Prekidi i tajmeri

Programska podrška sistemu prekida

Kompajler AVR-GCC pruža mogućnost upravljanja prekidima AVR familije mikrokontrolera na jednostavan način. Ovo će biti ilustrovano primerima koji koriste prekide Tajmera 0, kao i prekide izazvane promenom stanja ulaznih pinova (eng. *Pin Change Interrupt*).

Za početak, potrebno je uključiti biblioteku koja podržava rad sa prekidima:

```
#include <avr/interrupt.h>
```

Da bi prekid bio omogućen, potrebno je obezbediti sledeće:

- Mora biti setovan bit za dozvolu prekida u odgovarajućem konfiguracionom registru.
- Mora biti setovan bit I u statusnom registru. U slučaju kada je I=0 podrazumeva globalnu zabranu svih prekida (izuzev sistemskih, odnosno nemaskirajućih). Bit I se setuje pomoću makroa **sei()**, a resetuje pomoću makroa **cli()**.
- Mora postojati prekidna rutina. Prekidne rutine su funkcije koje se automatski pozivaju prilikom pojave zahteva za prekidom, pod uslovom da je prekid dozvoljen. Prekidne rutine deklarišu se na sledeći način:

```
ISR(vektor_prekida)
{
    // telo prekidne rutine...
}
```

Kao parametar prekidne rutine navodi se vektor prekida, u zavisnosti od toga šta je izvor prekida. Mikrokontroler Atmega328P ima 25 vektora prekida + reset vektor (na adresi 0 u programskoj memoriji). Definicije vektora prekida navedene su u nastavku (redni broj vektora prekida ujedno predstavlja i njegov nivo prioriteta: 0 - najviši, 25 - najniži) i dostupne su korisniku ukoliko je u okviru programa uključena biblioteka interrupt.h:

```
/* Interrupt Vectors */
/* Interrupt Vector 0 is the reset vector. */
#define INT0_vect            _VECTOR(1)  /* External Interrupt Request 0 */
#define INT1_vect            _VECTOR(2)  /* External Interrupt Request 1 */
#define PCINT0_vect          _VECTOR(3)  /* Pin Change Interrupt Request 0 */
#define PCINT1_vect          _VECTOR(4)  /* Pin Change Interrupt Request 1 */
#define PCINT2_vect          _VECTOR(5)  /* Pin Change Interrupt Request 2 */
#define WDT_vect             _VECTOR(6)  /* Watchdog Time/out Interrupt */
#define TIMER2_COMPA_vect    _VECTOR(7)  /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect    _VECTOR(8)  /* Timer/Counter2 Compare Match B */
#define TIMER2_OVF_vect      _VECTOR(9)  /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect     _VECTOR(10) /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect    _VECTOR(11) /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect    _VECTOR(12) /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect      _VECTOR(13) /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect    _VECTOR(14) /* Timer/Counter0 Compare Match A */
#define TIMER0_COMPB_vect    _VECTOR(15) /* Timer/Counter0 Compare Match B */
#define TIMER0_OVF_vect      _VECTOR(16) /* Timer/Counter0 Overflow */
#define SPI_STC_vect         _VECTOR(17) /* SPI Serial Transfer Complete */
#define USART_RX_vect        _VECTOR(18) /* USART Rx Complete */
#define USART_UDRE_vect      _VECTOR(19) /* USART, Data Register Empty */
#define USART_TX_vect        _VECTOR(20) /* USART Tx Complete */
#define ADC_vect             _VECTOR(21) /* ADC Conversion Complete */
#define EE_READY_vect        _VECTOR(22) /* EEPROM Ready */
#define ANALOG_COMP_vect     _VECTOR(23) /* Analog Comparator */
#define TWI_vect             _VECTOR(24) /* Two-wire Serial Interface */
#define SPM_READY_vect       _VECTOR(25) /* Store Program Memory Read */
```

Tajmer0

Ovde će biti objašnjen rad sa prekidima tajmera 0 ATmega328P mikrokontrolera. Tajmer 0 je 8-bitni tajmer što znači da njegov brojački registar može da menja vrednosti u opsegu brojeva 0 do $2^8 - 1$, odnosno od 0 do 255. Pored njega, ovaj mikrokontroler sadrži još dva tajmera: 16-bitni tajmer 1 i 8-bitni tajmer 2. S obzirom da je korišćenje tajmera 1 i 2 analogno korišćenju tajmera 0, oni neće biti zasebno razmatrani u nastavku.

