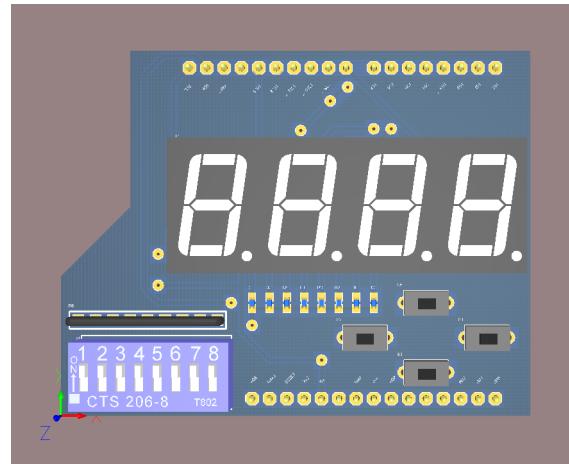


Poglavlje 18

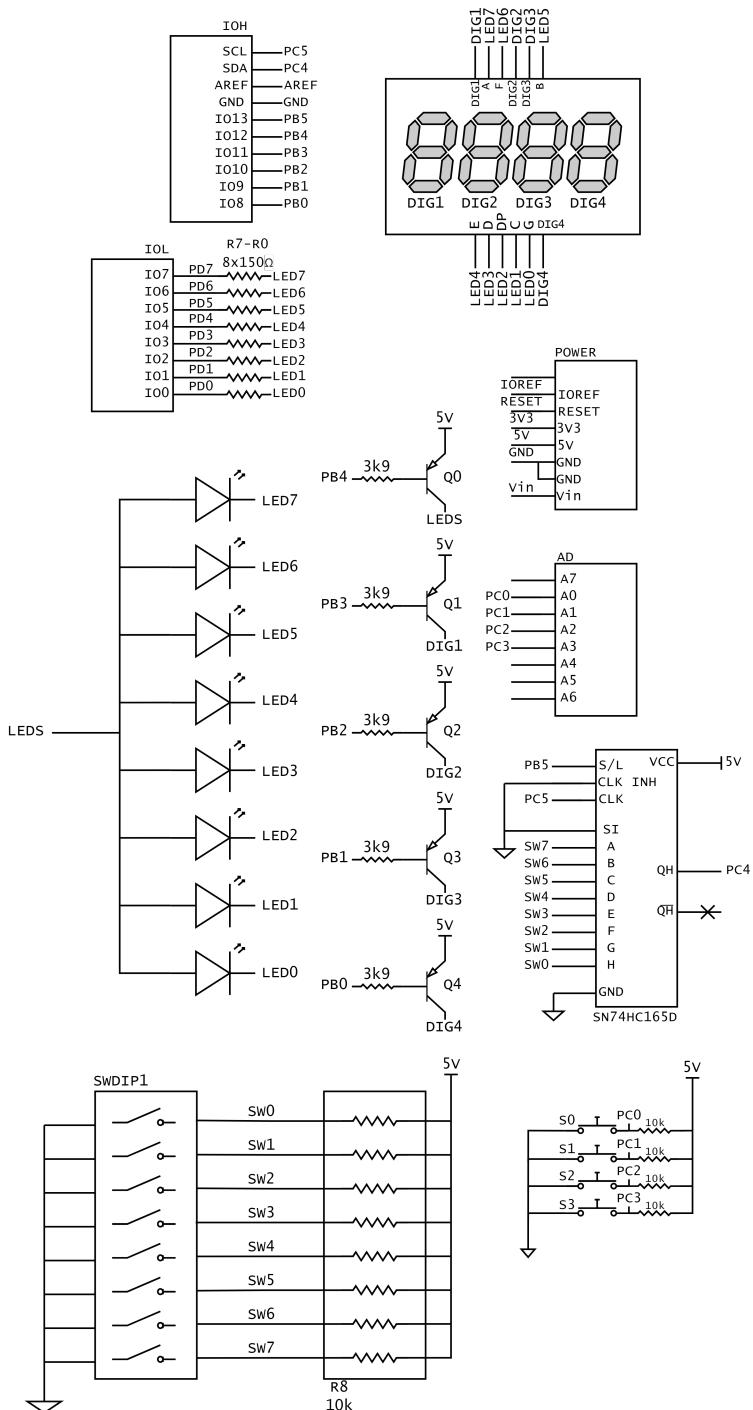
PLS7 ekspanziona ploča

18.1 Shema PLS7 ekspanzione ploče

Na slici 18.1 dat je fizički izgled PLS7 ekspanzione ploče, dok je shematski prikaz ekspanzione ploče dat na slici 18.2.



