjom digitalnih pinova. Shema spajanja četiri numerička displeja i potenciometra na mikrokontroler ATmega16 prikazana je na slici 9.4. Na shemi se nalaze četiri numerička displeja u spoju zajedničke anode sa sedam segmenta za prikaz znamenaka i jednim segmentom za prikaz decimalne točke.



Slika 9.4: Shema spajanja četiri numerička displeja i potenciometra na mikrokontroler ATmega16

Kako bi smanjili potrošnju digitalnih pinova za prikaz četiri znamenke, iskoristit ćemo tromost oka. Segmente a, b, c, d, e, f, g i p četiri numerička displeja spojiti ćemo zajedno, a napon Vcc na zajedničku anodu pojedinog numeričkog displeja dovest ćemo pomoću PNP tranzistora oznake BC557 (slika 9.4). Četiri numerička displeja upravljana su pomoću četiri PNP tranzistora koji rade kao sklopke. Ovi tranzistori u stanju su vođenja ako na bazu tranzistora dovedemo nisko stanje (0 V). Na primjer, ako je tranzistor T1 na slici 9.4 u stanju vođenja, tada će segment a na numeričkom displeju DISP1 biti uključen ako je na tom segmentu nisko stanje (0 V). Upravljački tranzistori spojeni su na port D mikrokontrolera ATmega16 na sljedeći način:

- tranzistor T1 koji omogućuje ispis znamenaka na numeričkom displeju DISP1 spojen je na digitalni pin PD4,
- tranzistor T2 koji omogućuje ispis znamenaka na numeričkom displeju DISP2 spojen je na digitalni pin PD5,
- tranzistor T3 koji omogućuje ispis znamenaka na numeričkom displeju DISP3 spojen je na digitalni pin PD6,
- tranzistor T4 koji omogućuje ispis znamenaka na numeričkom displeju DISP4 spojen je na digitalni pin PD7.

Segmenti numeričkih displeja spojeni su na port C mikrokontrolera ATmega16 na sljedeći način:

- segment a spojen je na digitalni pin PC1,
- segment b spojen je na digitalni pin PC3,
- segment c spojen je na digitalni pin PC6,
- segment d spojen je na digitalni pin PC4,
- segment e spojen je na digitalni pin PC5,
- segment f spojen je na digitalni pin PC0,
- segment g spojen je na digitalni pin PC2,
- segment p spojen je na digitalni pin PC7.

Četveroznamenkasti broj na numeričkim displejima prikazat ćemo tako da brzo izmjenjujemo sljedeće korake:

1. Za prikaz znamenke na numeričkom displeju DISP1 tranzistor T1 postavite u stanje vođenja tako da na bazu tranzistora dovedete nisku razinu pomoću pina PD4. Na port C postavite kombinaciju koja će uključiti segmente za prikaz željene znamenke. Pričekajte 5 ms.
2. Za prikaz znamenke na numeričkom displeju DISP2 tranzistor T2 postavite u stanje vođenja tako da na bazu tranzistora dovedete nisku razinu pomoću pina PD5. Na port C postavite kombinaciju koja će uključiti segmente za prikaz željene znamenke. Pričekajte 5 ms.
3. Za prikaz znamenke na numeričkom displeju DISP3 tranzistor T3 postavite u stanje vođenja tako da na bazu tranzistora dovedete nisku razinu pomoću pina PD6. Na port C postavite kombinaciju koja će uključiti segmente za prikaz željene znamenke. Pričekajte 5 ms.
4. Za prikaz znamenke na numeričkom displeju DISP4 tranzistor T4 postavite u stanje vođenja tako da na bazu tranzistora dovedete nisku razinu pomoću pina PD7. Na port C postavite kombinaciju koja će uključiti segmente za prikaz željene znamenke. Pričekajte 5 ms.

Prethodna četiri koraka potrebno je neprestano ponavljati. Zbog tromosti oka nismo u stanju vidjeti brze promjene na numeričkim displejima pa će nam četveroznamenasti broj izgledati statično. Na ovaj je način umjesto 32 digitalna pina potrebno koristiti samo 12 digitalnih pinova (8 za upravljanje segmentima i 4 za upravljanje numeričkim displejima). Kombinacija bitova registra **PORTC** za prikaz znamenaka dekadskog sustava na numeričkim displejima sa sheme na slici 9.4 prikazana je u tablici 9.2.

Tablica 9.2: Stanja pojedinih segmenata za prikaz jedne znamenke dekadskog sustava prema shemi prikazanoj na slici 9.4

Znamenka	Segmenti								PORTC (hex)
	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
	p	c	e	d	b	g	a	f	
0	1	0	0	0	0	1	0	0	0x84
1	1	0	1	1	0	1	1	1	0xB7
2	1	1	0	0	0	0	0	1	0xC1
3	1	0	1	0	0	0	0	1	0xA1
4	1	0	1	1	0	0	1	0	0xB2
5	1	0	1	0	1	0	0	0	0xA8
6	1	0	0	0	1	0	0	0	0x88
7	1	0	1	1	0	1	0	1	0xB5
8	1	0	0	0	0	0	0	0	0x80
9	1	0	1	0	0	0	0	0	0xA0

U zadnjem stupcu tablice 9.2 prikazane su heksadecimalne vrijednosti koje se mogu upisati u registar **PORTC** ako na numeričkom displeju želimo prikazati željenu znamenku. Pokušajte samostalno definirati heksadecimalne vrijednosti registra **PORTC** za slova A, B, C, D, E i F.

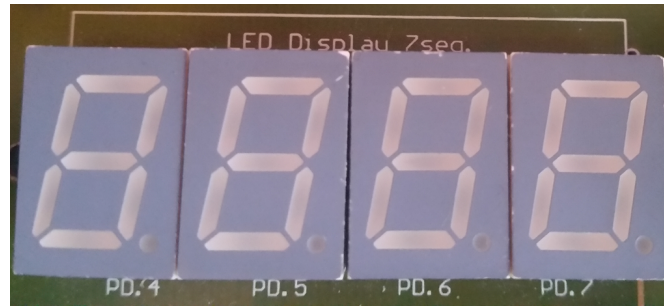
Ako je aktivan tranzistor T1 (pin PD4 je u niskom stanju) i ako u registar **PORTC** upišemo heksadecimalnu vrijednost **0xA0**, tada će prema tablici 9.2 na numeričkom displeju DISP1 biti prikazana znamenka 9.

Ako je aktivan tranzistor T3 (pin PD6 je u niskom stanju) i ako u registar **PORTC** upišemo heksadecimalnu vrijednost **0xB7**, tada će prema tablici 9.2 na numeričkom displeju DISP3 biti prikazana znamenka 1.

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

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

Na razvojnom okruženju sa slike 3.1 odspojite LCD displej. Na razvojno okruženje postavite četiri numerička displeja prema slici 9.5.



Slika 9.5: Montaža numeričkih displeja na razvojno okruženje s mikrokontrolerom ATmega16



### Vježba 9.1.1

Napravite program u kojem ćete na numeričkim displejima prikazati broj 5555. Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

U projektom stablu otvorite datoteku `vjezba911.c`. Omogućite prevođenje samo datoteke `vjezba911.c`. Početni sadržaj datoteke `vjezba911.c` prikazan je programskim kodom 9.1.

Programski kod 9.1: Početni sadržaj datoteke `vjezba911.c`

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

void inicijalizacija(){

    // pinovi PD7, PD6, PD5 i PD4 postavljeni kao izlazni
    DDRD |= (1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4);
    // pinovi PD7, PD6, PD5 i PD4 postavljeni u nisko stanje
    // omogućen je prikaz znamenaka na svim numeričkim displejima
    PORTD &= ~(1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4));

    DDRC = 0xFF; // svi pinovi na port C postavljeni kao izlazni
    PORTC = 0xFF; // inicijalno uključeni svi pinovi/isključeni segmenti
}

int main(void){

    inicijalizacija();

    PORTC = 0xA8; // znamenka 5 prema tablici 9.2

    return 0;
}
```

Prikaz znakova na pojedinom numeričkom displeju omogućuje se pinovima PD7, PD6, PD5 i PD4 koje je u registru `DDRD` potrebno konfigurirati kao izlazne pinove. Tranzistori kojima se omogućuje ispis na pojedinom numeričkom displeju su PNP tipa. Ovi tranzistori bit će u stanju vođenja ako na bazu tranzistora preko pinova PD7, PD6, PD5 i PD4 dovedemo nisko stanje. U ovoj vježbi na svim ćemo numeričkim displejima prikazivati istu vrijednost pa je pinove PD7, PD6, PD5 i PD4 potrebno postaviti u nisko stanje.

Segmenti numeričkih displeja spojeni su na port C pa je u registar `DDRC` potrebno upisati vrijednost `0xFF` što će osigurati da svi pinovi na portu C budu izlazni. Navedene konfiguracije ostvarene su u funkciji `inicijalizacija()` u programskom kodu 9.1. Na numeričkim displejima potrebno je ispisati 5555. Prema tablici 9.2 u registar `PORTC` potrebno je upisati heksadeci-

malnu vrijednost 0xA8. Istu vrijednost možemo zapisati i binarno u registar `PORTC` naredbom `PORTC = 0b10101000;`.

Prevedite datoteku `vjezba911.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

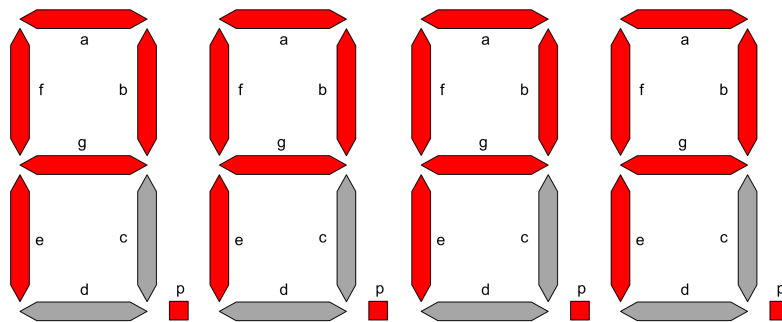
Pokušajte na numeričkom displeju ispisati broj 1111. Ponovno prevedite datoteku `vjezba911.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba911.c` i onemogućite prevođenje ove datoteke.



## Vježba 9.1.2

Napravite program u kojem ćete na numeričkim displejima prikazati znakove sa slike 9.6.



Slika 9.6: Znakovi za prikaz na numeričkim displejima

Schema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

U projektnom stablu otvorite datoteku `vjezba912.c`. Omogućite prevođenje samo datoteke `vjezba912.c`. Početni sadržaj datoteke `vjezba912.c` prikazan je programskim kodom 9.2.

Programski kod 9.2: Početni sadržaj datoteke `vjezba912.c`

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

void inicijalizacija(){

    // pinovi PD7, PD6, PD5 i PD4 postavljeni kao izlazni
    DDRD |= (1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4);
    // pinovi PD7, PD6, PD5 i PD4 postavljeni u nisko stanje
    // omogućen je prikaz znamenaka na svim numeričkim displejima
    PORTD &= ~(1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4));

    DDRC = 0xFF; // svi pinovi na port C postavljeni kao izlazni
    PORTC = 0xFF; // inicijalno uključeni svi pinovi/isključeni segmenti
}

int main(void){


    inicijalizacija();

    //u PORTC upisati vrijednost koja će ostvariti prikaz zadanih znakova

return 0;
}
```

Inicijalizacija mikrokontrolera u ovoj vježbi ista je kao i u vježbi 9.1.1. Znakovi sa slike 9.6 zahtijevaju da segmenti **a**, **b**, **e**, **f**, **g** i **p** budu uključeni. Definirani znak sa slike 9.6 prikazan je u tablici 9.3. U programski kod 9.2 u registar **PORTC** upišite heksadecimalnu vrijednost **0x50**.

Tablica 9.3: Definiranje znaka sa slike 9.6

Znak	Segmenti								PORTC (hex)
	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	
	p	c	e	d	b	g	a	f	
	0	1	0	1	0	0	0	0	0x50

Prevedite datoteku `vjezba912.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba912.c` i onemogućite prevođenje ove datoteke.



### Vježba 9.1.3

Napravite program u kojem ćete neprestano ponavljati sljedeća četiri koraka:

1. na numeričkom displeju DISP1 prikažite broj 1 te pričekajte jednu sekundu,
2. na numeričkom displeju DISP2 prikažite broj 2 te pričekajte jednu sekundu,
3. na numeričkom displeju DISP3 prikažite broj 3 te pričekajte jednu sekundu,
4. na numeričkom displeju DISP4 prikažite broj 4 te pričekajte jednu sekundu.

Testirajte prethodni program, a zatim vrijeme čekanja između prikaza znakova na dva numerička displeja postavite na 5 ms. Nakon toga napravite dio programa koji će, kada se pritisne tipkalo spojeno na pin PB0 postaviti kašnjenje `while` petlje od dvije sekunde. Što se događa kada pritisnete tipkalo spojeno na pin PB0 i kako riješiti problem koji se pojavio? Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

U projektnom stablu otvorite datoteku `vjezba913.c`. Omogućite prevođenje samo datoteke `vjezba913.c`. Početni sadržaj datoteke `vjezba913.c` prikazan je programskim kodom 9.3.

Programski kod 9.3: Početni sadržaj datoteke `vjezba913.c`

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

void inicijalizacija(){

    // pinovi PD7, PD6, PD5 i PD4 postavljeni kao izlazni
    DD RD |= (1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4);
    // pinovi PD7, PD6, PD5 i PD4 postavljeni u visoko stanje
    // onemogućen je prikaz znamenaka na svim numeričkim displejima
    PORTD |= ((1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4));

    DD RC = 0xFF; // svi pinovi na port C postavljeni kao izlazni
    PORTC = 0xFF; // inicijalno uključeni svi pinovi/isključeni segmenti
}
```



```
int main(void){
    inicijalizacija();

    while (1)
    {

        set_port(PORTD, PD7, 1); // onemogući DISP4
        set_port(PORTD, PD4, 0); // omogući DISP1
        PORTC = 0xB7; // 1
        _delay_ms(1000);

        set_port(PORTD, PD4, 1); // onemogući DISP1
        set_port(PORTD, PD5, 0); // omogući DISP2
        PORTC = 0xC1; // 2
        _delay_ms(1000);

        set_port(PORTD, PD5, 1); // onemogući DISP2
        set_port(PORTD, PD6, 0); // omogući DISP3
        PORTC = 0xA1; // 3
        _delay_ms(1000);

        set_port(PORTD, PD6, 1); // onemogući DISP3
        set_port(PORTD, PD7, 0); // omogući DISP4
        PORTC = 0xB2; // 4
        _delay_ms(1000);
    }

    return 0;
}
```

U funkciji `inicijalizacija()` u programskom kodu 9.3 konfigurirani su pinovi PD7, PD6, PD5 i PD4 kao izlazni pinovi te im je početno stanje postavljeno u visoko kako bi početno svi numerički displeji bili onemogućeni. U `while` petlji potrebno je redom:

- omogućiti prikaz na numeričkom displeju DISP1 (nisko stanje na pinu PD4), a onemogućiti prikaz na numeričkom displeju DISP4 (visoko stanje na pinu PD7),
- ispisati broj 1 na numeričkom displeju DISP1 tako da u registar `PORTC` upišete vrijednost `0xB7` i pričekati jednu sekundu,
- omogućiti prikaz na numeričkom displeju DISP2 (nisko stanje na pinu PD5), a onemogućiti prikaz na numeričkom displeju DISP1 (visoko stanje na pinu PD4),
- ispisati broj 2 na numeričkom displeju DISP2 tako da u registar `PORTC` upišete vrijednost `0xC1` i pričekati jednu sekundu,
- omogućiti prikaz na numeričkom displeju DISP3 (nisko stanje na pinu PD6), a onemogućiti prikaz na numeričkom displeju DISP2 (visoko stanje na pinu PD5),
- ispisati broj 3 na numeričkom displeju DISP3 tako da u registar `PORTC` upišete vrijednost `0xA1` i pričekati jednu sekundu,
- omogućiti prikaz na numeričkom displeju DISP4 (nisko stanje na pinu PD7), a onemogućiti prikaz na numeričkom displeju DISP3 (visoko stanje na pinu PD6),
- ispisati broj 4 na numeričkom displeju DISP4 tako da u registar `PORTC` upišete vrijednost `0xB2` i pričekati jednu sekundu.

Prevedite datoteku `vjezba913.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite argument funkcije `_delay_ms` iz 1000 u 5 kako biste promijenili kašnjenje iz jedne sekunde u 5 ms. Ponovno prevedite datoteku `vjezba913.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Numerički displeji sada prikazuju broj 1234 statično. Zašto je to tako?

U nastavku ćemo koristiti pin PB0 kojim ćemo generirati dodatno kašnjenje od dvije sekunde. U funkciji `inicijalizacija()` konfigurirajte pin PB0 kao ulazni i uključite pritezni otpornik na tom pinu. U `while` petlju dodajte programski kod 9.4 koji unosi kašnjenje u petlju ako je pritisnuto tipkalo spojeno na pin PB0.

Programski kod 9.4: Kašnjenje od dvije sekunde ako je pritisnuto tipkalo spojeno na pin PB0