U zavisnosti od toga kako konfigurišemo tajmer, on može raditi u nekom od sledećih režima rada:

- **Normalni režim** – brojački registar tajmera se uvećava redom od 0 do $2^8 - 1$ (kod tajmera 0 i 2), odnosno do $2^{16} - 1$ (kod tajmera 1). Nakon što dostigne maksimalnu vrednost, prekoračuje se opseg i brojački registar ponovo nastavlja da broji od 0. Prilikom prekoračenja opsega se generiše signal koji se može iskoristiti kao izvor prekida.
- **CTC (*Clear Timer on Compare Match*) režim** – u ovom režimu se opseg brojanja brojačkog registra ograničava odgovarajućim registrom (**OCRnA** ili **OCRnB**, objašnjeni ispod naredne slike). Kada brojački registar dostigne vrednost koja je zadata nekim od

tih registra, on se resetuje na 0 i ponovo nastavlja da broji do zadate granice. Prilikom reseta se generiše signal koji se može iskoristiti kao izvor prekida.

- **Brzi PWM režim**
- **Fazno korektni PWM režim**

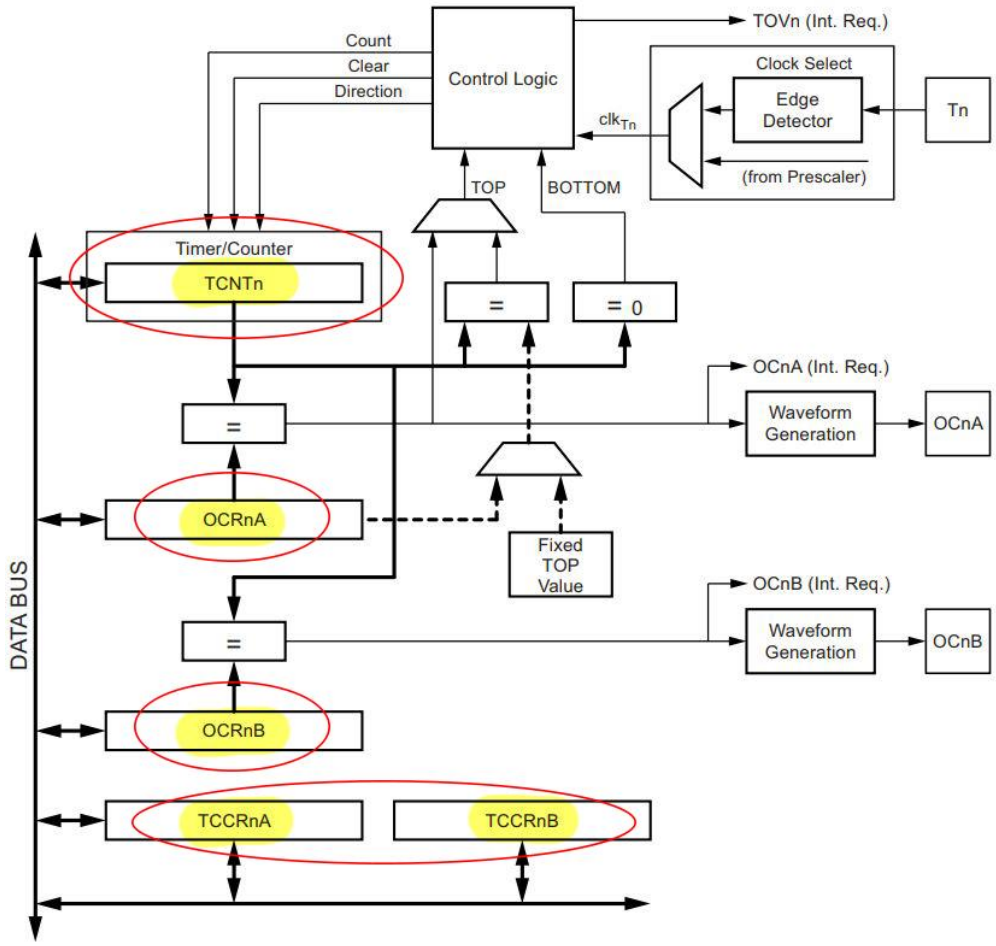
} ova dva režima rada izlaze iz okvira ovog kursa, pa neće biti detaljnije razmatrani.