Slika 18.1: Izgled PLS7 ekspanzione ploče



Slika 18.2: Shema PLS7 ekspanzije ploče

18.2 LED diode

Niz od osam LED dioda (LED7...LED0) povezan je u spoju sa zajedničkom anodom. Spoj između napona napajanja ($V_{cc} = 5V$) i čvora na koji su vezane anode ostvaruje se preko prekidačkog tranzistora Q0. Tranzistorom Q0 se upravlja pomodu pina PB4. Pošto je u pitanju PNP tranzistor čiji emiter je spojen na napon napajanja, njegovo uključivanje se vrši dovođenjem niskog napona na bazu (odnosno kada je PB4=0), a kada je napon na bazi visok (PB4=1), Q0 de biti isključen, čime se istovremeno prekida proticanje struje kroz ceo niz LED dioda.

Katode su preko otpornika R7...R0 vrednosti 150Ω povezane sa pinovima porta D mikrokontrolera ATmega328P (PD7...PD0). Na ovaj način moguće je kontrolisati stanje svake diode pojedinačno. Za uključenje diode na poziciji n , pored toga što tranzistor Q0 mora biti uključen, potrebno je još i postaviti pin PDn na nizak logički nivo (PDn=0). U suprotnom, kada je PDn=1, na katodi se pojavljuje visok napon, čime se dioda isključuje.

```
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>

int16_t main(void)
{
    DDRD = 0xff; //port D -> izlaz
    DDRB |= 1 << 4; //PB4 -> izlaz
    PORTB &= ~(1 << 4); //PB4 = 0, cime se ukljucuje
                         tranzistor Q0

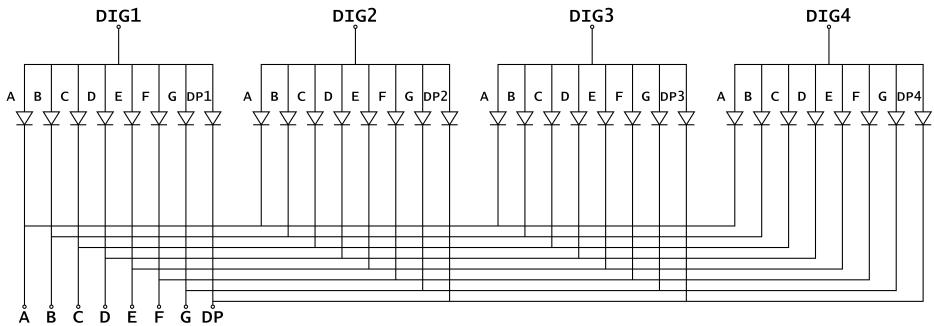
    while(1)
    {
        PORTD = 0xaa; //uključuju se LED na pozicijama 6, 4,
                      2 i 0
        _delay_ms(1000); //pauza 1s
        PORTD = 0x55; //uključuju se LED na pozicijama 7, 5,
                      3 i 1
        _delay_ms(1000); //pauza 1s
    }
    return 0;
}
```

Listing 18.1: Primer rada sa diodama na PLS7 ekspanzionoj ploči

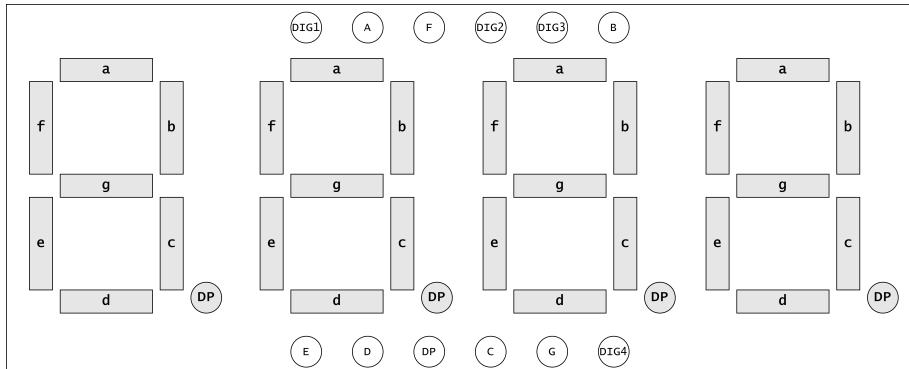
18.3 LED displej 4x7SEG

Na ploči se nalazi komponenta U1 koja sadrži četiri sedmo-segmentna displeja sa decimalnom tačkom, integrisana u zajedničko kućište. LED diode koje čine segmente displeja povezane su u spoju sa zajedničkom anodom. Katode odgovarajućih

segmenata kod sva 4 displeja kratko su spojene, kao što je prikazano na internoj shemi displeja, na slici 18.3. Takođe, na slici 18.4 prikazano je i kućište za četiri sedmo-segmentna displeja na PLS7 ekspanzionoj ploči.