```
if (get_pin(PINB, PB0) == 0)
{
    _delay_ms(2000);
}
```

Prevedite datoteku `vjezba913.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Što se događa kada pritisnete tipkalo spojeno na pin PB0 i kako riješiti problem koji se pojavio?

Zbog kašnjenja od dvije sekunde mikrokontroler dvije sekunde ne radi ništa i na numeričkom displeju će biti prikazan samo broj 4, odnosno zadnji broj koji je prikazan. Ovaj problem možemo riješiti pomoću tajmera. Potrebno je konfigurirati tajmer koji će prekidnu rutinu pozivati svakih 5 ms. U tom slučaju, bez obzira na kašnjenje u `while` petlji, ispis na numeričkim displejima bit će statičan.

Zatvorite datoteku `vjezba913.c` i onemogućite prevođenje ove datoteke.



#### Vježba 9.1.4

Napravite program koji će na numeričkim displejima ispisati broj 1234 pomoću tajmera. Napravite dio programa koji će, kada se pritisne tipkalo spojeno na pin PB0 postaviti kašnjenje `while` petlje od dvije sekunde. Što se događa kada pritisnete tipkalo spojeno na pin PB0? Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

U projektnom stablu otvorite datoteku `vjezba914.c`. Omogućite prevođenje samo datoteke `vjezba914.c`. Početni sadržaj datoteke `vjezba914.c` prikazan je programskim kodom 9.5.

Programski kod 9.5: Početni sadržaj datoteke `vjezba914.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "timer/timer.h"
#include <avr/interrupt.h>
#include "SSD/SSD.h"
#include <util/delay.h>

uint8_t brojac;

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 0;

    switch (++brojac)
    {
        case 1:
            set_port(PORTD, PD7, 1); // onemogući DISP4
            set_port(PORTD, PD4, 0); // omogući DISP1
            PORTC = znakovi[1];
            break;
    }
}
```

```

        case 2:
            set_port(PORTD, PD4, 1); // onemogući DISP1
            set_port(PORTD, PD5, 0); // omogući DISP2
            PORTC = znakovi[2];
            break;
        case 3:
            set_port(PORTD, PD5, 1); // onemogući DISP2
            set_port(PORTD, PD6, 0); // omogući DISP3
            PORTC = znakovi[3];
            break;
        case 4:
            set_port(PORTD, PD6, 1); // onemogući DISP3
            set_port(PORTD, PD7, 0); // omogući DISP4
            PORTC = znakovi[4];
            brojac = 0;
            break;
    }
}

void inicijalizacija(){

    // pinovi PD7, PD6, PD5 i PD4 postavljeni kao izlazni
    DDRD |= (1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4);
    // pinovi PD7, PD6, PD5 i PD4 postavljeni u visoko stanje
    // onemogućen je prikaz znamenaka na svim numeričkim displejima
    PORTD |= ((1 << PD7) | (1 << PD6) | (1 << PD5) | (1 << PD4));

    DDRC = 0xFF; // svi pinovi na port C postavljeni kao izlazni
    PORTC = 0xFF; // inicijalno uključeni svi pinovi/isključeni segmenti

    input_port(DDRB, PBO); // pin PBO postavljen kao ulazni pin
    set_port(PORTB, PBO, 1); // uključen pritezni otpornik na PBO

    //normalan način rada - 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
    // F_CPU / 8
    reset_bit_reg(TCCR1B, CS10); // CS10 = 0
    set_bit_reg(TCCR1B, CS11); // CS11 = 1
    reset_bit_reg(TCCR1B, CS12); // CS12 = 0
    // omogući prekid TOV1
    set_bit_reg(TIMSK, TOIE1); // TOIE1 = 1

    sei(); // globalno omogućavanje prekida
    TCNT1 = 0;
}

int main(void){

    inicijalizacija();

    while(1)
    {
        // kašnjenje od 2 sekunde ako se pritisne tipkalo spojeno na pin PBO
    }

    return 0;
}

```

U funkciji `inicijalizacija()` u programskom kodu 9.5 konfigurirani su pinovi PD7, PD6, PD5 i PD4 kao izlazni pinovi te im je početno stanje postavljeno u visoko kako bi početno svi

numerički displeji bili onemogućeni. Nadalje, port C je postavljen kao izlazni port i početno stanje svih pinova je visoko. Pin PB0 konfiguriran je kao ulazni i uključen je pritezni otpornik.

U ovoj vježbi koristit ćemo sklop *Timer/Counter1* kao tajmer u normalnom načinu rada za prozivanje pojedinog numeričkog displeja. Prema tablicama 47 i 48 u literaturi [1] u funkciji `inicijalizacija()` konfiguriran je sklop *Timer/Counter1* kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 8. Prekid kojeg izaziva preljev u registru `TCNT1` omogućuje se upisivanjem broja 1 na mjesto bita `TOIE1` u registru `TIMSK`. Za globalno omogućavanje prekida pozvana je makronaredba `sei()`.

Vrijeme između dva poziva prekidne rutine mora biti 5 ms ( $t_{T1} = 0,005$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 0,005 \cdot \frac{8000000}{8} = 60536. \quad (9.1)$$

Upišite početnu vrijednost registra `TCNT1` u programski kod 9.5 na odgovarajuća mjesta. Obratite pažnju na uključena zaglavlja u programskom kodu 9.5. U prekidnoj rutini `ISR(TIMER1_OVF_vect)` pomoću `switch case` bloka omogućujemo i onemogućujemo redom prikaz na numeričkim displejima. Kako bismo olakšali prikaz brojeva u zaglavlju `ssd.h`, definirali smo polje s deset elemenata prema tablici 9.2. Polje se zove `znakovi`, a broj iz polja kojeg želimo prikazati adresiramo pomoću njega samoga. Na primjer, ako želimo prikazati broj 0, tada ćemo pozvati element polja `znakovi[0]`. U programskom kodu 9.5 naredbom `#include "SSD/SSD.h"` uključili smo zaglavlje `ssd.h` u kojem je definirano polje `znakovi`.

U `while` petlju dodajte programski kod 9.4 koji unosi kašnjenje u petlju ako je pritisnuto tipkalo spojeno na pin PB0.

Prevedite datoteku `vjezba914.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Što se sada događa kada pritisnete tipkalo spojeno na pin PB0?

Ispišite na numeričkim displejima broj 3145. Ponovno prevedite datoteku `vjezba914.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba914.c` i onemogućite prevođenje ove datoteke.



### Vježba 9.1.5

Napravite program koji će na numeričkim displejima ispisati napon na potencijometru spojenom na pin PA5. Napon ispišite na tri decimalna mjesta. Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

U projektnom stablu otvorite datoteku `vjezba915.c`. Omogućite prevođenje samo datoteke `vjezba915.c`. Početni sadržaj datoteke `vjezba915.c` prikazan je programskim kodom 9.6.

Programski kod 9.6: Početni sadržaj datoteke `vjezba915.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "timer/timer.h"
#include <avr/interrupt.h>
#include "SSD/SSD.h"
#include <util/delay.h>
#include "ADC/adc.h"
```

```

uint8_t brojac;
uint8_t z1, z2, z3, z4; // znamenke za prikaz

ISR(TIMER1_OVF_vect){ // prekidna rutina za timer1
    TCNT1 = 60536; // početna vrijednost registra za 50 ms

    switch (++brojac)
    {
        case 1:
            set_port(PORTD, PD7, 1); // onemogući DISP4
            set_port(PORTD, PD4, 0); // omogući DISP1
            PORTC = znakovi[z1];
            set_port(PORTC, PC7, 0); // uključenje točke
            break;
        case 2:
            set_port(PORTD, PD4, 1); // onemogući DISP1
            set_port(PORTD, PD5, 0); // omogući DISP2
            PORTC = znakovi[z2];
            break;
        case 3:
            set_port(PORTD, PD5, 1); // onemogući DISP2
            set_port(PORTD, PD6, 0); // omogući DISP3
            PORTC = znakovi[z3];
            break;
        case 4:
            set_port(PORTD, PD6, 1); // onemogući DISP3
            set_port(PORTD, PD7, 0); // omogući DISP4
            PORTC = znakovi[z4];
            brojac = 0;
            break;
    }
}

// kopirajte funkciju inicijalizacija iz datoteke vjezba914.c
// dodajte u funkciju inicijalizacija funkciju adc_init();

int main(void){

    inicijalizacija();

    uint32_t ADC5;
    uint32_t VADC5;

    while(1)
    {
        ADC5 = adc_read_10bit(5);

        VADC5 = ADC5 * 5000 / 1023;

        z1 = VADC5 / 1000;
        z2 = (VADC5 / 100) % 10;
        z3 = (VADC5 / 10) % 10;
        z4 = VADC5 % 10;
    }