Jasno je, na osnovu režima rada, da je moguće generisati prekide od strane tajmera svaki put kada se njegov brojački registar resetuje na 0. Na primer, ukoliko je tajmer 0 u normalnom režimu rada i ukoliko mu je frekvencija rada jednaka 1kHz, onda bi on generisao prekide svakih $256 \cdot 1/1\text{kHz} = 256\text{ms}$. S obzirom da je frekvencija oscilatora mikrokontrolera ATmega328P na Arduino UNO pločici jednaka $f_{osc} = 16\text{MHz}$, ako bi tajmer radio sa istom frekvencijom generisao bi prekide jako velike učestanosti (reda mikrosekundi). Kako bi se pravili prekidi manje učestanosti, signal od oscilatora se dovodi na tajmer preko **preskalera** – kola koje omogućuje da se inicijalna frekvencija od 16MHz deli sa određenim faktorom N (1, 8, 64...), čime postižemo da tajmer radi na manjoj frekvenciji. Takođe, kako bismo generisali prekide tačno određene učestanosti (npr. 1ms, 100ms, itd.), tajmer ćemo uvek koristiti u **CTC** režimu rada.

Generalno, prilikom rada sa bilo kojim hardverom, poželjno je uvek proučiti njegovu tehničku dokumentaciju (eng. *datasheet*) i iz nje izvući informacije od interesa (npr. kakvo napajanje je potrebno obezbediti, koje module taj hardver sadrži u sebi, kako se njima upravlja i slično). Za mikrokontroler ATmega328P, tehnička dokumentacija je data u fusnoti[1]. U našem slučaju, zanima nas modul tajmer 0 ovog mikrokontrolera i kako njime da upravljamo. Bitno je imati na umu da, iako tehničke dokumentacije često mogu imati po nekoliko stotina (ako ne i hiljada) stranica, nama je od interesa da pronađemo samo informacije koje su nam potrebne. Zbog toga se korišćenje tehničke dokumentacije svodi na njeno pretraživanje, a ne na detaljno čitanje od početka do kraja.

Struktura tajmerskog modula, predstavljena u tehničkoj dokumentaciji, je data u nastavku:

[1] https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf



Registri koji su nama od primarnog značaja su označeni na slici (slovo n u nazivima registara zamenjuje indeks odgovarajućeg tajmera, tj. 0, 1 i 2), a ti registri su sledeći:

- **TCNTn** – brojački registar, odnosno registar čija se vrednost uvećava/smanjuje frekvencijom na kojoj je podešen da radi tajmer;
- **OCRnA** i **OCRnB** – oba ova registra se mogu koristiti za poređenje njihove vrednosti i vrednosti brojačkog registra (uočiti komparatore na koje su spojeni), čime se može generisati signal kada brojački registar dostigne njihovu vrednost, odnosno prekid u CTC režimu rada;
- **TCCRnA** i **TCCRnB** – ova dva registra predstavljaju konfiguracione registre tajmera, kojim se određuje režim rada tajmerskog modula.

Pored ovih registara, bitan je i registar **TIMSKn**, kojim se postavlja/uklanja dozvola prekida odgovarajućeg tajmera.