Slika 18.3: Interna shema sedmo-segmentnog displeja



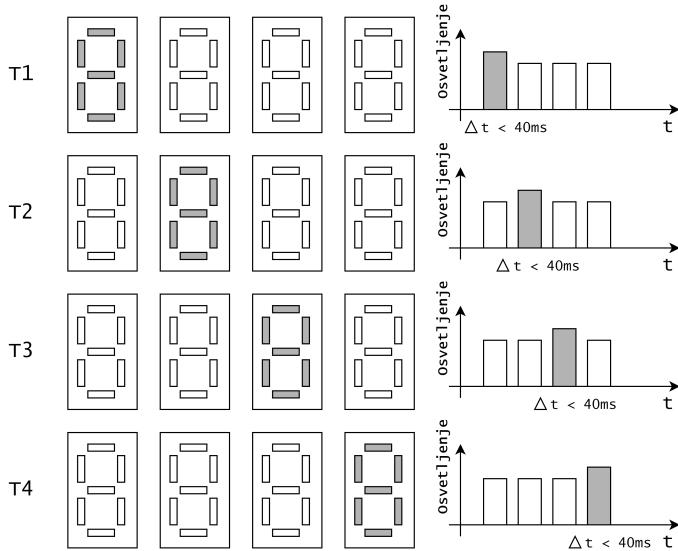
Slika 18.4: Kućište sedmo-segmentnog displeja

Na različitim displejima (DIG1–DIG4), katode odgovarajućih segmenata (A–DPx) su kratko spojene, a anode su razdvojene. Na red sa anodama displeja DIG1...DIG4 povezani su prekidački tranzistori Q1...Q4 pomoću kojih je moguće prekidati struju i time uključivati, odnosno isključivati, pojedine displeje. Anode su preko otpornika za ograničavanje struje R7...R0 povezane sa pinovima porta D mikrokontrolera (PD7...PD0).

Logika upravljanja displejima realizuje se tzv. vremenskim multipleksiranjem, kod kog se displej osvežava u pravilnim vremenskim intervalima. Perioda osvežavanja displeja T podeljena je na četiri intervala jednakog trajanja: $T_1 = T_2 = T_3 = T_4 = T/4$. Na početku intervala T_1 , uključuje se tranzistor Q1 i isključuju tranzistori Q2...Q4. Tada stanja pinova porta D (PD7...PD0) određuju stanja pojedinih segmenata displeja DIG1¹. Nakon toga, tokom intervala T_2 uključuje se Q2, ostali

¹Kao i kod niza LED dioda, pojedini segmenti displeja DIGn uključuju se postavljanjem

tranzistori se isključuju, a pinovi porta D kontrolisu segmente displeja DIG2, i tako redom do poslednjeg displeja DIG4. Frekvencija osvežavanja treba da bude dovoljno velika (bar 25Hz), kako bi korisnik imao iluziju da je prikaz na displeju nepomičan. Tranzistori Q1...Q4 kontrolisu se pinovima porta B (PB4...PB0)². Ideja vremenskog multipleksiranja je ilustrovana na sledećoj slici.



Slika 18.5: Vremensko multipleksiranje

U narednim tabelama prikazana je veza između stanja pinova na portovima B i D i segmenata na displeju, odnosno LED dioda.

Tabela 18.1: Veza između periferija i pinova porta B

Displej	PB4	PB3	PB2	PB1	PB0
LEDS	0	1	1	1	1
DIG1	1	0	1	1	1
DIG2	1	1	0	1	1
DIG3	1	1	1	0	1
DIG4	1	1	1	1	0

Tabela 18.2: Veza između periferija i pinova porta D

pinova porta D na logičku nulu ($PDx = 0$), pod uslovom da je odgovarajući prekidački tranzistor Qn uključen.

²Poput tranzistora Q0 i tranzistori Q1...Q4 su PNP sa emiterom direktno spojenim na Vcc . Uključivanje se vrši dovođenjem niskog napona na bazu, odnosno kada je odgovarajući pin $PBn = 0$.

Displej	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
LEDS	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
DIG1	A1	F1	B1	E1	D1	DP1	C1	G1
DIG2	A2	F2	B2	E2	D2	DP2	C2	G2
DIG3	A3	F3	B3	E3	D3	DP3	C3	G3
DIG4	A4	F4	B4	E4	D4	DP4	C4	G4

```
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>

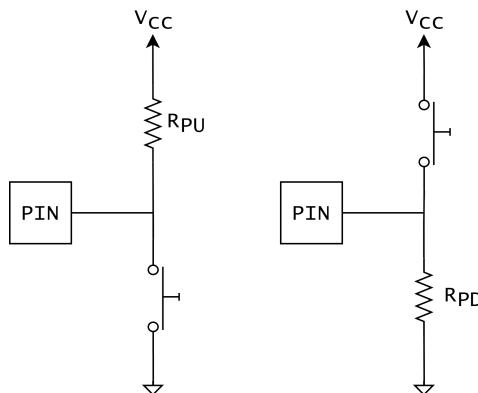
const uint8_t symbols[] = {
    0x0c, 0xa4, 0x27, 0xc4
}; //look-up tabela sa simbolima za ispis na displej (A,b,C,d)

int16_t main(void)
{
    uint8_t display;
    DDRD = 0xff; //port D -> izlaz
    DDRB = 0x0f; //PB3 - PB0 -> izlazi
    while(1)
    {
        for (display = 1; display <= 4; display++)
        {
            PORTB = ~(0x01 << (4-display)); //uključuje se
            //tranzisto na odgovarajucoj poziciji
            PORTD = symbols[display-1]; //ispis simbola iz
            //tabele
            _delay_ms(2); //pauza 2ms
        }
    }
    return 0;
}
```

Listing 18.2: Primer rada sa sedmo-segmentnim na PLS7 ekspanzionoj ploči

18.4 Tasteri

Prilikom povezivanja tastera sa pinovima mikrokontrolera, postoje dva standardna načina povezivanja prikazana na slici 18.6.