    return 0;
}

```

U ovoj vježbi potrebno je prezentirati napon potencijometra na pinu PA5 pomoću numeričkih displeja. U programski kod 9.6 iskopirajte funkciju `inicijalizacija` iz datoteke `vjezba914.c`. Dodatno, u funkciji `inicijalizacija()` pozovite funkciju `adc_init()` kojom ćete konfigurirati analogno-digitalnu pretvorbu.

Napon koji ćemo prikazivati na numeričkim displejima mora biti prezentiran na tri decimalna

mjesta. Ako broj ima tri decimalna mjesta, tada je taj broj realan. Iz realnog broja moguće je izvući znamenke prije i poslije decimalne točke, ali to je previše zahtjevno za mikrokontroler u smislu izvođenja operacija s realnim brojevima. Za mikrokontroler najpovoljnija je cjelobrojna aritmetika. U tom smislu, napon možemo pomnožiti s 1000 i dobiti četveroznamenkasti cijeli broj. Taj cijeli broj lako je rastaviti na znamenke tisućica, stotica, desetica i jedinica. Budući da znamo da smo broj pomnožili s 1000, decimalnu ćemo točku postaviti nakon prve znamenke.

Analogno-digitalnom pretvorbom na pinu PA5 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]. Relacija za izračun intervala [0, 5000] je:

$$V_{ADC5} = \frac{ADC5}{1023} \cdot 5000. \quad (9.2)$$

U relaciji (9.2) dobili smo napon za 1000 puta veći od stvarnoga. Želimo li prezentirati stvarni napon na numeričkim displejima, na rezultatu pretvorbe prema relaciji (9.2) decimalnu točku pomičemo za tri mjesta ulijevo. Prema tome, decimalna točka dolazi iza prve znamenke. Rezultat  $V_{ADC5}$  potrebno je rastaviti na znamenke tisućica, stotica, desetica i jedinica. Znamenke smo u programskom kodu 9.6 redom nazvali:

- **z1** - znamenka tisućica koja se dobije cjelobrojnim dijeljenjem varijable  $V_{ADC5}$  s 1000 (npr.  $z1 = 5123/1000 = 5$ ),
- **z2** - znamenka stotica koja se dobije tako da se varijabla  $V_{ADC5}$  podijeli sa 100, a zatim se izračuna ostatak pri cjelobrojnem dijeljenju s 10 (npr.  $5123/100 = 51$ ,  $z2 = 51\%10 = 1$ )
- **z3** - znamenka desetica koja se dobije tako da se varijabla  $V_{ADC5}$  podijeli s 10, a zatim se izračuna ostatak pri cjelobrojnem dijeljenju s 10 (npr.  $5123/10 = 512$ ,  $z3 = 512\%10 = 2$ ),
- **z4** - znamenka jedinica koja se dobije tako da se izračuna ostatak cjelobrojnog djeljenja varijable  $V_{ADC5}$  s 10 (npr.  $z4 = 5123\%10 = 3$ ).

U prekidnoj rutini `ISR(TIMER1_OVF_vect)` ispisuju se znakovi u `switch case` bloku. U prvom slučaju ispisujemo znamenku `z1` i nakon nje uključujemo decimalnu točku tako da u nisko stanje postavimo pin PC7 naredbom `set_port(PORTC,PC7,0);`. Na portu C prikazujemo znakove koje smo definirali u zaglavlju `ssd.h`. Znamenke `z1`, `z2`, `z3` i `z4` na portu C prikazujemo tako da pozivamo elemente polja `znakovi`. Na primjer, za znamenku `z1` pozivamo element polja `znakovi[z1]`. Ako je `z1 = 2`, tada će se pozvati element polja `znakovi[2]` gdje je definiran upravo znak 2 prema tablici 9.2.

Prevedite datoteku `vjezba915.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba915.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 9.2 Zadaci - numerički displej

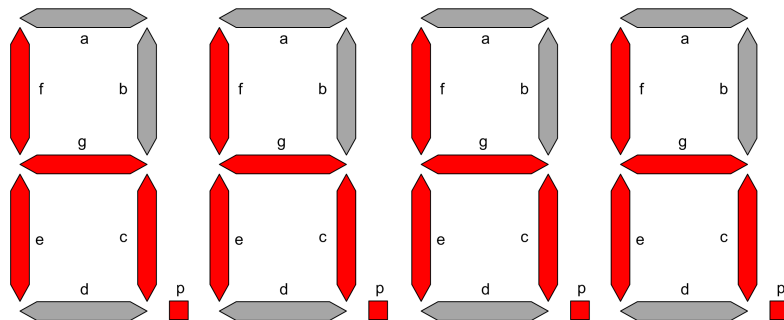
### Zadatak 9.2.1

Napravite program u kojem ćete na numeričkim displejima prikazati broj 0000. Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

---

### Zadatak 9.2.2

Napravite program u kojem ćete na numeričkim displejima prikazati znakove sa slike 9.7.



Slika 9.7: Znakovi za prikaz na numeričkim displejima

Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

---

### Zadatak 9.2.3

Napravite program u kojem ćete neprestano ponavljati sljedeća četiri koraka:

1. na numeričkom displeju DISP1 prikažite broj 9 te pričekajte jednu sekundu,
2. na numeričkom displeju DISP2 prikažite broj 8 te pričekajte jednu sekundu,
3. na numeričkom displeju DISP3 prikažite broj 7 te pričekajte jednu sekundu,
4. na numeričkom displeju DISP4 prikažite broj 6 te pričekajte jednu sekundu.

Testirajte prethodni program, a zatim vrijeme čekanja između prikaza znakova na dva numerička displeja postavite na 5 ms. Nakon toga napravite dio programa koji će, kada se pritisne tipkalo spojeno na pin PB0 postaviti kašnjenje `while` petlje od dvije sekunde. Što se događa kada pritisnete tipkalo spojeno na pin PB0 i kako riješiti problem koji se pojavio? Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

---

### Zadatak 9.2.4

Napravite program koji će na numeričkim displejima ispisati broj 9876 pomoću tajmera. Napravite dio programa koji će, kada se pritisne tipkalo spojeno na pin PB0, postaviti kašnjenje `while` petlje od dvije sekunde. Što se događa kada pritisnete tipkalo spojeno na pin PB0? Shema

spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

---

### **Zadatak 9.2.5**

Napravite program koji će na numeričkim displejima ispisati otpor potencijometra spojenog na pin PA5. Shema spajanja numeričkih displeja na mikrokontroler ATmega16 prikazana je na slici 9.4.

---



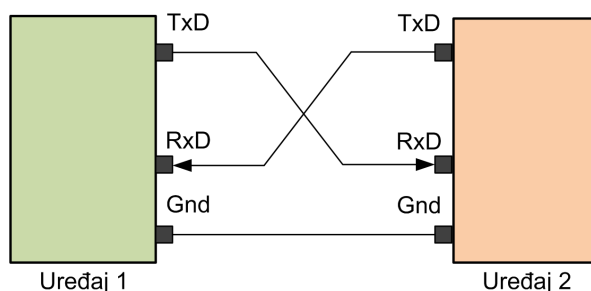
## Poglavlje 10

# Univerzalna asinkrona serijska komunikacija

Univerzalnu asinkronu serijsku komunikaciju omogućuje sklopovlje UART (eng. *Universal Asynchronous Receiver Transmitter*). Ovo sklopovlje koristi dva pina:

- RxD - pin za primanje podataka koji je povezan s primateljom (eng. *Receiver*),
- TxD - pin za slanje podataka koji je povezan s pošiljateljom (eng. *Transmitter*).

Pomoću RxD i TxD pinova omogućena je dvosmjerna (eng. *full-duplex*) komunikacija, što znači da dva uređaja mogu istovremeno i slati i primiti podatke pomoću dvije žice. Razlog tome je 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 je uvijek 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 zajedno spojene kako bi uređaji imali isti referentni potencijal.



Slika 10.1: Shema povezivanja dvaju uređaja sa sklopovljem UART

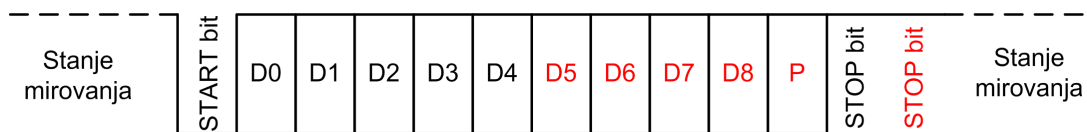
Komunikacija između dva sklopovlja UART je asinkrona, što znači da ne postoji izvor takta koji sinkronizira prijenos podataka. Zbog asinkrone serijske komunikacije dva uređaja sa sklopovljem UART koji razmjenjuju podatke moraju imati jednaku konfiguraciju parametara. Parametri sklopovlja UART su [4]:

- Brzina prijenosa podataka (eng. *Baude Rate*) - sklopovlje UART ima registar kojime se konfigurira brzina prijenosa podataka u bitovima po sekundi (b/s). Često korištene brzine prijenosa su od 9600 do 115200 b/s. Brzina prijenosa ovisi o radnom taktu uređaja

(mikrokontrolera, 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 jedan bajt (B) podataka.
- Paritetni bit (eng. *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 dva uređaja, oba pina (RxD i TxD) su u stanju logičke jedinice. To stanje je stanje mirovanja. Svako slanje podataka između dva 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. 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 su:

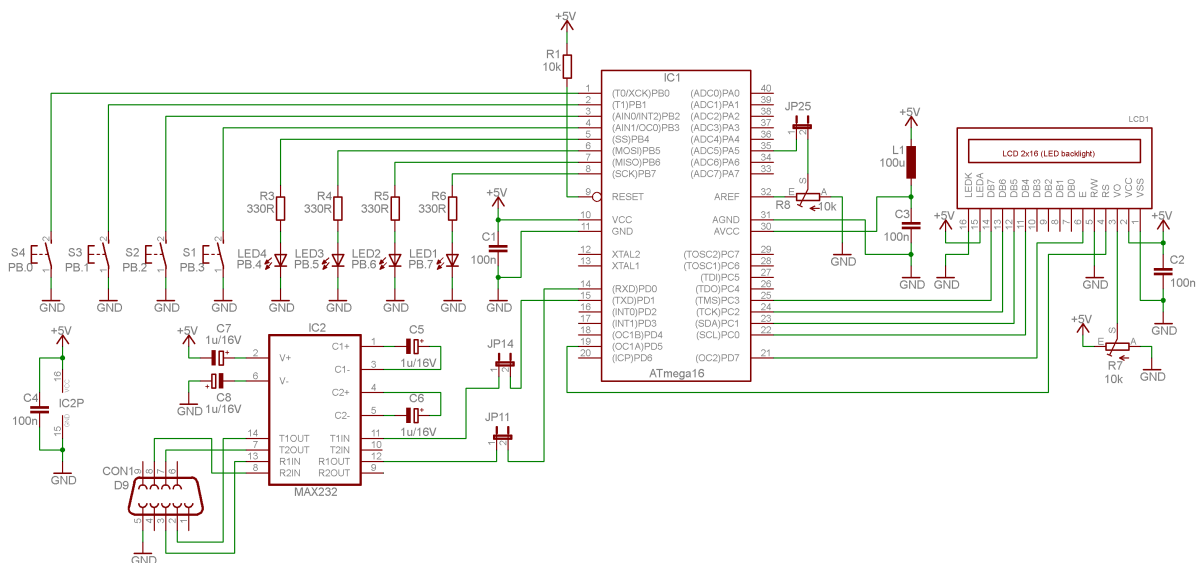
- brzina prijenosa podataka: prema potrebi i mogućnostima uređaja,
- broj podatkovnih bitova: 8,
- paritetni bit: onemogućen,
- *Stop* bit: 1.

Kako bismo poslali 1 B (osam 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 koji razmjenjuju podatke moraju biti jednako konfigurirani. U suprotnom tumačenje podatka neće biti jednoznačno.

Sklopovlje UART definira samo asinkroni serijski komunikacijski protokol, ali ne i fizičke karakteristike uređaja koji komuniciraju. Na primjer, kod mikrokontrolera 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 mikrokontrolera i računala, potrebno je koristiti prilagodni međusklop koji će uskladiti naponske nivoe. Sklop koji se koristi u tu svrhu zove se MAX232.

## 10.1 Vježbe - univerzalna asinkrona serijska komunikacija

Mikrokontroler ATmega16 posjeduje sklopovlje USART (eng. *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 mikrokontroleru ATmega16 sklopovlje USART kompatibilno je sa sklopovljem UART kojeg smo prethodno opisali. Podaci se na mikrokontroler ATmega16 asinkronom serijskom komunikacijom primaju pomoću pina RxD (PD0), a šalju pomoću pina TxD (PD1). Ukoliko na mikrokontroleru ATmega16 koristite asinkronu serijsku komunikaciju, tada digitalne pinove PD0 i PD1 ne možete koristiti u druge svrhe. U svrhu vježbi, mikrokontroler ATmega16 spojiti ćemo s računalom koristeći prilagodni međusklop MAX232. Shema spajanja prilagodnog međusklopa MAX232 na mikrokontroler ATmega16 prikazana je na slici 10.3.



Slika 10.3: Shema spajanja LED dioda, tipkala, LCD displeja, potencijometra i prilagodnog međusklopa MAX232 na mikrokontroler ATmega16

Na shemi sa slike 10.3 nalaze se LCD displej, LED diode, tipkala i potencijometar koje ćemo koristiti u svrhu testiranja rada serijske komunikacije. Kako bi prilagodni međusklop MAX232 bio povezan s mikrokontrolerom ATmega16, na shemi sa slike 10.3 potrebno je postaviti kratkospojnike JP11 i JP14.

Podatak primljen preko RxD pina sprema se u međuspremnik primljenih podataka. Kada se podatak šalje preko TxD pina, on se neposredno prije slanja sprema u međuspremnik podataka za slanje. Sklopovlje USART u mikrokontroleru ATmega16 koristi istu memorijsku lokaciju registra za slanje i za primanje podataka. Ime registra je **UDR**. Kada čitamo podatak iz registra **UDR**, tada kao povratnu vrijednost dobijemo podatak iz međuspremnika primljenih podataka. Kada podatak zapisujemo u registar **UDR**, 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 **UDR** jer postoji opasnost da se novi podatak prepíše preko staroga. Kada se na pinu RxD pojavi *Start* bit, mikrokontroler generira prekid. Na taj način novi podatak možemo odmah pročitati iz registra **UDR** bez obzira na to što se u glavnom programu izvodi dio programskog koda.

Rad sklopovlja USART konfigurira se pomoću registara **UCSRA**, **UCSRB** i **UCSRC**. Konfiguracija se odnosi na broj podatkovnih bitova, paritetni bit, broj *Stop* bitova, omogućavanje prekida kada pristigne podatak i drugo. Podatkovni okvir koji podržava mikrokontroler ATmega16 isti je kao podatkovni okvir prikazan na slici 10.2. Više detalja o konfiguraciji registara **UCSRA**, **UCSRB** i

UCSRC pogledajte u literaturi [1] na stranicama 164 - 167.

Brzina prijenosa podataka konfigurira se pomoću registra **UBRR**, a računa se prema relaciji [1]:

$$BAUD = \frac{F\_CPU}{16(UBRR + 1)}. \quad (10.1)$$

Vrijednost registra **UBRR** za frekvenciju radnog takta  $F\_CPU$  i brzinu prijenosa  $BAUD$  može se izračunati iz relacije (10.1) na sljedeći način:

$$UBRR = \frac{F\_CPU}{16 \cdot BAUD} - 1. \quad (10.2)$$

Rezultat dobiven pomoću relacije (10.2) mora biti cijeli broj jer će se u suprotnom javljati pogreška pri prijenosu podataka. U tablicama 68 - 71 u literaturi [1] nalaze se vrijednosti registra **UBRR** 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 je 8 MHz. Ako odaberemo brzinu prijenosa podataka od 19200 b/s, javit će se pogreška u prijenosu podataka od 0,2 %. 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.

Računalo ćemo povezati s razvojnim okruženjem sa slike 3.1 pomoću kabela za serijsku komunikaciju s konektorom D-SUB9. Nova računala u standardnoj opremi nemaju serijski port, no postoje pretvornici USB protokola na RS232 protokol i obratno. Kada ovaj pretvornik ukopčate u računalo, stvorit će se virtualni serijski port koji će omogućiti serijsku komunikaciju. Bolje rješenje od ovog je korištenje uređaja s pretvornikom FT232 koji USB protokol pretvara u UART protokol i obratno. Kada uređaj FT232 ukopčate u računalo, stvorit će se virtualni serijski port koji će omogućiti serijsku komunikaciju. Uređaj s pretvornikom FT232 direktno se spaja na pinove RxD i TxD mikrokontrolera ATmega16.

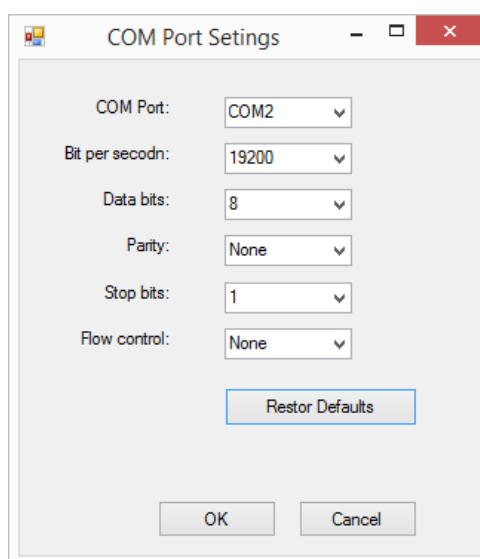
Rad asinkrone serijske komunikacije testirat ćemo pomoću aplikacija za upravljanje i nadzor razvojnog okruženja s mikrokontrolerom ATmega16. S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala) skinite datoteku **Serijska komunikacija-aplikacija.zip** u kojoj se nalazi instalacija aplikacije. Datoteku **Serijska komunikacija-aplikacija.zip** raspakirajte na radnu površinu i instalirajte aplikaciju pokretanjem datoteke **setup.exe**. Aplikaciju pokrenite tako da odaberete **Start** → **All Programs** → **VTSBJ Mikroracunala** → **Serijska komunikacija RS232**. Početni prozor aplikacije prikazan je na slici 10.4.

Aplikacija omogućuje uključenje LED dioda na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću tipkala koja se nalaze u aplikaciji. Pomoću aplikacije možemo prezentirati napon na potencijometru, slati i primiti razne tekstualne poruke. Konfiguracija serijske komunikacije na računalu može se postaviti pomoću prozora prikazanog na slici 10.5. Ovaj prozor će se otvoriti tako da u aplikaciji odaberete **COM** → **Edit**.

<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, ...



Slika 10.4: Aplikacija za upravljanje i nadzor razvojnog okruženja s mikrokontrolerom ATmega16 pomoću serijske komunikacije



Slika 10.5: Konfiguracija serijske komunikacije na računalo

Postavke mikrokontrolera 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. 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 drugim uređajem.

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala) skinite datoteku **USART.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 **USART.zip** raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite **mikroracunala.atsln** u datoteci **\\USART\vjezbe**. U otvorenom projektu nalaze se sve vježbe koje ćemo obraditi u poglavlju **Univerzalna asinkrona serijska komunikacija**. Vježbe ćemo pisati u datoteke s ekstenzijom **\*.c**.

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



### Vježba 10.1.1

Napravite program kojim će se pomoću tipkala PB0, PB1, PB2 i PB3 u aplikaciji na slici 10.4 uključivati redom bijela LED dioda, zelena LED dioda, žuta LED dioda i crvena LED dioda. Klikom miša na tipkalo u aplikaciji preko serijske komunikacije šalje se niz znakova u obliku "PBxy" gdje je:

- "PB" - niz znakova koji označuje port B,
- "x" - znak koji određuje poziciju pina,
- "y" - znak koji određuje stanje pina.

Na primjer, ako je pristigla poruka "PB71", crvenu LED diodu treba uključiti, a ako je pristigla poruka "PB70", crvenu LED diodu treba isključiti. Za niz znakova u obliku "PBxy" kontrolni okvir **Drugi string za LED** ne smije biti označen. Poslane poruke iz aplikacije potrebno je prikazivati na LCD displeju. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku **vjezba1011.c**. Omogućite prevođenje samo datoteke **vjezba1011.c**. Početni sadržaj datoteke **vjezba1011.c** prikazan je programskim kodom 10.1.

Programski kod 10.1: Početni sadržaj datoteke **vjezba1011.c**

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "USART/USART.h"

void inicijalizacija(){

    // PB7,PB6,PB5,i PB4 postavljeni kao izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);

    lcd_init();
    usart_init(19200);
    sei(); // globalno omogućavanje prekida
}

int main(void){
```

```

inicijalizacija();

while (1){

    if(usart_read_all() == 1){

        lcd_clrscr();
        lcd_home();
        lcd_print("%s", usart_buffer);

        if(usart_buffer[0] == 'P' && usart_buffer[1] == 'B'){

            set_port(PORTB, usart_buffer[2] - 48, usart_buffer[3] - 48);

        }

    }

}

return 0;
}

```

Funkcije koje se koriste za asinkronu serijsku komunikaciju definirane su u zaglavlju `usart.h`. Naredba kojom uključujemo zaglavlje `usart.h` u datoteku koja se prevodi je `#include "USART/usart.h"`.

U programskom kodu 10.1 u funkciji `inicijalizacija()` nalazi se funkcija `usart_init(uint32_t baud)` za konfiguriranje asinkrone serijske komunikacije mikrokontrolera. Ova funkcija kao argument prima brzinu prijenosa podataka. Osim brzine prijenosa podataka, funkcija `usart_init` konfigurira sljedeće parametre:

- broj podatkovnih bitova: 8,
- paritetni bit: onemogućen,
- *Stop* bit: 1,
- prekid za pristigle poruke: omogućen,
- pinovi RxD i TxD: omogućeni.

Brzina prijenosa podataka u programskom kodu 10.1 postavljena je na 19200 b/s. Aplikacija će serijskom komunikacijom slati tekstualne poruke oblika "PBxy". Ova poruka šalje se na način da se svaki znak šalje zasebno i dodatno na kraju poruke postavlja se zaključni znak koji označava kraj pristigle poruke. Registar `UDR` širine je osam bitova i u njega stane samo jedan znak (podatak od osam bitova). Postavlja se pitanje kako primiti cijelu tekstualnu poruku koja može biti proizvoljno dugačka? U zaglavlju `usart.h` deklariran je niz znakova `usart_buffer` koji predstavlja međuspremnik znakova (eng. *Buffer*) koji se puni pomoću registra `UDR`. Parametri za međuspremnik znakova `usart_buffer` prikazani su u programskom kodu 10.2.