Bez zalaženja u detalje korišćenja konfiguracionih registara, ovde će biti naglašeno kako će se oni koristiti za potrebe ovog kursa:

- TCCRnA registar će se koristiti za postavljanje tajmera u CTC režimu rada. Za tajmer 0, to se ralizuje dodelom vrednosti `0x02` TCCR0A registru (odnosno setovanjem bita 2 ovog registra);
- TCCRnB registar će se koristiti za određivanje vrednosti preskalera. Za tajmer 0, to se realizuje dodeljivanjem vrednosti u skladu sa sledećom tabelom:

TCCR0B	N
0x00	Tajmer je isključen
0x01	1
0x02	8
0x03	64
0x04	256
0x05	1024

- TIMSKn će se koristiti za dozvolu prekida tajmera. Za tajmer 0, to se realizuje dodelom vrednosti `0x02` TIMSK0 registru (odnosno setovanjem bita 2 ovog registra).

Kao što je spomenuto, u zadacima ćemo koristiti tajmer 0 u CTC režimu rada, pri čemu ćemo za ograničavanje brojačkog registra koristiti registar OCR0A. Vrednost ovog registra, kao i preskalera ćemo određivati na osnovu sledeće formule:

$$f_T = \frac{f_{osc}}{N \cdot (OCR0A+1)},$$

gde $\mathbf{f_{osc}}$ predstavlja frekvenciju oscilatora (16 MHz), $\mathbf{f_T}$ - frekvenciju generisanja prekida, **N** - vrednost preskalera (1, 8, 64...) i **OCR0A** - vrednost OCR0A registra.

Zadatak:

Podesiti tajmer 0 tako da se njegovom upotrebom upravlja jednom LED tako da:

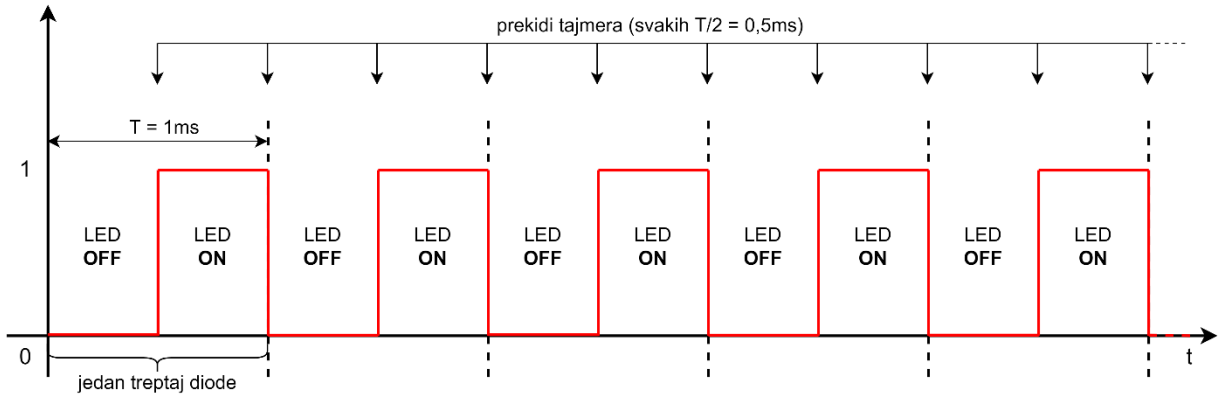
a) ona trepće frekvencijom od 1kHz

b) ona trepće frekvencijom od 1Hz (pola sekunde svetli-pola ne svetli)