Slika 18.6: Standardni načini povezivanja tastera sa pinovima mikrokontrolera

Prvi način (konfiguracija levo) podrazumeva povezivanje tastera između pina i mase, u kombinaciji sa *pull-up* otpornikom. Uloga otpornika jeste uspostavljanje podrazumevane vrednosti na pinu u situaciji kada tastera nije pritisnut (stanje logičke jedinice). Prilikom pritiska tastera, zatvara se strujni krug od napona napajanja, preko otpornika, ka masi i logičko stanje na pinu će biti 0.

Kod drugog načina (konfiguracija desno) taster je smešten između pina i napajanja, u kombinaciji sa *pull-down* otpornikom. U ovoj situaciji, logika je obrnuta. Kada je taster pušten, tada je, zahvaljujući otporniku, stanje pina logička 0. Prilikom pritiska tastera, pin je kratko spojen sa naponom napajanja i očitava se logička 1.

Iako je drugi način povezivanja u skladu sa pozitivnom logikom (taster pritisnut – logička 1, taster pušten – logička 0), u praksi se češće koristi prvi način. Naime, prva konfiguracija u potpunosti kompatibilna sa *open-collector* logikom, pa u toj situaciji, čak nije ni neophodan *pull-up* otpornik. Takođe, kod ATmega328P kontrolera postoji interni *pull-up* otpornik, što dodatno olakšava povezivanje tastera, budući da nije potrebno koristiti eksterni otpornik. Treba primetiti da se i u slučaju PLS7 ekspanzione ploče koristi prvi pristup.

Tasteri S3-S0 povezani su direktno na pinove porta C (PC3-PC0). Svaki od tastera povezan je između ulaznog pina i mase, u kombinaciji sa *pull-up* otpornikom. To znači da kada taster nije pritisnut, stanje pina će biti 1 usled prisustva *pull-up* otpornika, a u pritisnutom stanju taster kratko spaja pin na masu, što se čita kao logička 0.

```
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>

const uint8_t symbols[] = {0x0c, 0xa4, 0x27, 0xc4};

void write7SEG (uint8_t value, uint8_t position)
{
```

```

//ukljucenje displeja na zeljenoj poziciji
PORTB = ~(0x01 << (4 - position));
//prikaz karaktera
PORTD = value;
//pauza 2ms
_delay_ms(2);

}

int16_t main(void)
{
    uint8_t buttons;
    DDRD = 0xff; //port D -> izlaz
    DDRC = 0x00; //port C -> ulaz

    DDRB = 0x0f; //PB3 - PBO izlazi

    while (1)
    {
        buttons = PINC & 0x0f; //ocitavanje stanja tastera

        if (!(buttons & 0x01)) //provera stanja tastera S0
            write7SEG (symbols[0], 1);
        else
            write7SEG (0xfe, 1);

        if (!(buttons & 0x02)) //provera stanja tastera S1
            write7SEG (symbols[1], 2);
        else
            write7SEG (0xfe, 2);

        if (!(buttons & 0x04)) //provera stanja tastera S2
            write7SEG (symbols[2], 3);
        else
            write7SEG (0xfe, 3);

        if (!(buttons & 0x08)) //provera stanja tastera S3
            write7SEG (symbols[3], 4);
        else
            write7SEG (0xfe, 4);
    }
    return 0;
}

```

Listing 18.3: Primer rada sa tasterima na PLS7 ekspanzionoj ploči

Svi pinovi na koje su povezani tasteri (PC0...PC3) pripadaju grupi pinova koja izaziva PCINT1 prekid prema tabeli 18.3.

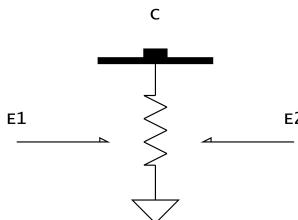
Tabela 18.3: Način vezivanja tastera

Taster	Pin	PCINT ulaz
Levo (S0)	PC0	PCINT8
Dole (S1)	PC1	PCINT9
Desno (S2)	PC2	PCINT10
Gore (S3)	PC3	PCINT11

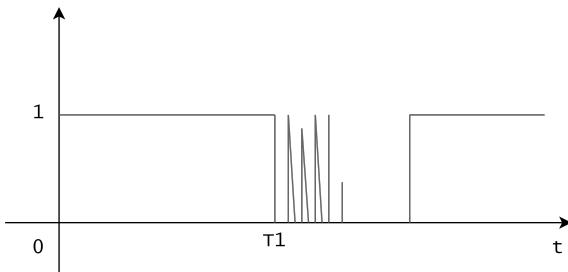
U skladu sa datom tabelom, kako bi rad sa prekidom usled promene stanja na pinu bio moguć, potrebno je na odgovarajući način konfigurisati PCICR i PCMSK1 registre. Objasnjenje ovih registara je dato u Poglavlju 3. Naravno, pored toga, potrebno je napisati i odgovarajuću prekidnu rutinu koja vrši obradu prekida.