Programski kod 10.2: Parametri za međuspremnik znakova `usart_buffer`

```

#define end_char 0x0D // zaključni znak
#define MESSAGE_LENGTH 50 // maksimalna duljina poruke
char usart_buffer[MESSAGE_LENGTH];

```

Parametri u programskom kodu 10.2 redom su:

- `end_char` - definirana konstanta koja predstavlja zaključni znak. Aplikacija sa slike 10.4 kao zaključni znak šalje *Carriage Return* znak s ASCII kodom 0x0D. Zaključni znak najčešće je znak koji se ne koristi u kreiranju tekstualne poruke (npr. '\*', ';', ':').

- `MESSAGE_LENGTH` - definirana konstanta koja predstavlja maksimalnu duljinu tekstualne poruke koju ćete poslati na mikrokontroler i spremiti u međuspremnik znakova `usart_buffer`. Vrijednost se mijenja prema potrebi.
- `usart_buffer` - međuspremnik znakova koji se puni pomoću niza znakova koji pristižu u registar `UDR`. Kada registar `UDR` primi zaključni znak (u ovom slučaju znak s ASCII kodom `0x0D`), međuspremnik znakova zaključuje se s tzv. *null* znakom<sup>2</sup>.

Prekidna rutina koja se poziva kada novi znak preko pina `RxD` dolazi na mikrokontroler zove se `USART_RXC_vect`. Međuspremnik znakova uvijek čuva zadnju pristiglu poruku. Kada u registar `UDR` dođe znak koji predstavlja početak nove poruke, on se u prekidnoj rutini `USART_RXC_vect` sprema na memorijsku lokaciju `usart_buffer[0]`. Sljedeći se znak sprema na memorijsku lokaciju `usart_buffer[1]` i tako redom dok ne dođe zaključni znak. Prekidna rutina `USART_RXC_vect` napisana je u datoteci `usart.c` i preporučuje se samo njeno korištenje bez izmjene tijela prekidne rutine. Korisnik mijenja samo parametre prikazane programskim kodom 10.2.

U programskom kodu 10.1 pinovi `PB4`, `PB5`, `PB6` i `PB7` konfigurirani su kao izlazni pinovi, konfiguriran je LCD displej i globalno su omogućeni prekidni makronaredbom `sei()`.

U `while` petlji koristi se funkcija `usart_read_all()`. Ova funkcija vraća vrijednost 1 ako je dostupna nova poruka u međuspremniku znakova `usart_buffer`, a inače vraća vrijednost 0. Funkcija `usart_read_all()` detektira novu poruku ako je pristigao zaključni znak definiran konstantom `end_char`.

Prevedite datoteku `vjezba1011.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port COM1 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`.

Kada pritisnete tipkalo `PB0` u aplikaciji sa slike 10.4, promijenit će se stanje bijele LED diode na način da će aplikacija serijskom komunikacijom na mikrokontroler ATmega16 poslati poruku "PB40" ili "PB41". Ove dvije poruke prikazali smo pomoću LCD displeja tako da smo pomoću funkcije `lcd_print` ispisali sadržaj međuspremnika znakova `usart_buffer`.

Naredbom `if(usart_buffer[0] == 'P' && usart_buffer[1] == 'B')` u programskom kodu 10.1 ispitujemo jesu li prva dva znaka poruke pristigla serijskom komunikacijom "PB". Ako nije tako, stanje LED dioda ne smije se mijenjati. Ako su prva dva znaka poruke pristigla serijskom komunikacijom "PB", tada sljedeća dva znaka u poruci predstavljaju pin na portu B kojem treba promijeniti stanje i stanje koje treba postaviti na tom pinu<sup>3</sup>. Makronaredba `set_port(port, pin, stanje)` kao drugi argument prima poziciju pina, a kao treći argument stanje pina koje želimo postići. U poruci "PBxy", 'x' i 'y' su znakovi s ASCII kodom brojeva 0 - 9. ASCII kod znaka '0' je 48 te je od znaka 'x' potrebno oduzeti broj 48 kako bismo dobili poziciju pina kojem želimo promijeniti stanje. Isto vrijedi i za znak 'y'. Ovo je standardna pretvorba znakova dekadskog sustava u brojeve dekadskog sustava. Prema navedenom, naredbom `set_port(PORTB, usart_buffer[2] - 48, usart_buffer[3] - 48)` mijenjamo stanje na pinu PBx.

Poruku oblika "PBxy" možete poslati pomoću tekstualnog okvira za slanje poruka. Na primjer, upišite u tekstualni okvir za slanje poruka "PB61" te pritisnite tipku `Pošalj`. Pomoću tekstualnog okvira pošaljite proizvoljnu poruku. Ako poruka nije oblika "PBxy", ona će se ispisati na LCD displeju, dok se stanja LED dioda neće mijenjati.

Dio programskog koda kojim ćete uvijek provjeravati nalazi li se u međuspremniku znakova `usart_buffer` nova poruka prikazan je programskim kodom 10.3. Ovaj dio koda uvijek mora

<sup>2</sup>U programskom jeziku C niz znakova mora se zaključiti se s *null* znakom kako bi prevoditelj znao gdje je kraj teksta.

<sup>3</sup>Ova tvrdnja vrijedi pod uvjetom da aplikacija šalje ispravnu poruku, a pretpostavljamo da šalje.



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(usart_read_all() == 1){  
  // izvođenje programskog koda na temelju pristigle poruke  
}
```

Pomoću asinkrone komunikacije, poruke mogu razmjenjivati svi uređaji koji podržavaju sklopovlje UART. Na primjer, mikrokontroler može komunicirati s računalom, s GSM modemom, s drugim mikrokontrolerom, s GPS modulom i drugim uređajima koji podržavaju sklopovlje UART.

Promijenite programski kod 10.1 tako da se stanje LED dioda mijenja porukom tipa "PORTBxy". Na primjer, ako je poslana poruka "PORTB41", bijelu LED diodu potrebno je uključiti. Prevedite datoteku `vjezba1011.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću aplikacije sa slike 10.4. Poruku tipa "PORTBxy" šaljite pomoću tekstualnog okvira za slanje poruka u aplikaciji sa slike 10.4.

Zatvorite datoteku `vjezba1011.c` i onemogućite prevođenje ove datoteke.



## Vježba 10.1.2

Napravite program kojim će se na LCD displeju prikazivati vrijednost klizača u aplikaciji sa slike 10.4. Dodatno napravite dio programskog koda koji će raditi sljedeće:

- ako je vrijednost klizača jednaka 0, sve LED diode potrebno je ugasiti,
- ako je vrijednost klizača veća od 0 i manja od 2500, uključite crvenu LED diodu,
- ako je vrijednost klizača veća i jednaka 2500 i manja od 5000, uključite crvenu i žutu LED diodu,
- ako je vrijednost klizača veća i jednaka 5000 i manja od 7500, uključite crvenu, žutu i zelenu LED diodu,
- ako je vrijednost klizača veća i jednaka 7500, uključite sve LED diode.

Brzinu prijenosa podataka postavite na 38400 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku `vjezba1012.c`. Omogućite prevođenje samo datoteke `vjezba1012.c`. Početni sadržaj datoteke `vjezba1012.c` prikazan je programskim kodom 10.4.

Programski kod 10.4: Početni sadržaj datoteke `vjezba1012.c`

```
#include "AVR lib/AVR_lib.h"  
#include <avr/io.h>  
#include "LCD/lcd.h"  
#include "USART/USART.h"  
  
void inicijalizacija(){  
  // PB7,PB6,PB5,i PB4 postavljeni kao izlazni pinovi
```

```

    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);

    lcd_init();
    // konfiguracija asinkrone serijske komunikacije
    sei(); // globalno omogućavanje prekida
}

int main(void){

    inicijalizacija();

    uint8_t i;
    uint16_t slider = 0;

    while (1){

        if(usart_read_all() == 1){

            lcd_clrscr();
            lcd_home();
            lcd_print("%s", usart_buffer);

            slider = 0;
            for(i = 0; i < 4; i++){

                if(usart_buffer[i] == '\0') break;

                slider = slider * 10 + (usart_buffer[i] - 48);

            }

            // napravite kod za LED diode

        }

    }

    return 0;
}

```

U programskom kodu 10.4 u funkciji `inicijalizacija()` napravite konfiguraciju asinkrone serijske komunikacije s brzinom prijenosa od 38400 b/s. Vrijednost klizača iz aplikacije šalje se u obliku niza znakova. Taj niz znakova potrebno je pretvoriti u cijeli broj koji se može kretati od [0, 9999]. Vrijednost međuspremnika znakova `usart_buffer` može se kretati od "0" do "9999". Kada se asinkronom serijskom komunikacijom primi nova poruka, ona se u `for` petlji pretvara u cijeli broj. Vrijednost klizača čuva se u varijabli `slider` čija je početna vrijednost 0. Proučite dio programskog koda za pretvorbu poruke u cijeli broj.

Napravite dio programskog koda koji će uključivati LED diode s obzirom na vrijednost varijable `slider` sukladno uputama.

Prevedite datoteku `vjezba1012.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću klizača u aplikaciji sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port COM1 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku **Start COM**.

Zatvorite datoteku `vjezba1012.c` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.3

Napravite program kojim ćete osigurati sljedeće:

- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "C", mikrokontroler aplikaciji mora poslati proizvoljan znak,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "S", mikrokontroler aplikaciji mora poslati proizvoljnu poruku,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "I", mikrokontroler aplikaciji mora vratiti proizvoljan cijeli broj,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "R", mikrokontroler aplikaciji mora vratiti proizvoljan realni broj,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "A", mikrokontroler aplikaciji mora vratiti rezultat analogno-digitalne pretvorbe na pinu PA5,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "N", mikrokontroler aplikaciji mora vratiti napon na potenciometru spojenom na pin PA5.

Pristigle poruke mikrokontrolera omogućit ćemo u aplikaciji tako da označimo kontrolni okvir **Omogući primljene poruke**. Znakove na mikrokontroler šaljite pomoću tekstualnog okvira za slanje poruka, a prikazujte ih na LCD displeju. Brzinu prijenosa podataka postavite na 19200 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku `vjezba1013.c`. Omogućite prevođenje samo datoteke `vjezba1013.c`. Početni sadržaj datoteke `vjezba1013.c` prikazan je programskim kodom 10.5.

Programski kod 10.5: Početni sadržaj datoteke `vjezba1013.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
// uključite potrebna zaglavlja

void inicijalizacija(){
    // inicijalizacija za LCD, ADC i USART
    sei(); // globalno omogućavanje prekida
}

int main(void){

    inicijalizacija();

    uint16_t ADC5; // rezultat AD pretvorbe
    float VPA5; // napon na pinu PA5
    const float VREF = 5.0; // AVCC

    while (1)
    {
        if(usart_read_all() == true){

            lcd_clrscr();
            lcd_home();
```

```

        lcd_print("%s", usart_buffer);

        switch (usart_buffer[0]){
            case 'C':
                // poslati znak V
                break;
            case 'S':
                //poslati poruku Mikrokontroler ATmega16
                break;
            case 'I':
                //poslati broj 1023
                break;
            case 'R':
                //poslati broj 3.14
                break;
            case 'A':
                ADC5 = adc_read_10bit(5);
                //poslati varijablu ADC5
                break;
            case 'N':
                ADC5 = adc_read_10bit(5);
                usart_write("Napon: %0.4f V", ADC5*5.0/1024);
                break;
            default:
                usart_write_string("Neispravan znak!");
                break;
        }
    }
}

return 0;
}

```

Često je u praksi potrebno prezentirati neku varijablu sustava na zahtjev aplikacije. U ovoj vježbi zahtjevi su znakovi koje šaljemo na mikrokontroler koji povratno šalje informacije iz okoline sustava u kojoj se mikrokontroler nalazi.

U programskom kodu 10.5 potrebno je uključiti sva zaglavlja koja se koriste u vježbi te konfigurirati analogno-digitalnu pretvorbu, LCD displej i asinkronu serijsku komunikaciju s brzinom prijenosa od 19200 b/s.

Funkcije koje služe za slanje poruka iz mikrokontrolera prema drugom uređaju su:

- `usart_write_char(char a)` - funkcija koja na pin TxD šalje znak `a` (npr. `usart_write_char('Z')`),
- `usart_write_string(char *s)` - funkcija koja na pin TxD šalje tekstualnu poruku `s` (npr. `usart_write_string("Temperatura")`),
- `usart_write_int(int d)` - funkcija koja na pin TxD šalje cijeli broj `d` u obliku niza znakova (npr. `usart_write_int(123)`),
- `usart_write_float(float f)` - funkcija koja na pin TxD šalje realan broj `f` na četiri decimalna mjesta u obliku niza znakova (npr. `usart_write_float(1.234)`).
- `usart_write(const char * format, ...)` - funkcija koja na pin TxD šalje tekstualnu poruku. Sintaksa funkcije `usart_write` identična je sintaksi funkcije `printf` koja je standardna funkcija programskog jezika C (npr. `usart_write("Struja: %0.3f A", Ia)`).

U programskom kodu 10.5 u `switch case` bloku provjeravamo prvi znak međuspremnik znakova `usart_buffer[0]`. U ovisnosti o tome koji smo znak poslali na mikrokontroler, on vraća povratnu vrijednost. Zamijenite komentare u `switch case` bloku s odgovarajućim pozivima funkcija. Ukoliko pošaljete krivi znak pomoću aplikacije na mikrokontroler, on će vratiti poruku "Neispravan znak!". Proučite slučaj u kojem mikrokontroler primi znak 'N'.

Prevedite datoteku `vjezba1013.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port COM1 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`.

U aplikaciji označite kontrolni okvir `Omogući primljene poruke` kako bi se omogućile primljene poruke. Pomoću tekstualnog okvira za slanje poruka i tipke `Pošalji` šalјite znakove na mikrokontroler i pratite povratne vrijednosti.

Zatvorite datoteku `vjezba1013.c` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.4

Napravite program kojim ćete osigurati zahtjeve vježbe 10.1.1. Dodatno je potrebno napraviti prekidnu rutinu `ISR(TIMER1_OVF_vect)` koja će svakih 400 ms putem serijske komunikacije u aplikaciju sa slike 10.4 slati vrijednost analogno-digitalne pretvorbe na pinu PA5. Aplikacija će vrijednost analogno-digitalne pretvorbe prezentirati u obliku napona na voltmetru i u obliku otpora potenciometra na digitalnom ommetru. Kontrolni okvir `Omogući primljene poruke` u ovom slučaju ne smije biti označen. Brzinu prijenosa podataka postavite na 19200 b/s. Znakovni niz koji šalјemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku `vjezba1014.c`. Omogućite prevođenje samo datoteke `vjezba1014.c`. Početni sadržaj datoteke `vjezba1014.c` prikazan je programskim kodom 10.6.