Rešenje:

a) Frekvencija oscilatora iznosi $\mathbf{f_{osc}}$ **= 16MHz**. S obzirom da se u zadatku traži da LED **trepće** frekvencijom od 1kHz, odnosno periodom 1ms, to znači da je potrebno da se za 1ms dioda upali

i ugasi. Drugim rečima, potrebno generisati prekide duplo veće frekvencije kako bi se dioda prvo upalila (prvi prekid) i zatim ugasila (drugi prekid) za vremenski period od 1ms. Dakle, frekvencija prekida tajmera je u ovom slučaju $\mathbf{f_T = 2\ kHz}$. Ovim se efektivno postiže generisanje periodičnog signala, što je prikazano na slici ispod (signal je označen crvenom bojom):



Tajmer ćemo koristiti u **CTC** režimu rada i u skladu sa time formulu koja je prikazana ranije[2], na osnovu koje ćemo odrediti vrednost OCR0A registra, kao i vrednost preskaliranja:

$$f_T = \frac{f_{osc}}{N \cdot (OCR0A+1)} \Rightarrow OCR0A = \frac{f_{osc}}{N \cdot f_T} - 1$$

U ovoj formuli su nam vrednosti f_{osc} i f_T poznate, dok je potrebno odrediti vrednosti OCR0A i N. S obzirom da N može imati samo određene vrednosti, na osnovu tabele koja je prikazana ranije, tj. vrednosti 1, 8, 64, 256 i 1024, potrebno je proizvoljno odabrati jednu od tih vrednosti tako da, kada se zameni u formulu za OCR0A, dobijemo vrednost OCR0A registra koja je u opsegu brojeva od 0 do 255 (jer je to 8-bitni registar i ne može imati veću vrednost). Na primer, ako uzmemo za N da je 8, dobijamo sledeće:

f_{osc} = 16MHz, f_T = 2 kHz, $N = 8 \Rightarrow OCR0A = 999$

Pošto smo dobili vrednost OCR0A koja je veća od 255, N ne može biti jednako 8, već je potrebno uzeti veću vrednost:

f_{osc} = 16MHz, f_T = 2 kHz, $N = 64 \Rightarrow OCR0A = 124$

Na osnovu ovih vrednost, uz potrebnu konfiguraciju tajmera tako da radi u CTC režimu i uz omogućavanje njegovih prekida, uspešno ćemo generisati prekide frekvencije 2kHz, tj. prekide na svakih 0.5 milisekundi, a pri svakom prekidu ćemo paliti/gasiti diodu.

[2] Ekvivalentan postupak bi bio i korišćenje slične formule, za generisanje periodičnog signala kod ATmega328p mikrokontrolera, kod koje umesto N figuriše 2N, a umesto f_T figuriše frekvencija signala (1kHz u našem primeru).

```c
#include <avr/io.h>
#include <avr/interrupt.h>

#pragma GCC optimize("O0") // iskljucivanje optimizacije kompajlera

unsigned char slika_led = 0xff;

ISR(TIMER0_COMPA_vect)
{
    // prekid tajmera 0 usled dostizanja vrednosti registra OCR0A
    slika_led ^= 0x01;
    PORTD = slika_led;
}

int main()
{
    DDRD = 0xff;          // port D -> izlaz
    DDRB |= 1 << 4;       // PB4 -> izlaz
    PORTB &= ~(1 << 4); // PB4 = 0, cime se ukljucuje tranzistor Q0
    PORTD = slika_led;  // inicijalno su sve diode ugasene

    // inicijalizacija tajmera 0:
    TCCR0A = 0x02;        // tajmer 0: CTC mod
    TCCR0B = 0x03;        // tajmer 0: fclk = fosc/64
    OCR0A = 124;          // prekid tajmera 0: 125 Tclk (OCR0A + 1 = 125)

    TIMSK0 = 0x02;        // dozvola prekida tajmera 0 usled dostizanja
                          // vrednosti registra OCR0A

    sei();                // I = 1 (globalna dozvola prekida)


    while(1);      // glavni program se vrti u praznoj petlji i ceka prekide

    return 0;
}
```

b) Frekvencija oscilatora iznosi $\mathbf{f_{osc} = 16\ MHz}$, a frekvencija prekida tajmera je u ovom slučaju $\mathbf{f_T}$ = 2·1 Hz = **2Hz**. Potrebno je koristiti **CTC** mod tajmera.