18.4.1 Problem *bouncing-a* i tehnike za njegovo rešavanje

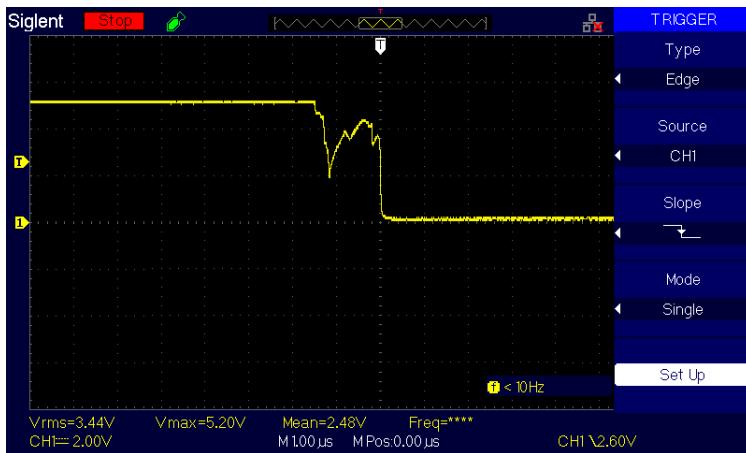
Usled mehaničke konstrukcije tastera prikazane na slici 18.7, prilikom njegove upotrebe dolazi do pojave fenomena koji se naziva *bouncing* ili "podrhtavanje". Naime, prilikom pritiska tastera, ostvaruje se mehanički (a potom i električni) kontakt između dve elektrode $E1$ i $E2$ i centralnog elementa C . Centralni element predstavlja elastičnu oprugu koja, u trenucima kada taster nije pritisnut, drži taster u otvorenom stanju. Zahvaljujući ovoj elastičnoj opruzi, prilikom pritiska tastera, taj kontakt nije trenutan, već postoji tzv. "podrhtavanje" tastera tokom kratkog vremenskog intervala ($\sim 10\text{ms}$). Ova situacija je ilustrovana na slici 18.8. Kao što se može primetiti, u trenutku pritiska tastera (trenutak $T1$), prelazak sa logičke 1 na logičku 0 nije čist, budući da postoje neželjene oscilacije u prelaznoj oblasti.



Slika 18.7: Mehanička konstrukcija tastera



Slika 18.8: Problem "podrhtavanja"

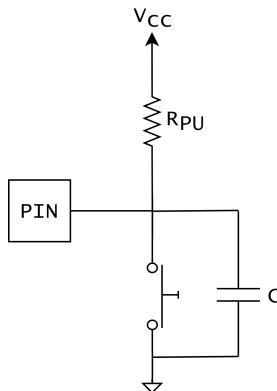


Slika 18.9: Problem "podrhtavanja" uhvaćen na osciloskopu

U većini aplikacija ovo je nepoželjna pojava koja može dovesti do neispravnog očitavanja tastera. Kako bi se ovo prevazišlo koriste se odgovarajuće hardverske i softverske tehnike. Ove tehnike su u engleskoj terminologiji poznate i kao *debouncing*.

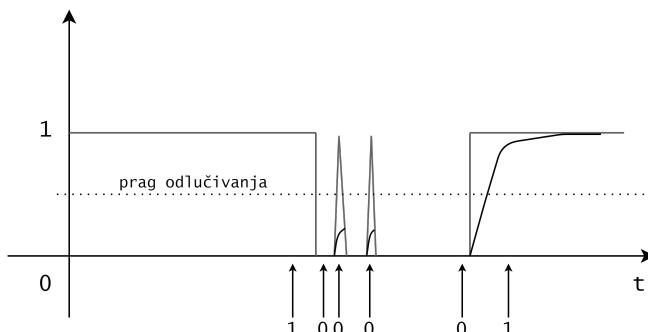
Hardverski debouncing

Hardverska tehnika za rešavanje problema "podrhtavanja" tastera podrazumeva upotrebu kondenzatora između pina i mase, kao na slici 18.10.



Slika 18.10: Hardverska tehnika rešavanja problema "podrhtavanja"

U ustaljenom stanju, odnosno kada je taster pušten, kondenzator je napunjena na vrednost napona napajanja. Prilikom pritiska tastera, kondenzator počinje da se prazni. Kada je taster ponovo pušten, kondenzator počinje da se puni u skladu sa vrednošću vremenske konstante $\tau = C \cdot R_{PU}$. Ovo je ilustrovano na slici 18.11.