Programski kod 10.6: Početni sadržaj datoteke `vjezba1014.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "USART/USART.h"
#include "ADC/adc.h"
#include "timer/timer.h"

uint16_t ADC5;

ISR(TIMER1_OVF_vect){
    TCNT1 = 0;

    // slanje vrijednosti analogno-digitalne pretvorbe
}

void inicijalizacija(){

    // PB7,PB6,PB5,i PB4 postavljeni kao izlazni pinovi
    DDRB |= (1 << PB7) | (1 << PB6) | (1 << PB5) | (1 << PB4);

    lcd_init();
    adc_init();
    usart_init(19200);
    // F_CPU/64, normalan način rada, omogućen prekid tajmera 1
    timer1_init();
}
```

```

    TCNT1 = 0;
    sei(); // globalno omogućavanje prekida
}

int main(void){

    inicijalizacija();

    while (1){

        if(usart_read_all() == 1){

            lcd_clrscr();
            lcd_home();
            lcd_print("%s", usart_buffer);

            if(usart_buffer[0] == 'P' && usart_buffer[1] == 'B'){

                set_port(PORTB, usart_buffer[2] - 48, usart_buffer[3] - 48);

            }

        }

    }

    return 0;
}

```

Dio programskog koda 10.6 isti je kao programski kod 10.1 iz vježbe 10.1.1. U ovoj vježbi koristi se sklop *Timer/Counter1* u normalnom načinu rada koji je konfiguriran pozivom funkcije `timer1_init()`. Provjerite u zaglavlju `timer.h` je li djelitelj frekvencije radnog takta ispravno definiran.

Vrijeme između dva poziva prekidne rutine mora biti 400 ms ( $t_{T1} = 0,4$  s). Početnu vrijednost registra `TCNT1` izračunat ćemo pomoću korigirane relacije (8.2):

$$TCNT1_0 = 65536 - t_{T1} \cdot \frac{F_{CPU}}{PRESCALER} = 65536 - 0,4 \cdot \frac{8000000}{64} = 15536. \quad (10.3)$$

Upišite početnu vrijednost registra `TCNT1` u programski kod 10.6 na odgovarajuća mjesta. U prekidnoj rutini vrijednost analognodigitalne pretvorbe na pinu PA5 spremite u varijablu `ADC5`. Nakon toga je varijablu `ADC5` potrebno poslati serijskom komunikacijom u aplikaciju sa slike 10.4 pomoću funkcije `usart_write_int`.

Prevedite datoteku `vjezba1014.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port COM1 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`.

U aplikaciji sa slike 10.4 kontrolni okvir `Omogući primljene poruke` ne smije biti označen. Mijenjajte vrijednost potencijometra i promatrajte voltmetar i digitalni ommetar u aplikaciji.

Zatvorite datoteku `vjezba1014.c` i onemogućite prevođenje ove datoteke.



### Vježba 10.1.5

Napravite program kojim će aplikacija sa slike 10.4 mijenjati stanja svih LED dioda pomoću jedne poruke. Poruka koju je potrebno slati pomoću tekstualnog okvira za slanje poruka ima oblik "`PBx0y0x1y1x2y2x3y3`" gdje je:

- "PB" - niz znakova koji označuje port B,

- " $x_i$ " - znak koji određuje poziciju pina  $i$ , ( $i = 0, 1, 2, 3$ ),
- " $y_i$ " - znak koji određuje stanje  $i$ -tog pina, ( $i = 0, 1, 2, 3$ ).

Na primjer, ako je pristigla poruka "PB71406150", crvenu je LED diodu potrebno uključiti, bijelu je LED diodu potrebno isključiti, žutu je LED diodu potrebno uključiti i zelenu je LED diodu potrebno isključiti. Poslane poruke iz aplikacije potrebno je prikazivati na LCD displeju. Brzinu prijenosa podataka postavite na 19200 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

U projektnom stablu otvorite datoteku vjezba1015.c. Omogućite prevođenje samo datoteke vjezba1015.c. Početni sadržaj datoteke vjezba1015.c prikazan je programskim kodom 10.7.

Programski kod 10.7: Početni sadržaj datoteke vjezba1015.c

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "USART/USART.h"

void inicijalizacija(){
    // kopirati iz datoteke vjezba1011.c
}

int main(void){
    inicijalizacija();

    while (1){
        if(usart_read_all() == 1){
            lcd_clrscr();
            lcd_home();
            lcd_print("%s", usart_buffer);

            if(usart_buffer[0] == 'P' && usart_buffer[1] == 'B'){
                set_port(PORTB, usart_buffer[2] - 48, usart_buffer[3] - 48);
                set_port(PORTB, usart_buffer[4] - 48, usart_buffer[5] - 48);
                set_port(PORTB, usart_buffer[6] - 48, usart_buffer[7] - 48);
                set_port(PORTB, usart_buffer[8] - 48, usart_buffer[9] - 48);
            }
        }
    }

    return 0;
}
```

U programski kod 10.7 u funkciju `inicijalizacija()` kopirajte tijelo funkcije `inicijalizacija()` iz vježbe 10.1.1. Naredbom `if(usart_buffer[0] == 'P' && usart_buffer[1] == 'B')` u programskom kodu 10.7 ispitujemo jesu li prva dva znaka poruke pristigla serijskom komunikacijom "PB". Ako nije tako, stanje LED dioda ne smije se mijenjati. Ako su prva dva znaka poruke pristigla serijskom komunikacijom "PB", tada sljedećih osam znakova u poruci predstavljaju pinove na portu B čija stanja treba promijeniti i stanja koja treba postaviti

na tim pinovima. Treći i četvrti znak u poruci predstavljaju prvi pin i stanje tog pina koje je potrebno postaviti. Peti i šesti znak u poruci predstavljaju drugi pin i stanje tog pina koje je potrebno postaviti itd. Stanja pinova postavljena su makronaredbom `set_port`.

Prevedite datoteku `vjezba1015.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16 pomoću aplikacije sa slike 10.4. Namjestite parametre serijske komunikacije za COM Port COM1 pomoću prozora sa slike 10.5 te otvorite serijski port klikom miša na tipku `Start COM`.

U tekstualni okvir za slanje poruka upišite poruku oblika "`PBx0y0x1y1x2y2x3y3`", a zatim stisnite tipku `Pošalji`. Na primjer, ako ste upisali i poslali poruku `PB41516171`, sve LED diode će se upaliti.

Zatvorite datoteku `vjezba1015.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.



## 10.2 Zadaci - univerzalna asinkrona serijska komunikacija

### Zadatak 10.2.1

Napravite program kojim će se pomoću tipkala PB0, PB1, PB2 i PB3 u aplikaciji na slici 10.4 uključivati redom bijela LED dioda, zelena LED dioda, žuta LED dioda i crvena LED dioda. Klikom miša na tipkalo u aplikaciji preko serijske komunikacije šalje se niz znakova u obliku "Bxy" gdje je:

- "B" - niz znakova koji označuje port B,
- "x" - znak koji određuje poziciju pina,
- "y" - znak koji određuje stanje pina.

Na primjer, ako je pristigla poruka "B71", crvenu LED diodu treba uključiti, a ako je pristigla poruka "B70", crvenu LED diodu treba isključiti. Za niz znakova u obliku "Bxy" kontrolni okvir Drugi string za LED mora biti označen. Poslane poruke iz aplikacije potrebno je prikazivati na LCD displeju. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

---

### Zadatak 10.2.2

Napravite program kojim će se na LCD displeju prikazivati vrijednost klizača u aplikaciji sa slike 10.4. Dodatno napravite dio programskog koda koji će raditi sljedeće:

- ako je vrijednost klizača jednaka 0, sve LED diode potrebno je ugasiti,
- ako je vrijednost klizača veća od 0 i manja od 1000, uključite crvenu LED diodu,
- ako je vrijednost klizača veća i jednaka 1000 i manja od 4000, uključite crvenu i žutu LED diodu,
- ako je vrijednost klizača veća i jednaka 4000 i manja od 8000, uključite crvenu, žutu i zelenu LED diodu,
- ako je vrijednost klizača veća i jednaka 8000, uključite sve LED diode.

Brzinu prijenosa podataka postavite na 9600 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristiže serijskom komunikacijom.

---

### Zadatak 10.2.3

Napravite program kojim ćete osigurati sljedeće:

- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "Z", mikrokontroler aplikaciji mora poslati proizvoljan znak,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "P", mikrokontroler aplikaciji mora poslati proizvoljnu poruku,

- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "C", mikrokontroler aplikaciji mora vratiti proizvoljan cijeli broj,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "D", mikrokontroler aplikaciji mora vratiti proizvoljan realni broj,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "R", mikrokontroler aplikaciji mora vratiti rezultat analogno-digitalne pretvorbe na pinu PA5,
- ako pomoću aplikacije sa slike 10.4 na mikrokontroler pošaljemo znak "V", mikrokontroler aplikaciji mora vratiti napon na potencijometru spojenom na pin PA5.

Pristigle poruke mikrokontrolera omogućit ćemo u aplikaciji tako da označimo kontrolni okvir **Omogući primljene poruke**. Znakove na mikrokontroler šaljite pomoću tekstualnog okvira za slanje poruka, a prikazujte ih na LCD displeju. Brzinu prijenosa podataka postavite na 38400 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristizuje serijskom komunikacijom.

---

#### Zadatak 10.2.4

Napravite program kojim ćete osigurati zahtjeve vježbe 10.1.1. Dodatno je potrebno napraviti prekidnu rutinu `ISR(TIMERO_OVF_vect)` koja će svakih 250 ms putem serijske komunikacije u aplikaciju sa slike 10.4 slati vrijednost analogno-digitalne pretvorbe na pinu PA5. Aplikacija će vrijednost analogno-digitalne pretvorbe prezentirati u obliku napona na voltmetru i u obliku otpora potencijometra na digitalnom ommetru. Kontrolni okvir **Omogući primljene poruke** u ovom slučaju ne smije biti označen. Brzinu prijenosa podataka postavite na 38400 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristizuje serijskom komunikacijom.

---

#### Zadatak 10.2.5

Napravite program kojim će aplikacija sa slike 10.4 mijenjati stanja svih LED dioda pomoću jedne poruke. Poruka koju je potrebno slati pomoću tekstualnog okvira za slanje poruka ima oblik " $Bx_0y_0x_1y_1x_2y_2x_3y_3$ " gdje je:

- "B" - niz znakova koji označuje port B,
- " $x_i$ " - znak koji određuje poziciju pina  $i$ , ( $i = 0, 1, 2, 3$ ),
- " $y_i$ " - znak koji određuje stanje  $i$ -tog pina, ( $i = 0, 1, 2, 3$ ).

Na primjer, ako je pristigla poruka "B71406150", crvenu je LED diodu potrebno uključiti, bijelu je LED diodu potrebno isključiti, žutu je LED diodu potrebno uključiti i zelenu je LED diodu potrebno isključiti. Poslane poruke iz aplikacije potrebno je prikazivati na LCD displeju. Brzinu prijenosa podataka postavite na 9600 b/s. Znakovni niz koji šaljemo na serijski port zaključen je s *Carriage Return* znakom (ASCII kod 0x0D) kako bi se u mikrokontroleru detektirao kraj poruke koja pristizuje serijskom komunikacijom.

---

# Poglavlje 11

## Vanjski prekidi

Vanjski prekidi (eng. *External Interrupts*) su prekidi koji su izazvani vanjskim događajima. Vanjske događaje najčešće generiraju senzori na svome izlazu u obliku rastućih (eng. *rising*) i padajućih (eng. *falling*) bridova signala. Primjer takvih senzora su enkoderi, kapacitivni senzori, induktivni senzori, krajnji prekidači, tipkala i drugi. Izlazi senzora spajaju se na pinove mikrokontrolera koji omogućuju detekciju rastućih i padajućih bridova signala. Ovi bridovi signala izazivaju prekide u mikrokontroleru 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

Mikrokontroler ATmega16 ima tri pina (INT0 (PD2), INT1 (PD3) i INT2 (PB2)) za generiranje vanjskih prekida. Na pinovima INT0 (PD2) i INT1 (PD3) zahtjev za prekid može generirati padajući i/ili rastući brid signala te niska razina signala. Na pinu INT2 (PB2) zahtjev za prekid može generirati samo padajući ili rastući brid signala.

Vanjski prekid INT0 konfigurira se u registru **MCUCR** pomoću bitova **ISCO0** i **ISCO1** prema tablici 11.1. Prekid na pinu INT1 konfigurira se u registru **MCUCR** pomoću bitova **ISC10** i **ISC11** prema tablici 11.2. Primjer konfiguracije vanjskih prekida INT0 i INT1 u registru **MCUCR** prikazan je u programskom kodu 11.1. Prema konfiguraciji u programskom kodu 11.1 vanjski prekid INT0 generirat će zahtjev za prekid na padajući i rastući brid, a vanjski prekid INT1 generirat će zahtjev za prekid samo na padajući brid.

Tablica 11.1: Konfiguracija za vanjski prekid na pinu INT0 (PD2)

<b>ISCO1</b>	<b>ISCO0</b>	Način rada za prekid INT0
0	0	Niska razina signala na pinu INT0 (PD2) generira zahtjev za prekid
0	1	Bilo koja logička promjena signala na pinu INT0 (PD2) generira zahtjev za prekid
1	0	Padajući brid signala na pinu INT0 (PD2) generira zahtjev za prekid
1	1	Rastući brid signala na pinu INT0 (PD2) generira zahtjev za prekid

Tablica 11.2: Konfiguracija za vanjski prekid na pinu INT1 (PD3)

ISC11	ISC10	Način rada za prekid INT1
0	0	Niska razina signala na pinu INT1 (PD3) generira zahtjev za prekid
0	1	Bilo koja logička promjena signala na pinu INT1 (PD3) generira zahtjev za prekid
1	0	Padajući brid signala na pinu INT1 (PD3) generira zahtjev za prekid
1	1	Rastući brid signala na pinu INT1 (PD3) generira zahtjev za prekid

Programski kod 11.1: Konfiguracija vanjskih prekida INT0 i INT1 u registru `MCUCR`

```
// oba brida generiraju prekid INT0
set_bit_reg(MCUCR, ISC00); // ISC00 = 1
reset_bit_reg(MCUCR, ISC01); // ISC01 = 0
// padajući brid generira prekid INT1
reset_bit_reg(MCUCR, ISC10); // ISC10 = 0
set_bit_reg(MCUCR, ISC11); // ISC11 = 1
```

Vanjski prekid INT2 konfigurira se u registru `MCUCSR` pomoću bita `ISC2` na sljedeći način:

- ako je bit `ISC2` jednak 0, tada padajući brid signala na pinu INT2 (PB2) generira zahtjev za prekid,
- ako je bit `ISC2` jednak 1, tada rastući brid signala na pinu INT2 (PB2) generira zahtjev za prekid.

Na primjer, naredbom `set_bit_reg(MCUCSR, ISC2)`; vanjski prekid INT2 konfigurira se tako da rastući brid signala na pinu INT2 (PB2) generira zahtjev za prekid.

Vanjski prekidi omogućuju se u registru `GICR` na sljedeći način:

- vanjski prekid na pinu INT0 (PD2) bit će omogućen ako u registar `GICR` na mjesto bita `INT0` upišete vrijednost 1,
- vanjski prekid na pinu INT1 (PD3) bit će omogućen ako u registar `GICR` na mjesto bita `INT1` upišete vrijednost 1,
- vanjski prekid na pinu INT2 (PB2) bit će omogućen ako u registar `GICR` na mjesto bita `INT2` upišete vrijednost 1.

Primjer omogućavanja vanjskih prekida INT0, INT1 i INT2 u registru `GICR` prikazan je u programskom kodu 11.2.