$$f_T = \frac{f_{osc}}{N \cdot (OCR0A+1)} \quad \Rightarrow \quad OCR0A = \frac{f_{osc}}{N \cdot f_T} - 1$$

U ovom slučaju, čak i za najveću vrednost N = 1024, ne možemo dobiti vrednost OCR0A koja je manja od 256, pa samim tim nije moguće generisati ovakve prekide pomoću samog tajmera. Kako bi se ovo rešilo, potrebno je podesiti tajmer tako da pravi prekide proizvoljno velike učestanosti, koju može da postigne, pri čemu će se pri svakom prekidu uvećavati odgovarajuća promenljiva u softveru. Na primer, ukoliko uzmemo da je N = 256, za OCR0A dobijamo:

$$OCR0A = 31249 \approx 31250$$

Pošto OCR0A ne može imati vrednost 31249, možemo uzeti proizvoljno da je njegova vrednost 249, a da se pri svakom prekidu uvećava određena promenljiva sve dok ona ne dođe do 125, čime efektivno dobijamo (249 + 1) · 125 = 31250.

```
#include <avr/io.h>
#include <avr/interrupt.h>

#pragma GCC optimize("O0") // iskljucivanje optimizacije kompajlera

unsigned char slika_led = 0xff;
unsigned char t0_cnt = 0;

ISR(TIMER0_COMPA_vect)
{
    // prekid tajmera 0 usled dostizanja vrednosti registra OCR0A
    t0_cnt++;
}

int main()
{
    DDRD = 0xff;          // port D -> izlaz
    DDRB |= 1 << 4;       // PB4 -> izlaz
    PORTB &= ~(1 << 4); // PB4 = 0, cime se ukljucuje tranzistor Q0
    PORTD = slika_led;  // inicijalno su sve diode ugasene

    // inicijalizacija tajmera 0:
    TCCR0A = 0x02;        // tajmer 0: CTC mod
    TCCR0B = 0x04;        // tajmer 0: fclk = fosc/256
    OCR0A = 249;          // prekid tajmera 0: 250 Tclk (OCR0A + 1 = 250)

    TIMSK0 = 0x02;        // dozvola prekida tajmera 0 usled dostizanja
                          // vrednosti registra OCR0A

    sei();                // I = 1 (dozvola prekida)


    while(1)
    {
        if (t0_cnt == 125){
            t0_cnt = 0;
            slika_led ^= 0x01;
            PORTD = slika_led;
        }
    }
    return 0;
}
```

NAPOMENA: Kako bi programi ispravno funkcionisali potrebno je isključiti optimizacije kompajlera, što se za GCC kompajler ostvaruje sa:

```
#pragma GCC optimize("O0")
```

Ovo je neophodno uraditi jer koristimo globalne promenljive koje se menjaju u prekidnim rutinama (u našem slučaju promenljiva `t0_cnt`, koja se menja u prekidnoj rutini tajmera 0). Ako to ne uradimo, nakon optimizacije kompajlera, neće se registrovati promene vrednosti promenljive unutar prekidne rutine i samim tim program neće ispravno funkcionisati. Ovo se mora uraditi svaki put kada radimo sa prekidima i imamo globalne promenljive koje koristimo u prekidnim rutinama. Oni koji su zainteresovani mogu istražiti više o *volatile* tipu promenljivih (ukoliko bismo promenljivu `t0_cnt` deklarisali kao *volatile*, ne bismo morali isključivati optimizacije kompajlera), ali to izlazi iz okvira ovog kursa.

Zadatak:

Podesiti tajmer 0 tako da izaziva prekid 1000 puta u sekundi (tj. jednom svake milisekunde), a zatim iskoristiti prekidnu rutinu za automatski ispis karaktera na 7SEG displeju.