Slika 18.11: Izgled vremenske ose prilikom pritiska tastera

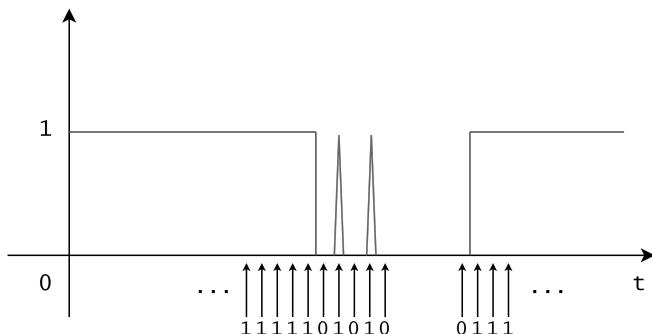
Ukoliko je vremenska konstanta dovoljno velika, amplituda neželjenih impulsa u ovoj situaciji će biti značajno manja od praga odlučivanja logičkog kola i time je problem *debouncing-a* rešen.

Neželjeni efekat jeste taj da, kada korisnik pritisne taster, potrebno je više vremena da signal na pinu dostigne vrednost logičke jedinice (usled punjenja kondenzatora).

Softverski *debouncing*

Posmatrano iz softverske perspektive, očitavanje tastera se uglavnom vrši u okviru neke petlje. Frekvencija odabiranja (broj prolaza kroz petlju) određena je frekven-

cijom na kojoj mikrokontroler radi. Budući da je ta frekvencija u većini situacija prilično velika, dešava se situacija u kojoj se vrši odabiranje parazitnih impulsa nastalih usled "podrhtavanja" tastera. Ovo je ilustrovano na slici 18.12.



Slika 18.12: Izgled vremenske ose prilikom pritiska tastera uz odabiranje

Jedan način da se ovo prevaziđe jeste skaliranje frekvencije odabiranja, čime će se verovatnoća odabiranja parazitnih impulsa značajno smanjiti. Međutim, u ovoj situaciji, vreme reakcije prilikom pritiska tastera je takođe smanjeno.

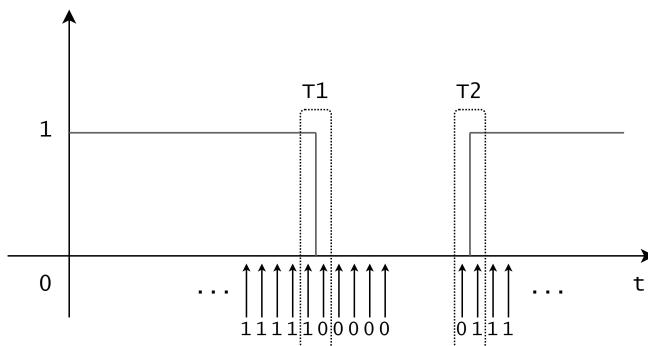
Drugi način, koji se i češće koristi u praksi podrazumeva da se, nakon detekcije pritiska tastera, generiše se odgovarajuće kašnjenje ($\sim 10\text{ms}$). Nakon isteka pomenuog intervala, potrebno je proveriti da li je očitano stanja isto kao i prilikom prvog očitavanja. Ukoliko jeste, možemo da tvrdimo da se pritisak tastera dogodio, odnosno da se period "podrhtavanja" završio. U suprotnom, postupak se ponavlja. Na ovaj način, problem *debouncing-a* je rešen, pri čemu je zadržana brzina odziva, za razliku od prvog načina.

18.4.2 Softversko "diferenciranje"

U praksi se često javlja potreba da se za *jedan* pritisak tastera izvrši *jedna* akcija (na primer, ispis neke vrednosti). Kao što je već ranije bilo reči, očitavanje tastera se u većini situacija vrši u okviru neke petlje sa određenom frekvencijom odabiranja, koja je po pravilu velika. Ovo znači da se u vremenskom intervalu tokom kog je taster pritisnut izvrši veliki broj odabiranja, odnosno da se petlja izvrši neodređeni broj puta. U nekim aplikacijama, ovo stvara izvesne probleme. Naime, ukoliko je pritisak tastera uslov izvršavanja neke akcije, data akcija će se izvršavati sve dok je taster pritisnut, što može biti zadovoljeno tokom više iteracija petlje, uzimajući u obzir vezu između vremenskog intervala tokom kog je taster pritisnut i broja prolazaka kroz petlju. Ovo će dalje uzrokovati izvršavanjem date akcije neodređeni (veći) broj puta. Ukoliko je zahtev bio da se za jedan pritisak taster izvrši jedna akcija, iz pomenuog razloga će biti narušen. Ovo je moguće rešiti tehnikom *softverskog diferenciranja*.