Programski kod 11.2: Omogućavanje prekida INT0, INT1 i INT2 u registru `GICR`

```
set_bit_reg(GICR, INT0); // omogućen vanjski prekid INT0
set_bit_reg(GICR, INT1); // omogućen vanjski prekid INT1
set_bit_reg(GICR, INT2); // omogućen vanjski 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 (PD2),

- `ISR(INT1_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT1 (PD3),
- `ISR(INT2_vect)` - prekidna rutina koja se poziva kada se generira prekid pomoću pina INT2 (PB2).

Kada se koriste vanjski prekidi na pinovima INT0 (PD2), INT1 (PD3) i INT2 (PB2), tada pinove PD2, PD3 i PB2 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 PD2, PD3 i PB2. U slučaju senzora s *push-pull* izlazom pritezne otpornike nije potrebno uključiti.

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.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 `mikroracunala.atsln` u datoteci `\\Vanjski prekidi\vjezbe`. U otvorenom projektu nalaze se sve vježbe koje ćemo obraditi u poglavlju `Vanjski prekidi`. Vježbe ćemo pisati u datoteke s ekstenzijom `*.c`.

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



### Vježba 11.1.1

Napravite program kojim ćete:

- na pinu INT0 (PD2) brojati padajuće i rastuće bridove signala s tipkala spojenog na pin PB0,
- na pinu INT1 (PD3) brojati rastuće bridove signala s tipkala spojenog na pin PB1,
- na pinu INT2 (PB2) brojati padajuće bridove signala s tipkala spojenog na pin PB2.

Vrijednosti brojača bridova za pinove INT0, INT1 i INT2 prikazite na LCD displeju.

U projektnom stablu otvorite datoteku `vjezba1111.c`. Omogućite prevođenje samo datoteke `vjezba1111.c`. Početni sadržaj datoteke `vjezba1111.c` prikazan je programskim kodom 11.3.

Programski kod 11.3: Početni sadržaj datoteke `vjezba1111.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include <avr/interrupt.h>

uint8_t brojac_int0;
uint8_t brojac_int1;
uint8_t brojac_int2;

ISR(INT0_vect) // prekidna rutina za INT0
{
    brojac_int0++;
}
```

<sup>1</sup>Ovo je preporuka, iako će se vanjski prekidi generirati i ako su pinovi PD2, PD3 i PB2 postavljeni kao izlazni pinovi.

```

}

ISR(INT1_vect) // prekidna rutina za INT1
{
    brojac_int1++;
}

ISR(INT2_vect) // prekidna rutina za INT2
{
    brojac_int2++;
}

void inicijalizacija(){

    // oba brida generiraju prekid INTO
    set_bit_reg(MCUCR,ISC00); // ISC00 = 1
    reset_bit_reg(MCUCR,ISC01); // ISC01 = 0

    //konfigurirajte INT1 i INT2

    set_bit_reg(GICR,INT0); // omogućen vanjski prekid INTO

    // omogućite vanjski prekid za INT1 i INT2

    sei(); // globalno omogućavanje prekida

    input_port(DDRD,PD2); // pin PD2 postavljen kao ulazni
    set_port(PORTD,PD2,1); // uključen pritezni otpornik na PD2
    input_port(DDRD,PD3); // pin PD3 postavljen kao ulazni
    set_port(PORTD,PD3,1); // uključen pritezni otpornik na PD3
    input_port(DDRB,PB2); // pin PB2 postavljen kao ulazni
    set_port(PORTB,PB2,1); // uključen pritezni otpornik na PB2

    lcd_init();
}

int main(void){

    inicijalizacija();

    while(1)
    {
        lcd_clrscr();
        lcd_home();
        lcd_print("INT0=%d,INT1=%d\n", brojac_int0, brojac_int1);
        lcd_print("INT2=%d", brojac_int2);
        _delay_ms(200);
    }

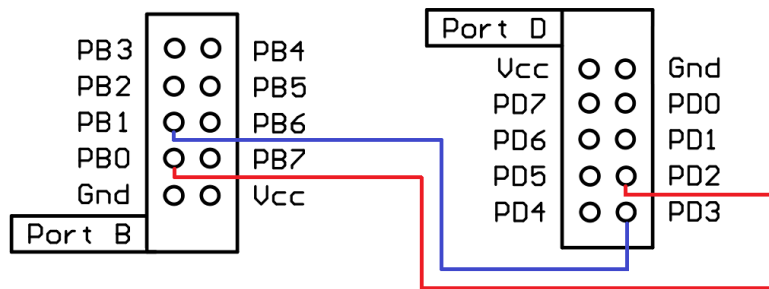
    return 0;
}

```

Za generiranje vanjskih prekida INT0, INT1 i INT2 koristit ćemo tipkala PB0, PB1 i PB2. Na razvojnom okruženju s mikrokontrolerom ATmega16 na pinu INT2 (PB2) spojeno je tipkalo. Na pinove INT0 (PD2) i INT1 (PD3) spojiti ćemo tipkala koja su spojena na pinove PB0 i PB1. Shema spajanja pinova INT0 (PD2) i INT1 (PD3) na tipkala koja su spojena na pinove PB0 i PB1 prikazana je na slici 11.1. Ove pinove spojiti ćete na sljedeći način:

- izvučeni pin PB0 na priključnici Port B spojite pomoću žice na izvučeni pin PD2 na priključnici Port D (slika 11.1),
- izvučeni pin PB1 na priključnici Port B spojite pomoću žice na izvučeni pin PD3 na

priključnici Port D (slika 11.1).



Slika 11.1: Spajanje pinova INT0 (PD2) i INT1 (PD3) na tipkala koja su spojena na pinove PB0 i PB1 pomoću priključnica Port B i Port D

U programskom kodu 11.3 u funkciji `inicijalizacija()` vanjski prekid INT0 konfiguriran je tako da se zahtjev za prekid generira na rastući i padajući brid signala. Za ovu konfiguraciju, prema tablici 11.1, bit `ISCO0` u registru `MCUCR` mora biti jednak 1, a bit `ISCO1` u registru `MCUCR` mora biti jednak 0. Ovu konfiguraciju napravili smo pomoću makronaredbe `set_bit_reg`.

Napravite konfiguraciju vanjskih prekida INT1 i INT2. Vanjski prekid INT1 mora generirati zahtjev za prekid na rastući brid signala. Prema tablici 11.2, bitove `ISC10` i `ISC11` u registru `MCUCR` potrebno je postaviti u vrijednost 1. Na pinu INT2 (PB2) zahtjev za prekid mora se generirati na padajući brid signala. U tom slučaju vrijednost bita `ISC2` u registru `MCUCSR` mora biti 0.

Vanjski prekidi INT0, INT1 i INT2 generirat će zahtjev za prekid ako je prekid za vanjske prekide INT0, INT1 i INT2 omogućen. Prekidi se omogućuju u registru `GICR` tako da na mjesto bitova `INT0`, `INT1` i `INT2` upišete vrijednost 1. Vanjski prekid INT0 omogućen je u funkciji `inicijalizacija()` naredbom `set_bit_reg(GICR,INT0)`; . Omogućite vanjske prekide INT1 i INT2. Makronaredbom `sei()` globalno smo omogućili prekide.

Pinovi PD2, PD3 i PB2 postavljeni su kao ulazni pinovi. Na pinovima PD2, PD3 i PB2 uključeni su pritezni otpornici s obzirom da koristimo tipkalo kao generator padajućih i rastućih bridova signala.

Za brojanje impulsa deklarirane su tri globalne varijable `brojac_int0`, `brojac_int1` i `brojac_int2`. 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 i rastući brid signala na pinu INT0 (PD2),
- u prekidnoj rutini `ISR(INT1_vect)` vrijednost varijable `brojac_int1` uvećava se za jedan na svaki padajući i rastući brid signala na pinu INT1 (PD3),
- u prekidnoj rutini `ISR(INT2_vect)` vrijednost varijable `brojac_int2` uvećava se za jedan na svaki padajući i rastući brid signala na pinu INT2 (PB2).

U `while` petlji vrijednosti se brojača bridova signala prikazuju na LCD displeju svakih 200 ms. Konfiguracija LCD displeja napravljena je u funkciji `inicijalizacija()`.

Prevedite datoteku `vjezba1111.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16. Pritisnite tipkala spojena na pinove PB0, PB1 i PB2 te promatrajte vrijednosti brojača bridova signala.

Zatvorite datoteku `vjezba1111.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 11.2 Zadaci - vanjski prekidi

### Zadatak 11.2.1

Napravite program kojim ćete:

- na pinu INT0 (PD2) brojati padajuće bridove signala s tipkala spojenog na pin PB1,
- na pinu INT1 (PD3) brojati padajuće i rastuće bridove signala s tipkala spojenog na pin PB3,
- na pinu INT2 (PB2) brojati rastuće bridove signala s tipkala spojenog na pin PB2.

Vrijednosti brojača bridova za pinove INT0, INT1 i INT2 prikazite na LCD displeju.

---



## Poglavlje 12

# Odabrani senzori i aktuatori

Velik je broj senzora i aktuatora koji se mogu spojiti na mikrokontroler ATmega16. U ovoj vježbi odabrali smo sljedeće senzore i aktuatore:

- tranzistor kao sklopka i relej,
- ultrazvučni senzor HC-SR04,
- temperaturni senzor LM35,
- temperaturni senzor DS18B20.

Senzore ćemo spojiti na mikrokontroler ATmega16 pomoću priključnica **Port A**, **Port B** i **Port D**. Za svaki od navedenih senzora i aktuatora napraviti ćemo programski kod kojim ćemo obrađivati signale sa senzora i slati signale na aktuatore.

### 12.1 Vježbe - odabrani senzori i aktuatori

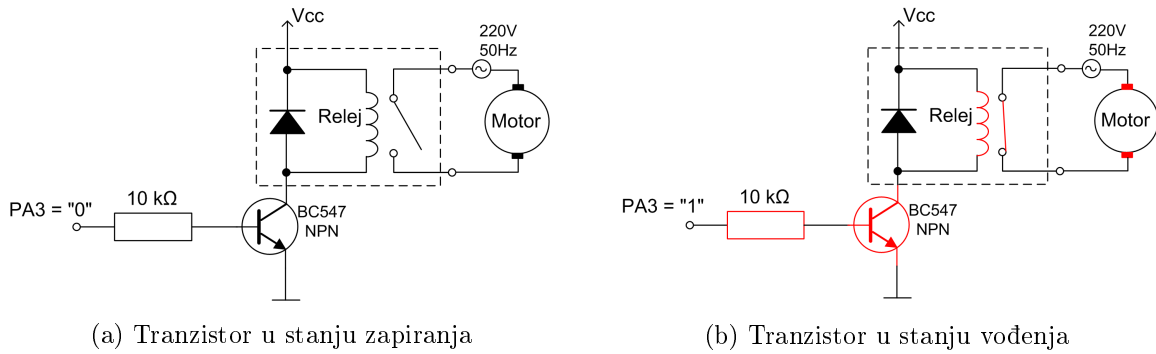
#### 12.1.1 Tranzistor kao sklopka i relej

Bipolarni tranzistor kao sklopka često se koristi za uključanje 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 ograničen je,
- tranzistor kao sklopka na svom izlazu može uklopiti samo istosmjerni napon.

Ove probleme možemo riješiti pomoću releja. Releji su elektromehaničke sklopke koji 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 uključanja trošila, što mu je glavni nedostatak uz malu frekvenciju rada.

U nastavku ćemo prikazati klasičan primjer koji se koristi za uključanje i isključenje izmjeničnog motora pomoću mikrokontrolera. Shema spajanja releja na mikrokontroler posredno pomoću tranzistora BC547 prikazana je na slici 12.1.

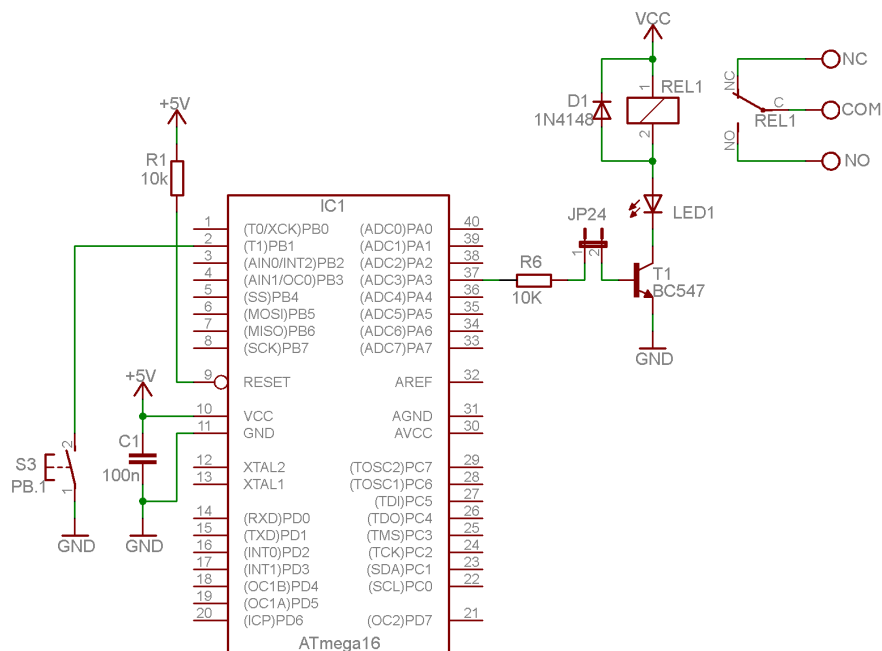


Slika 12.1: Shema spajanja releja na mikrokontroler posredno pomoću tranzistora BC547

U ovom primjeru pokazat ćemo korištenje bipolarnog tranzistora kao sklopke i releja. Na slici 12.1 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 mikrokontrolera pa se kao posrednik u uključenju i isključenju releja koristi bipolarni tranzistor. Bazu tranzistora potrebno je spojiti na digitalni pin mikrokontrolera (npr. pin PA3). Releji se postavlja u kolektorski krug tranzistora jer s pozicije tranzistora relej je trošilo. Paralelno releju potrebno je spojiti diodu koja štiti tranzistor od induciranog napona na zavojnici releja u trenutku njegovog isključenja. Pretpostavimo da je napon upravljačkog kruga releja jednak  $V_{cc}$  (npr.  $V_{cc} = 12\text{ V}$ ).

Kada na digitalni pin PA3 dovedemo nisko stanje (0 V), tranzistor će biti u stanju zapiranja (slika 12.1a). U tom slučaju kroz zavojnicu releja ne protječe struja i relej je isključen. Ako je relej isključen, prema slici 12.1a, izmjenični motor je također isključen.

Kada na digitalni pin PA3 dovedemo visoko stanje (5 V), tranzistor će biti u stanju vođenja (slika 12.1b). U tom slučaju kroz zavojnicu releja protječe struja i relej je uključen. Ako je relej uključen, prema slici 12.1b, izmjenični motor je također uključen.



Slika 12.2: Shema spajanja bipolarnog tranzistora BC547 s relejem u kolektorskom krugu na mikrokontroler ATmega16

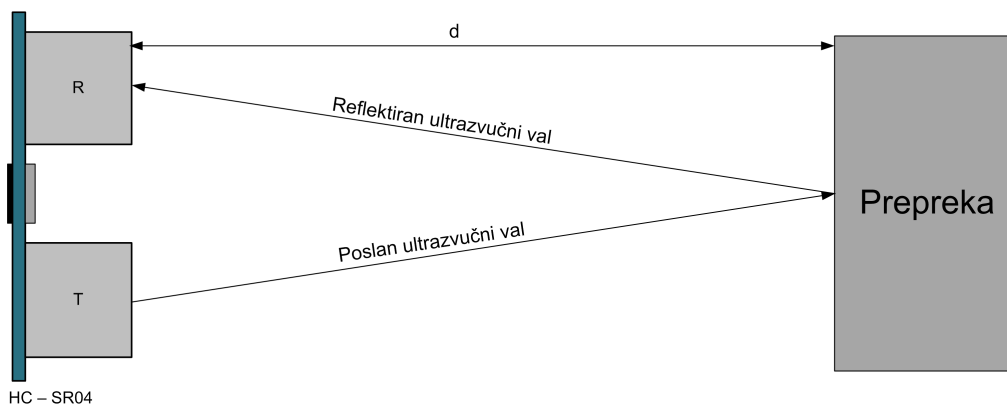
Ovaj primjer nam pokazuje da s malom upravljačkom strujom možemo upravljati velikom strujom trošila. Shema spajanja bipolarnog tranzistora BC547 s relejem u kolektorskom krugu na mikrokontroler ATmega16 prikazana je na slici 12.2. Baza bipolarnog tranzistora BC547 spojena je na pin PA3 preko kratkospojnika JP24. U kolektorskom krugu bipolarnog tranzistora BC547 nalazi se relej u seriji s LED diodom koja svijetli kada je relej uključen. Relej na slici 12.2 ima tri kontakta:

- NC (eng. *Normally Closed*) - kontakt koji je zatvoren sa zajedničkim kontaktom kada relej nije uključen,
- NO (eng. *Normally Open*) - kontakt koji je zatvoren sa zajedničkim kontaktom kada je relej uključen,
- COM (eng. *Common*) - zajednički kontakt.

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 PA3 visoko stanje (5 V). Ako je izmjenični motor spojen između kontakata NC i COM, tada će izmjenični motor biti uključen ako je na pinu PA3 nisko stanje (0 V).