Rešenje:

Frekvencija oscilatora iznosi f_{osc} = 16MHz, a frekvencija prekida tajmera je u ovom slučaju f_T = 1 kHz. Potrebno je koristiti **CTC** mod tajmera, gde je moguće precizno podesiti periodu brojanja. U **CTC** modu, vrednost komparatorskog registra (**OCR0A**) računa se na sledeći način:

$$f_T = \frac{f_{osc}}{N \cdot (OCR0A+1)}$$

$$\Rightarrow \quad OCR0A = \frac{f_{osc}}{N \cdot f_T} - 1 \quad \Rightarrow \quad N = 64,\ OCR0A = 249$$

```
#include <avr/io.h>
#include <avr/interrupt.h>

#pragma GCC optimize("O0") // iskljucivanje optimizacije kompajlera

const unsigned char simboli[] = {
    0x0c, 0xa4, 0x27, 0xc4
}; // look-up tabela sa simbolima

unsigned char DISP_BAFER[4] = {
    0xfe, 0xfe, 0xfe, 0xfe
}; // bafer displeja

unsigned long ms = 0;
unsigned char disp = 3;

ISR(TIMER0_COMPA_vect)
{
    // prekid tajmera 0 usled dostizanja vrednosti registra OCR0A
    if (++disp > 3)
        disp = 0;
    PORTB = ~(1 << (3 - disp)); // ukljucenje tranzistora
    PORTD = DISP_BAFER[disp];  // ispis na trenutno aktivan displej

    ms++;  // uvecavanje proteklih milisekundi
}

int main()
{
    unsigned long t0;
```

```
    unsigned char i;

    // inicijalizacija portova:
    DDRB = 0x0f;  // PB3-PB0 -> izlaz
    DDRD = 0xff;  // port D -> izlaz

    // inicijalizacija tajmera 0:
    TCCR0A = 0x02;  // tajmer 0: CTC mod
    TCCR0B = 0x03;  // tajmer 0: fclk = fosc/64
    OCR0A = 249;    // prekid tajmera 0: 250 Tclk (OCR0A + 1 = 250)
    TIMSK0 = 0x02;  // dozvola prekida tajmera 0 usled dostizanja
                    // vrednosti registra OCR0A

    sei();   // I = 1 (dozvola prekida)

    while(1)
    {
        t0 = ms;
        while((ms - t0) < 500); // pauza 500ms
        for (i = 0; i < 4; i++)
            DISP_BAFER[i] = simboli[i];
        t0 = ms;
        while((ms - t0) < 500); // pauza 500ms
        for (i = 0; i < 4; i++)
            DISP_BAFER[i] = 0xfe;
    }
    return 0;
}
```

Zadatak za domaći:

Korišćenjem prekida tajmera 0, napisati program koji svake sekunde inkrementira stanje brojača i ispisuje ga na 7SEG displeju. Početno stanje brojača je 0.

Napredni zadatak:

Realizovati neki od prethodnih zadataka, korišćenjem tajmera 1, kao i tajmera 2, umesto tajmera 0 (pogledati njihove registre u tehničkoj dokumentaciji mikrokontrolera).

Prekid usled promene stanja ulaznih pinova

Zadatak:

Realizovati elektronsku kockicu na sledeći način: na 7SEG displeju D4 (krajnji displej sa desne strane) ispisuju se redom decimalne cifre 1-6, frekvencijom 25Hz. Pritiskom na taster DESNO, "obrtanje kockice" se zaustavlja i na displeju se zadržava cifra koja je bila prikazana u trenutku pritiska tastera. Proces "obrtanja" se nastavlja nakon pritiska na taster LEVO.

Rešenje:

Slično prethodnom primeru, biće korišćena prekidna rutina tajmera 0 za upravljanje displejom. U ovom slučaju moguć je stacionaran prikaz na displeju, što znači da nema potrebe za osvežavanjem u vremenskom multipleksu, pošto je displej D4 permanentno uključen. Ukoliko se tajmer 0 ponovo podesi na identičan način (prekid na 1ms), a promena stanja kockice se dešava sa periodom 1/25Hz = 40ms, to znači da je stanje kockice potrebno ažurirati jednom na svakih 40 prekida.