Osnovna ideja ove tehnike jeste ta, da se detektuje tranzicija prilikom pritiska tastera. Ovo podrazumeva pamćenje prethodnog stanja očitanog tastera i postavljanje logičkog uslova koji proverava da li je prethodno stanje tastera različito od trenutnog stanja tastera. Ukoliko jeste, dogodila se tranzicija. Ukoliko nije, tranzicije nije bilo. Razlog zašto se ova tehnika naziva softversko "diferenciranje" jeste ta, što sam postupak u osnovi podseća na diferenciranje pravougaonog signala (analogija sa vremenskim intervalom tokom kog je taster pritisnut), čiji je rezultat *Dirakov impuls* (analogija sa detekcijom tranzicije između dva stanja tastera)

Na slici 18.13 su ilustrovane dve situacije (vremenski intervali $T1$ i $T2$) u kojima je pomenuti uslov zadovoljen.



Slika 18.13: Detekcija tranzicije između dva stanja tastera

U skladu sa tim, implementacija tehnike softverskog diferenciranja je data na listingu 18.4.

```
...
uint8_t last_state;
uint8_t current_state;
...
while(1)
{
    current_state = readButtons();

    /*detekcija tranzicije*/
    if (last_state != current_state)
    {
        last_state = current_state;
        /*akcija*/
    }
}
```

Listing 18.4: Implementacija tehnike softverskog diferenciranja

18.5 Prekidači

Komponenta SWDIP1, koja sadrži 8 prekidača u zajedničkom kućištu, nije direktno povezana na pinove mikrokontrolera, budući da bi za tako nešto bilo potrebno osam pinova, a povezivanjem već opisanih komponenti preostala su samo tri raspoloživa pina. Stoga, prekidači su povezani na 8-bitni pomerički registar sa mogućnošću paralelnog upisa *74HC165* (komponenta U2 na shemii 18.3). Pin mikrokontrolera PB5 povezan je na liniju za dozvolu upisa u registar (SH/LD) i upravlja njenim stanjem. Upis tekućeg stanja svih osam prekidača u registar vrši se postavljanjem ove linije na logičku 0 (PB5 = 0). Nakon toga, potrebno je ukinuti dozvolu upisa (postaviti PB5 = 1), čime se registar "zaključava" za dalji upis. Čitanje vrednosti registra obavlja se serijskim protokolom, na sledeći način. Promenom stanja pina PC5 mikrokontrolera generiše se takt signal (doveden na CLK ulaz) kojim se vrši pomeranje sadržaja registra. Serijski izlaz registra (QH izlaz) povezan je na pin PC4, preko koga se vrednost čita, bit-po-bit, tokom osam perioda takta. Vremenski dijagrami tokom jednog ciklusa učitavanja stanja prekidača u kolo 74HC165 i njihovog serijskog prenosa prikazani su na slici 18.14³. Na listingu 18.5 je dat primer programa koji radi sa prekidačima.

```
#include <avr/io.h>
#include <stdint.h>
#include <util/delay.h>

//makroi za kontrolu pinova preko kojih je kontroler povezan
//sa 74HC165:
#define SCL_HI (PORTC |= (1<<5))
#define SCL_LO (PORTC &= ~(1<<5))
#define SDA (PINC & (1 << 4))

#define SHLD_HI (PORTB |= (1<<5))
#define SHLD_LO (PORTB &= ~(1<<5))

uint8_t readSwitches()
{
    uint8_t i, tmp = 0, mask = 0x80;

    //impuls za upis stanja prekidaca u registar
    SHLD_HI;
    SHLD_LO;
    SHLD_HI;

    for (i = 0; i < 8; i++)
    {
        SCL_LO;
        SCL_HI; //generisanje aktivne ivice takta
        if (SDA) //provera stanja ulaznog pina
            tmp |= mask;
        mask = mask << 1;
    }
    return tmp;
}
```