### 12.1.2 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 12.3).

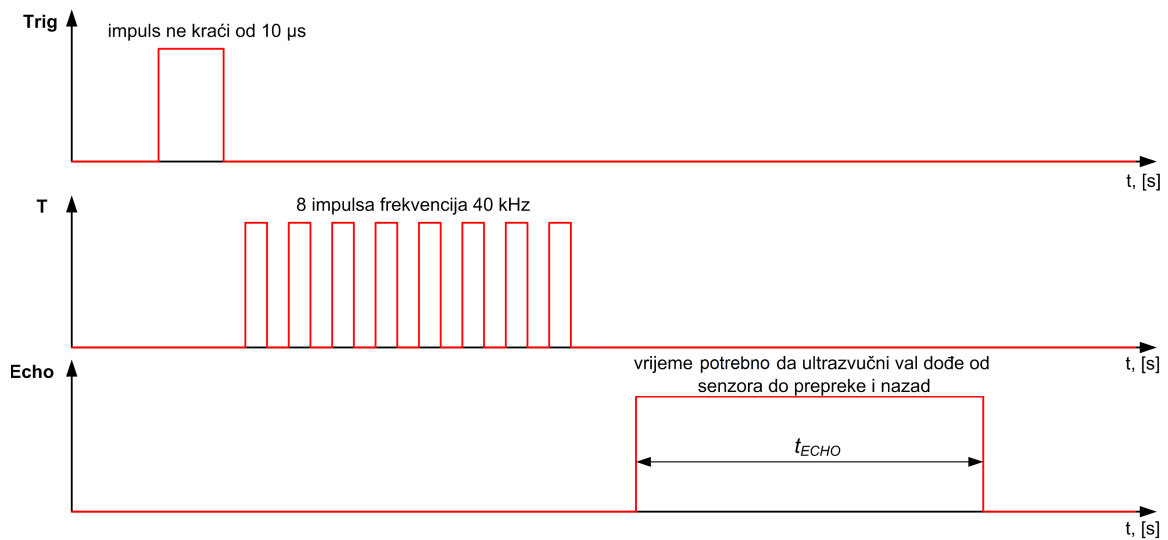


Slika 12.3: Rasprostiranje ultrazvučnog vala kroz prostor

Princip rada ultrazvučnog senzora HC-SR04 prikazan je vremenskim dijagramom na slici 12.4 [5]. Pomoću digitalnog pina mikrokontrolera 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 (eng. *Transmitter*) u prostor šalje ultrazvučni val, odnosno osam impulsa frekvencijom 40 kHz. Ultrazvučni val odbija se od prepreka i reflektira na mikروفon s oznakom R (eng. *Receiver*). Mikrokontroler 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  $t_{ECHO}$ .

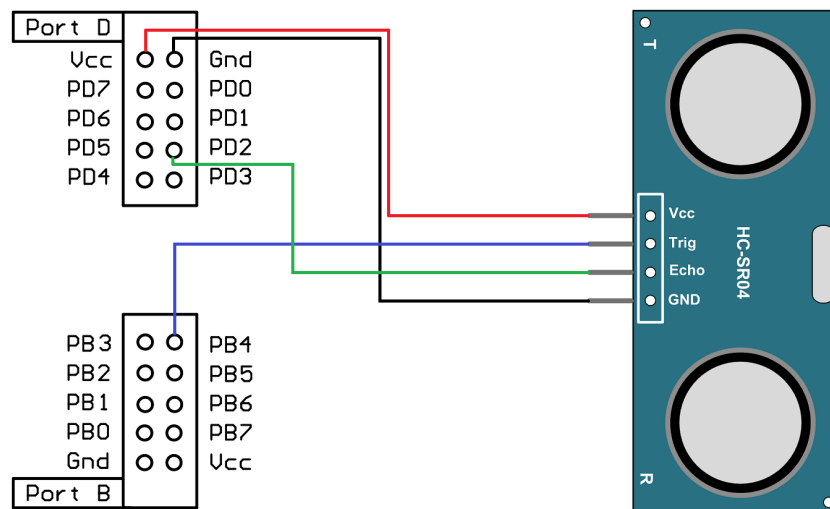
<sup>1</sup>Tip izlaza je *push-pull*. Ovakav izlaz može se spojiti na ulaz mikrokontrolera bez uključivanja priteznog otpornika.

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 mikrokontrolera.



Slika 12.4: Vremenski dijagram signala ultrazvučnog senzora HC-SR04

Shema spajanja ultrazvučnog senzora HC-SR04 pomoću priključnica Port B i Port D na mikrokontroler ATmega16 prikazana je na slici 12.5.



Slika 12.5: Shema spajanja ultrazvučnog senzora HC-SR04 pomoću priključnica Port B i Port D na mikrokontroler ATmega16

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 340 m/s. Umnožak  $t_{ECHO} \cdot v_z$  u relaciji (12.1) dijeli se s dva jer ultrazvučni val prevaljuje dvije udaljenosti prepreke od ultrazvučnog senzora HC-SR04.

### 12.1.3 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:

$$V_{out} = T \cdot 10 \frac{\text{mV}}{^{\circ}\text{C}} = T \cdot 0,01 \frac{\text{V}}{^{\circ}\text{C}}. \quad (12.2)$$

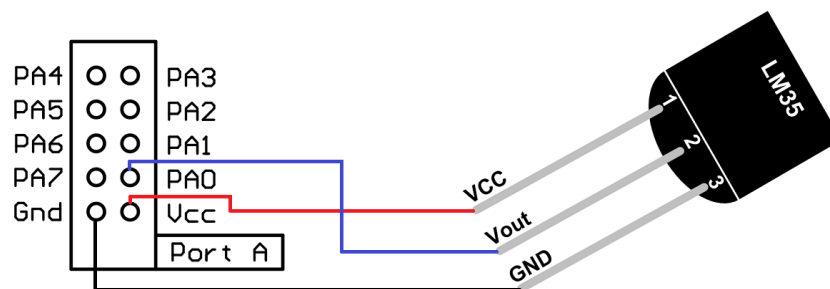
Temperaturu iz relacije (12.2) možemo izračunati na sljedeći način:

$$T = V_{out} \cdot 100 \frac{^{\circ}\text{C}}{\text{V}}. \quad (12.3)$$

Na primjer, ako je izlazni napon temperaturnog senzora LM35 jednak  $350\text{ mV}$ , temperatura u njegovoj okolini je  $35^{\circ}\text{C}$ .

Shema spajanja temperaturnog senzora LM35 pomoću priključnice **Port A** na mikrokontroler ATmega16 prikazana je na slici 12.6. Ovaj način spajanja osnovni je i omogućuje mjerenje temperature u opsegu od  $2^{\circ}\text{C}$  do  $150^{\circ}\text{C}$  s konstantom proporcionalnosti koja iznosi  $10\text{ mV}/^{\circ}\text{C}$ .

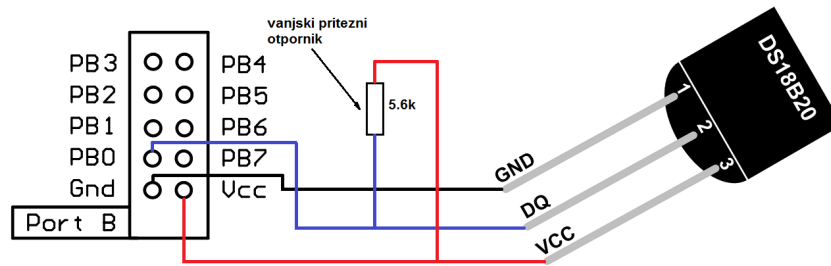
Pin  $V_{out}$  temperaturnog senzora LM35 spaja se na bilo koji analogni pin na portu A mikrokontrolera ATmega16. Analogno-digitalnom pretvorbom mjerimo napon na pinu  $V_{out}$  te pomoću relacije (12.3) izračunamo temperaturu u okolini temperaturnog senzora LM35.



Slika 12.6: Shema spajanja temperaturnog senzora LM35 pomoću priključnice **Port A** na mikrokontroler ATmega16

### 12.1.4 Temperaturni senzor DS18B20

Temperaturni senzor DS18B20 digitalni je mjerni senzor. Ovaj senzor komunicira s mikrokontrolerom pomoću tzv. *1-Wire* komunikacije. Podatkovni pin  $DQ$  temperaturnog senzora DS18B20 služi za dvosmjernu komunikaciju i zahtjeva vanjski pritezni otpornik. Shema spajanja temperaturnog senzora DS18B20 pomoću priključnice **Port B** na mikrokontroler ATmega16 prikazana je na slici 12.7. Ostale detalje o temperaturnom senzoru DS18B20 i o načinu komunikacije s mikrokontrolerom proučite u literaturi [7].



Slika 12.7: Shema spajanja temperaturnog senzora DS18B20 pomoću priključnice Port B na mikrokontroler ATmega16

S mrežne stranice [www.vtsbj.hr/mikroracunala](http://www.vtsbj.hr/mikroracunala) skinite datoteku **Senzori i aktuatori.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 **Senzori i aktuatori.zip** raspakirajte u novostvorenu datoteku na radnoj površini. Pozicionirajte se u novostvorenu datoteku na radnoj površini te dvostrukim klikom pokrenite **mikroracunala.atstln** u datoteci **\\Senzori i aktuatori\vjezbe**. U otvorenom projektu nalaze se sve naredne vježbe koje ćemo obraditi u poglavlju **Odabrani senzori i aktuatori**. Vježbe ćemo pisati u datoteke s ekstenzijom **\*.c**.

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



### Vježba 12.1.1

Napravite program kojim ćete pomoću tipkala koje je spojeno na pin PB1 uključivati relej. Shema spajanja bipolarnog tranzistora BC547 s relejem u kolektorskom krugu na mikrokontroler ATmega16 prikazana je na slici 12.2.

U projektnom stablu otvorite datoteku **vjezba1211.c**. Omogućite prevođenje samo datoteke **vjezba1211.c**. Početni sadržaj datoteke **vjezba1211.c** prikazan je programskim kodom 12.1.

Programski kod 12.1: Početni sadržaj datoteke **vjezba1211.c**

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

void inicijalizacija(){

    output_port(DDRA,PA3); // pin PA3 postavljen kao izlazni
    input_port(DDRB,PB1); // pin PB1 postavljen kao ulazni
    set_port(PORTB,PB1,1); // uključen pritezni otpornik na PB1
}

int main(void){

    inicijalizacija();

    while (1){

        set_port(PORTA,PA3,!get_pin(PINB,PB1));
    }
    return 0;
}
```

Bipolarni tranzistor koji uključuje relej spojen je na pin PA3 kojeg je potrebno konfigurirati kao izlazni pin. Pin PB1 konfiguriran je kao ulazni pin te je na njemu uključen pritezni otpornik. Navedene konfiguracije prikazane su u programskom kodu 12.1 u funkciji `inicijalizacija()`.

U `while` petlji kao stanje pina PA3 postavlja se negacija stanja na pinu PB1. Razlog tome je taj što relej uključujemo kada se pritisne tipkalo spojeno na pin PB1.

Prevedite datoteku `vjezba1211.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Promijenite programski kod tako da se relej uključuje ako tipkalo spojeno na pin PB1 nije pritisnuto. Ponovno prevedite datoteku `vjezba1211.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba1211.c` i onemogućite prevođenje ove datoteke.



### Vježba 12.1.2

Napravite program kojim ćete pomoću ultrazvučnog senzora HC-SR04 mjeriti udaljenost do preke u prostoru. Mjereni udaljenost prikažite na LCD displeju u centimetrima. Shema spajanja ultrazvučnog senzora HC-SR04 na mikrokontroler ATmega16 prikazana je na slici 12.5.

U projektnom stablu otvorite datoteku `vjezba1212.c`. Omogućite prevođenje samo datoteke `vjezba1212.c`. Početni sadržaj datoteke `vjezba1212.c` prikazan je programskim kodom 12.2.

Programski kod 12.2: Početni sadržaj datoteke `vjezba1212.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include <avr/interrupt.h>

uint16_t broj_impulsa = 0;

ISR(INT0_vect) // prekidna rutina za INTO
{
    // ako je brid rastući
    if(get_pin(PIND,PD2) == 1){
        TCNT1 = 0; // t1 = 0
    }
    else // ako je brid padajući
    {
        broj_impulsa = TCNT1; // t2 = TCNT1
    }
}

void inicijalizacija(){

    lcd_init();

    // oba brida generiraju prekid INTO
    set_bit_reg(MCUCR,ISC00);
    reset_bit_reg(MCUCR,ISC01);

    set_bit_reg(GICR,INT0); // omogućavanje vanjskog prekida INTO

    //F_CPU/8 za timer1
    reset_bit_reg(TCCR1B,CS10);
    set_bit_reg(TCCR1B,CS11);

    output_port(DDRB,PB4); // pin PB4 postavljen kao izlazni
```

```

    input_port(DDRD, PD2); // PD2 postavljen kao ulazni
    sei(); // globalno omogućavanje prekida
}

int main(void){

    inicijalizacija();

    float d;

    while (1){

        set_port(PORTB, PB4, 1);
        _delay_us(20); // trigger impuls
        set_port(PORTB, PB4, 0);
        _delay_ms(500);

        // relacija za proračun udaljenosti u cm

        // ispis udaljenosti

    }

    return 0;
}

```

Ulazni pin Trig ultrazvučnog senzora HC-SR04 spojen je prema shemi na slici 12.5 na pin PB4. Na pinu PB4 generirat ćemo impuls ne kraći od 10  $\mu$ s koji pokreće mehanizam mjerenja udaljenosti. Impuls kojeg 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 izlaznom pinu Echo (slika 12.4), pokrenite mjerenje vremena pomoću tajmera postavljenjem početne vrijednosti registra **TCNT1** u nulu,
- kada se pojavi padajući brid signala na izlaznom 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 kojeg broji registar **TCNT1** ovisi o djelitelju frekvencije radnog takta i frekvenciji radnog takta, a može se izračunati na sljedeći način:

$$t_{impulsa} = \frac{PRESCALER}{F_{CPU}}. \quad (12.4)$$

Vrijeme trajanja impulsa na izlaznom 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:

$$t_{ECHO} = broj\_impulsa \cdot t_{impulsa} = broj\_impulsa \frac{PRESCALER}{F_{CPU}}. \quad (12.5)$$

Frekvencija radnog takta je 8 MHz, a za djelitelja frekvencije radnog takta odabrat ćemo 8. Uvrstimo navedene parametre u relaciju (12.5):

$$t_{ECHO} = broj\_impulsa \frac{8}{8000000} = \frac{broj\_impulsa}{1000000}. \quad (12.6)$$



Vrijeme  $t_{ECHO}$  izračunato pomoću relacije (12.6) uvrstimo sada u relaciju (12.1):

$$d = \frac{t_{ECHO} \cdot v_z}{2} = \frac{\text{broj\_impulsa}}{1000000} \cdot 340. \quad (12.7)$$

Nakon sređivanja relacije (12.7) dobit ćemo konačnu relaciju za izračun udaljenosti prepreke od ultrazvučnog senzora HC-SR04 u metrima:

$$d[\text{m}] = \frac{\text{broj\_impulsa} \cdot 17}{100000}. \quad (12.8)$$

Naš je zadatak prikazati udaljenost prepreke od ultrazvučnog senzora HC-SR04 u centimetrima pa ćemo relaciju (12.9) pomnožiti sa konstantom  $\frac{100\text{cm}}{1\text{m}}$ :

$$d[\text{cm}] = \frac{\text{broj\_impulsa} \cdot 17}{1000}. \quad (12.9)$$

U programskom kodu 12.2 u funkciji `inicijalizacija` vanjski prekid INT0 konfiguriran je tako da se zahtjev za prekid generira na rastući i padajući brid signala. Za ovu konfiguraciju, prema tablici 11.1, bit `ISCO0` u registru `MCUCR` mora biti jednak 1, a bit `ISCO1` u registru `MCUCR` mora biti jednak 0. Ovu konfiguraciju napravili smo pomoću makronaredbe `set_bit_reg`. Prekid INT0 omogućuje se u registru `GICR` tako da na mjesto bita `INT0` upišete vrijednost 1.

Za mjerenje vremena  $t_{ECHO}$  koristit ćemo sklop `Timer/Counter1` kao tajmer u normalnom načinu rada. Prema tablicama 47 i 48 u literaturi [1] u funkciji `inicijalizacija()` konfiguriran je sklop `Timer/Counter1` kao tajmer u normalnom načinu rada s djeliteljem frekvencije radnog takta 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`. Za globalno omogućavanje prekida pozvana je makronaredba `sei()`.