Logiku detekcije pritiska tastera moguće je ostvariti korišćenjem prekida izazvanog promenom stanja pinova na koje su povezani tasteri. Kod mikrokontrolera Atmega328P, postoje 3 prekida koja se generišu na ovaj način:

- PCINT0, izazvan promenom na nekom od pinova iz grupe PCINT7..PCINT0
- PCINT1, izazvan promenom na nekom od pinova iz grupe PCINT14..PCINT8
- PCINT2, izazvan promenom na nekom od pinova iz grupe PCINT23..PCINT16

Svi pinovi na koje su povezani tasteri (PC3..PC0) pripadaju grupi koja izaziva prekid PCINT1. Pošto se u aplikaciji koja realizuje elektronsku kocku koriste samo tasteri LEVO (PC0, odnosno PCINT8) i DESNO (PC2, odnosno PCINT10), u okviru registra PCMSK1 potrebno je setovati samo bite na pozicijama 0 i 2. Na taj način, prekid će biti izazvan samo pritiskom na neki od ova dva tastera, dok će ostali biti ignorisani.

Taster	Pin	PCINT ulaz
S0(LEVO)	PC0	PCINT8
S1(DOLE)	PC1	PCINT9
S2(DESNO)	PC2	PCINT10
S3(GORE)	PC3	PCINT11

```c
#include <avr/io.h>
#include <avr/interrupt.h>

const unsigned char cifre[] = {
    0xdd, 0x46, 0x54, 0x9c, 0x34, 0x24
}; // look-up tabela sa simbolima

unsigned char kockica = 0, brojac = 0, ukljuceno = 1;

ISR(TIMER0_COMPA_vect)
{
    // prekid tajmera 0 usled dostizanja vrednosti registra OCR0A
    if (!ukljuceno)
        brojac = 0;

    if (++brojac == 40)
    {
        brojac = 0;
        // promena stanja kockice desava se na svakih 40ms
        if (++kockica == 7)
            kockica = 1;

        PORTD = cifre[kockica - 1];
    }
}

ISR(PCINT1_vect)
{
    // prekid usled promene stanja pina PCINT10 ili pina PCINT8
    switch ((~PINC) & 0x05)
    {
        case 0x01: // pritisnut levi taster
            ukljuceno = 1;
            break;
        case 0x04: // pritisnut desni taster
            ukljuceno = 0;
            break;
    }
}

int main()
{
    // inicijalizacija portova:
    DDRB = 0x01;    // PB0 -> izlaz
    DDRC = 0x00;    // port C -> izlaz
    DDRD = 0xff;    // port D -> izlaz
    PORTB = ~0x01;  // ukljucenje displeja D4

    PCICR = (1 << PCIE1); // dozvola prekida usled promene stanja
    PCMSK1 = 0x05;        // pina PCINT10, ili pina PCINT8

    // inicijalizacija tajmera 0:
    TCCR0A = 0x02;  // tajmer 0: CTC mod
    TCCR0B = 0x03;  // tajmer 0: fclk = fosc/64
    OCR0A = 249;    // prekid tajmera 0: 250 Tclk (OCR0A + 1 = 250)
    TIMSK0 = 0x02;  // dozvola prekida tajmera 0 usled dostizanja
                    // vrednosti registra OCR0A
```

```
    sei();   // I = 1 (dozvola prekida)

    while(1);  // glavni program se vrti u praznoj petlji i ceka prekide

    return 0;
}
```

Zadatak za vežbu:

Napisati program koji na 7SEG displeju ispisuje decimalni brojač, čije početno stanje je 0. Brojač se inkrementira pritiskom na taster GORE, a dekrementira pritiskom na taster DOLE. U slučajevima kad je stanje brojača negativno, omogućiti njegov prikaz dodavanjem predznaka „-“ ispred prve značajne cifre.