Pin PB4 konfiguriran je kao izlazni, a pin INT0 (PD2) konfiguriran je kao ulazni pin. Izlazni pin Echo ultrazvučnog senzora HC-SR04 tipa je *push-pull* pa nije potrebno uključiti pritezni otpornik na pinu PD2.

Prekidna rutina `ISR(INT0_vect)` poziva se kod svakog padajućeg i rastućeg brida signala na pinu INT0 (PD2). Ako je brid signala na pinu INT0 (PD2) rastući, to znači da je pri pozivu prekidne rutine `ISR(INT0_vect)` stanje pina PD2 visoko. Ovu provjeru radimo pomoću naredbe `if(get_pin(PIND,PD2)== 1)`. Ako je brid signala na pinu PD2 rastući, početno stanje registra `TCNT1` postaviti ćemo u nulu. Sljedeći poziv prekidne rutine javit će se pri padajućem bridu na pinu INT0 (PD2). Kada se pojavi padajući brid signala na pinu PD2, vrijednost registra `TCNT1` sprema se u varijablu `broj_impulsa`.

U `while` petlji na pinu PB4 generiramo impuls u trajanju od 20  $\mu\text{s}$ , a nakon toga čekamo 500 ms da ultrazvučni senzor HC-SR04 provede mjerenje udaljenosti, a prekidna rutina `ISR(INT0_vect)` prikupi vrijeme trajanja impulsa na izlaznom pinu Echo.

U `while` petlju upišite relaciju (12.9) te na LCD ispišite udaljenost ultrazvučnog senzora HC-SR04 od prepreke u centimetrima na dva decimalna mjesta.

Prevedite datoteku `vjezba1212.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba1212.c` i onemogućite prevođenje ove datoteke.



### Vježba 12.1.3

Napravite program kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini. Mjerenu temperaturu prikažite na LCD displeju u °C. Shema spajanja temperaturnog senzora LM35 na mikrokontroler ATmega16 prikazana je na slici 12.6.

U projektnom stablu otvorite datoteku `vjezba1213.c`. Omogućite prevođenje samo datoteke `vjezba1213.c`. Početni sadržaj datoteke `vjezba1213.c` prikazan je programskim kodom 12.3.

Programski kod 12.3: Početni sadržaj datoteke `vjezba1213.c`

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include "ADC/adc.h"
#include <util/delay.h>

void inicijalizacija(){
    // konfigurirajte ADC i LCD
}

int main(void){

    inicijalizacija();

    uint16_t ADC5; // rezultat AD pretvorbe
    float Vout; // napon na pinu PA0
    float T; // temperatura u okolini senzora LM35
    const float VREF = 5.0; // AVCC

    while (1)
    {
        // izračunajte i ispišite temperaturu na LCD
    }

    return 0;
}
```

U programskom kodu 12.3 u funkciji `inicijalizacija()` konfigurirajte analogno-digitalnu pretvorbu i LCD displej. U `while` petlji svakih 500 ms na LCD displeju ispišite vrijednost temperature u okolini temperaturnog senzora LM35. Za proračun temperature koristite relacije (12.3) i (7.3).

Prevedite datoteku `vjezba1213.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Zatvorite datoteku `vjezba1213.c` i onemogućite prevođenje ove datoteke.



### Vježba 12.1.4

Napravite program kojim ćete pomoću temperaturnog senzora DS18B20 mjeriti temperaturu u njegovoj okolini. Mjerenu temperaturu prikažite na LCD displeju u °C. Shema spajanja temperaturnog senzora DS18B20 na mikrokontroler ATmega16 prikazana je na slici 12.7.

U projektnom stablu otvorite datoteku `vjezba1214.c`. Omogućite prevođenje samo datoteke `vjezba1214.c`. Početni sadržaj datoteke `vjezba1214.c` prikazan je programskim kodom 12.4.

Programski kod 12.4: Početni sadržaj datoteke vjezba1214.c

```
#include "AVR lib/AVR_lib.h"
#include <avr/io.h>
#include "LCD/lcd.h"
#include <util/delay.h>

void inicijalizacija(){
    lcd_init();
}

int main(void){
    inicijalizacija();

    float T; // temperatura u okolini senzora DS18B20

    while (1)
    {
        // proračun temperature T

        lcd_clrscr();
        lcd_home();
        lcd_print("T = %0.4f°C", T, 178);
        _delay_ms(500);
    }

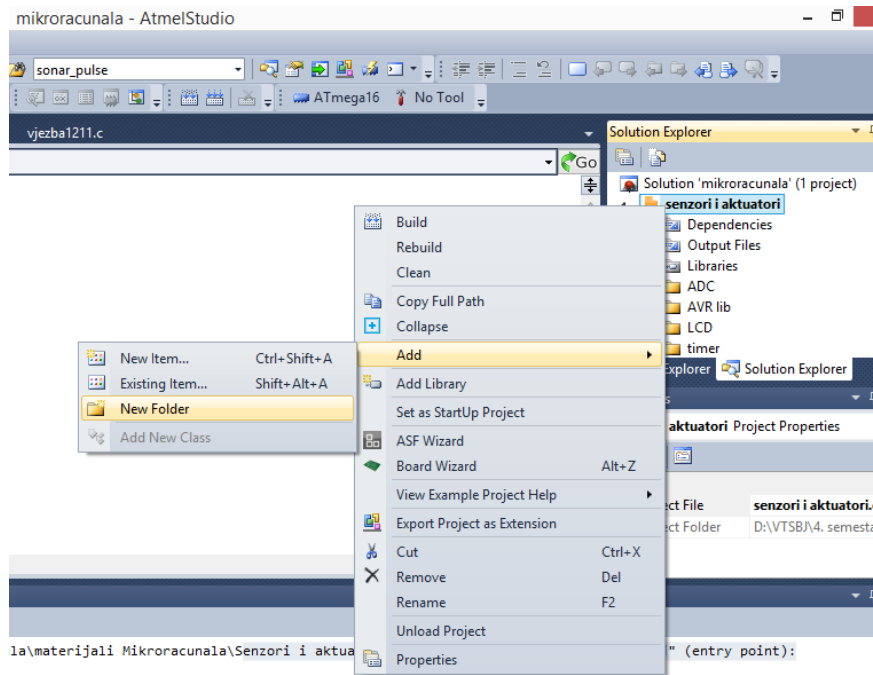
    return 0;
}
```

U ovoj vježbi imat ćemo tri pretpostavke:

1. o temperaturnom senzoru DS18B20 znamo da može mjeriti temperaturu, ali ne i princip na kojem radi senzor,
2. znamo da senzor s mikrokontrolerom komunicira pomoću jedne komunikacijske linije na koju je potrebno postaviti pritezni otpornik (npr. 5.6 kΩ),
3. na Internetu je moguće pronaći otvoreni programski kod (eng. *Open Source*) koji služi za čitanje temperature s temperaturnog senzora DS18B20, a napisan je u programskom jeziku C za mikrokontrolere porodice Atmel.

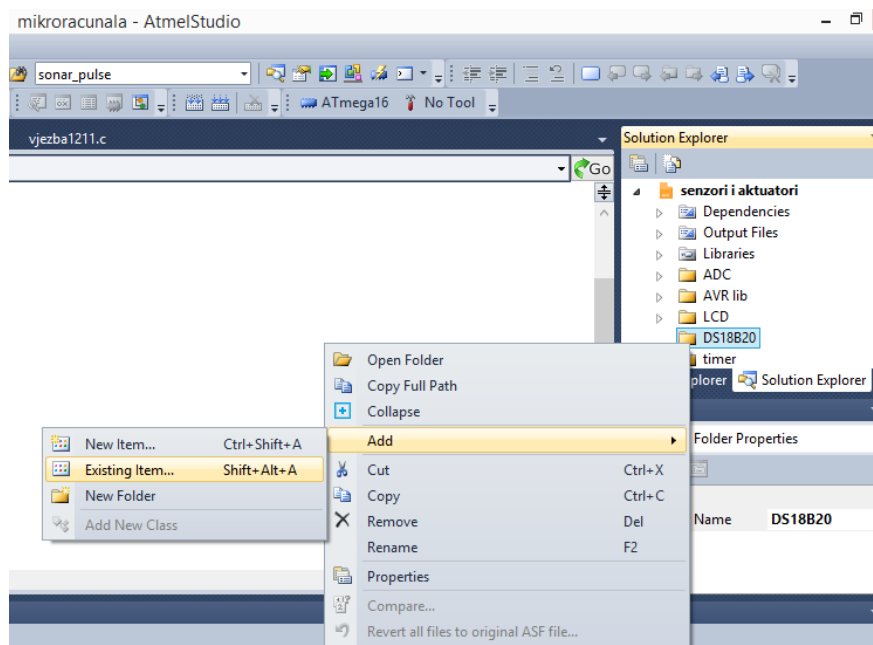
Česta je praksa u programiranju mikrokontrolera korištenje već postojećih rješenja koja su besplatna i dostupna na Internetu. Na radnoj površini stvorili ste vlastitu datoteku u kojoj se nalazi datoteka **Senzori i aktuatori**. U datoteci `\\Senzori i aktuatori\DS18B20` nalaze se definicije funkcija i konstanti koje se koriste za komunikaciju s temperaturnim senzorom DS18B20 napisane u datotekama `ds18b20.h` i `ds18b20.c`. Ove datoteke skinute su s Interneta i potrebno ih je dodati u otvoreni projekt **senzori i aktuatori** u programskom okruženju *Atmel Studio 6*.

Kreiranje datoteke DS18B20 u otvorenom projektu **senzori i aktuatori** u programskom okruženju *Atmel Studio 6* prikazano je na slici 12.8. Na datoteku **senzori i aktuatori** u programskom okruženju *Atmel Studio 6* potrebno je pritisnuti desni gumb miša. Nakon toga odaberite *Add* → *New Folder*. Ime *New Folder* promijenite u DS18B20 (slika 12.8).



Slika 12.8: Kreiranje datoteke DS18B20 u otvorenom projektu *senzori i aktuatori* u programskom okruženju *Atmel Studio 6*

Dodavanje datoteka `ds18b20.h` i `ds18b20.c` u novostvorenu datoteku DS18B20 prikazano je na slici 12.9. Na novostvorenu datoteku DS18B20 pritisnite desni gumb miša. Nakon toga odaberite *Add* → *Existing Item...*. U otvorenom prozoru pozicionirajte se u datoteku `\\Senzori i aktuatori\\DS18B20` te odaberite datoteke `ds18b20.h` i `ds18b20.c` i pritisnite *Add*.



Slika 12.9: Dodavanje datoteka `ds18b20.h` i `ds18b20.c` u novostvorenu datoteku DS18B20

Datoteke `ds18b20.h` i `ds18b20.c` sada su dostupne za korištenje u otvorenom projektu *senzori i aktuatori* u programskom okruženju *Atmel Studio 6*. Otvorite datoteke `ds18b20.h` i `ds18b20.c` i proučite ih.

U zaglavlju `ds18b20.h` potrebno je konfigurirati komunikacijski pin temperaturnog senzora DS18B20. Prema shemi na slici 12.7 komunikacijski pin temperaturnog senzora DS18B20 spojen je na pin PB0. Konfiguracija komunikacijskog pina prikazana je u programskom kodu 12.5. Temperaturni senzor DS18B20 možete spojiti na bilo koji digitalni pin mikrokontrolera ATmega16 te sukladno tome mijenjati konfiguraciju prikazanu programskim kodom 12.5.

Programski kod 12.5: Konfiguriranje komunikacijskog pina temperaturnog senzora DS18B20

```
#define ds18b20_PORT      PORTB
#define ds18b20_DDR      DDRB
#define ds18b20_PIN      PINB
#define ds18b20_DQ      PB0
```

U zaglavlju `ds18b20.h` deklarirana je funkcija `ds18b20_read_temperature()`. Ova funkcija koristi se za čitanje temperature s temperaturnog senzora DS18B20. U programski kod 12.4 pomoću naredbe `#include "DS18B20/ds18b20.h"` uključite zaglavlje `ds18b20.h` u datoteku koja se prevodi. U `while` petlju dodajte naredbu `T = ds18b20_read_temperature();`.

Prevedite datoteku `vjezba1214.c` u strojni kod i snimite ga na mikrokontroler ATmega16. Testirajte program na razvojnom okruženju s mikrokontrolerom ATmega16.

Ova vježba primjer je korištenja senzora o kojem ne znamo puno, ali za njega postoje napisane datoteke s funkcijama za komunikaciju koje možemo koristiti.

Zatvorite datoteku `vjezba1214.c` i onemogućite prevođenje ove datoteke. Zatvorite programsko razvojno okruženje *Atmel Studio 6*.

## 12.2 Zadaci - odabrani senzori i aktuatori

### Zadatak 12.2.1

Napravite program kojim ćete pomoću tipkala koje je spojeno na pin PB3 uključivati relej. Shema spajanja bipolarnog tranzistora BC547 s relejem u kolektorskom krugu na mikrokontroler ATmega16 prikazana je na slici 12.2.

---

### Zadatak 12.2.2

Napravite program kojim ćete pomoću ultrazvučnog senzora HC-SR04 mjeriti udaljenost do prepreke u prostoru. Mjerenu udaljenost prikažite na LCD displeju u milimetrima. U programu napravite dio koda za signalizaciju LED diodama na sljedeći način:

- ako je udaljenost veća i jednaka 3000 mm, neka je uključena samo bijela LED dioda,
- ako je udaljenost veća i jednaka 2000 mm i manja od 3000 mm, neka su uključene bijela i zelena LED dioda,
- ako je udaljenost veća i jednaka 500 mm i manja od 2000 mm, neka su uključene bijela, zelena i žuta LED dioda,
- ako je udaljenost manja od 500 mm, neka su uključene sve LED diode.

Shema spajanja ultrazvučnog senzora HC-SR04 na mikrokontroler ATmega16 prikazana je na slici 12.5.

---

### Zadatak 12.2.3

Napravite program kojim ćete pomoću temperaturnog senzora LM35 mjeriti temperaturu u njegovoj okolini. Mjerenu temperaturu prikažite na LCD displeju u °C. Shema spajanja temperaturnog senzora LM35 na mikrokontroler ATmega16 prikazana je na slici 12.6. Dodatno je potrebno napraviti prekidnu rutinu `ISR(TIMER1_OVF_vect)` koja će svakih 400 ms putem serijske komunikacije u aplikaciju sa slike 10.4 slati vrijednost temperature u obliku proizvoljne tekstualne poruke. Kontrolni okvir `Omogući primljene poruke` mora biti označen. Brzinu prijenosa podataka postavite na 19200 b/s.

---

### Zadatak 12.2.4

Napravite program kojim ćete pomoću temperaturnog senzora DS18B20 mjeriti temperaturu u njegovoj okolini. Mjerenu temperaturu prikažite na LCD displeju u °C. Shema spajanja temperaturnog senzora DS18B20 na mikrokontroler ATmega16 prikazana je na slici 12.7. Dodatno je potrebno napraviti prekidnu rutinu `ISR(TIMER1_OVF_vect)` koja će svakih 400 ms putem serijske komunikacije u aplikaciju sa slike 10.4 slati vrijednost temperature u obliku proizvoljne tekstualne poruke. Kontrolni okvir `Omogući primljene poruke` mora biti označen. Brzinu prijenosa podataka postavite na 19200 b/s.

---

# Bibliografija

- [1] ATMEL, [www.atmel.com/Images/doc2466.pdf](http://www.atmel.com/Images/doc2466.pdf), *8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash, ATmega16, ATmega16L*, 2010.
- [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 MIKORORAČUNALA*. Visoka tehnička škola u Bjelovaru, [www.vtsbj.hr/mikroracunala-predavanja-vjezbe/](http://www.vtsbj.hr/mikroracunala-predavanja-vjezbe/), Bjelovar, 2013.
- [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] Maxim Integrated, <http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>, *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*, 2008.







[www.vtsbj.hr](http://www.vtsbj.hr)

ISBN 9789537676179



9 789537 676179