³Linija CLK INH kojom se zabranjuje pomeranje sadržaja registra kratko je spojena na masu, čime je data stalna dozvola pomeranja.

```

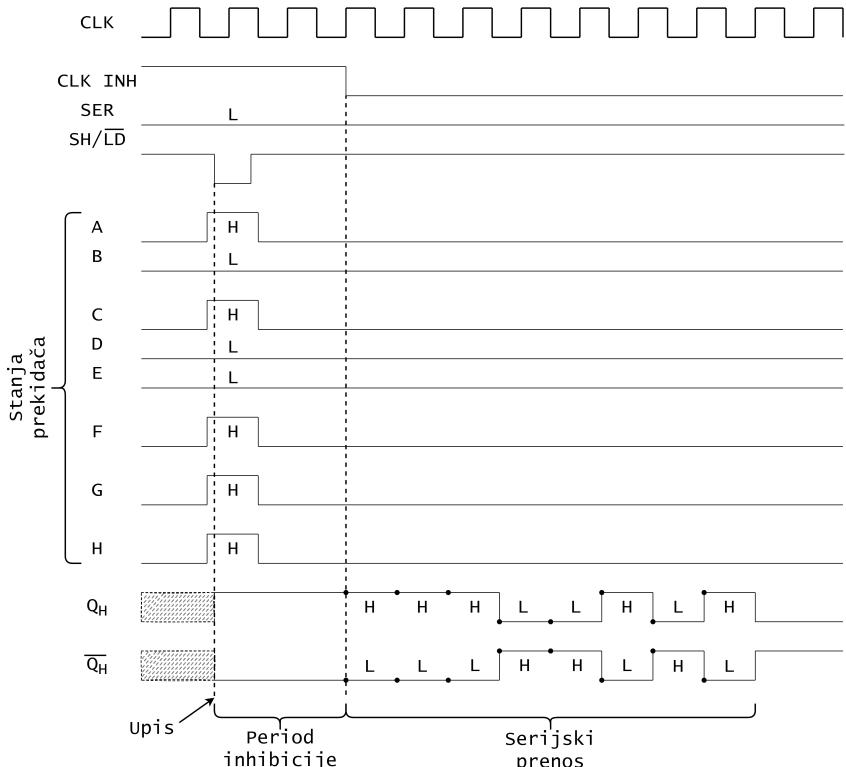
        tmp |= mask;
        mask >= 1;
    }
    return tmp;
}

int16_t main(void)
{
    DDRD = 0xff;      //port D -> izlaz
    DDRC = 0x20;      //PC5 -> izlaz
    DDRB = 0x30;      //PB5 i PB4 -> izlazi
    PORTB = ~0x10;    //uključenje tranzistora Q0

    while (1)
        PORTD = readSwitches();
    return 0;
}

```

Listing 18.5: Primer rada sa prekidačima na PLS7 ekspanzionoj ploči



Slika 18.14: Vremenski dijagrami tokom jednog ciklusa učitavanja stanja prekidača

18.6 Zadaci za vežbu I

Zadatak 18.6.1. Napisati program koji pomoću dioda na PLS7 ekspanzionoj ploči prikazuje elemente Fibonačijevog niza svake sekunde. Diode treba da prikažu binarnu vrednost trenutnog elementa. Kašnjenje realizovati:

- (a) upotrebom biblioteke `utils/delay.h`
- (b) upotrebom prekida usled isteka tajmer/brojač modula 0; po potrebi, koristiti gotovu biblioteku `timer0`.

Zadatak 18.6.2. Napisati program koji na slučajan način postavlja nasumično izabrani broj iz opsega 0-9 na nasumično odabranu cifru sedmo-segmentnog displeja, svake sekunde.

- (a) upotrebom biblioteke `utils/delay.h`
- (b) upotrebom prekida usled isteka tajmer/brojač modula 0; po potrebi, koristiti gotovu biblioteku `timer0`.

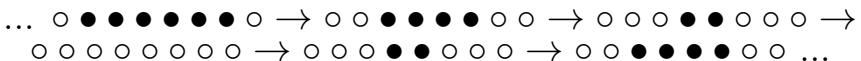
Zadatak 18.6.3. Napisati program koji pomoću dioda na PLS7 ekspanzionoj ploči implementira sledeće funkcionalnosti:

- (a) Grupa od četiri uključene diode se pomera naizmenično u smeru levo-desno. Nakon što grupa uključenih dioda dođe do poslednje pozicije - smer pomeranja se obrće. Prelazi između dva stanja se dešavaju periodično, na svakih pola sekunde.



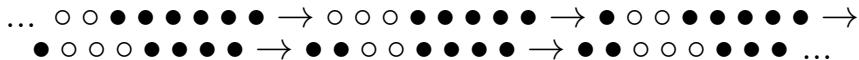
○ – dioda je uključena ● – dioda je isključena

- (b) Po jedna dioda se istovremeno uključuje sa obe strane, sve dok čitav niz dioda ne bude uključen. Nakon toga, diode se isključuju po istom principu, u suprotnom smeru, počevši od centra. Prelazi između dva stanja se dešavaju periodično, na svakih pola sekunde.



○ – dioda je uključena ● – dioda je isključena

- (c) Pomeranje grupe dioda sa leve na desnu stranu koje podseća na "kretanje" gusenice. Prelazi između dva stanja se dešavaju periodično, na svakih pola sekunde.



○ – dioda je uključena ● – dioda je isključena

Za precizno generisanje kašnjenja koristiti prekide usled isteka tajmer/brojač modula 0. Po potrebi, koristiti gotovu biblioteku **timer0**.

Zadatak 18.6.4. Napisati program koji implementira jednostavan brojač. Stanje brojača se povećava za jedan svake sekunde i ispisuje na sedmo-segmentnom displeju. Početno stanje brojača je *0000*, a krajnje *9999*. Nakon dostizanja krajnjeg stanja, potrebno je krenuti brojanje od početka. Kašnjenje realizovati:

- (a) upotreborom biblioteke `utils/delay.h`
- (b) upotreborom prekida usled isteka tajmer/brojač modula 0; po potrebi, koristiti gotovu biblioteku **timer0**.

Zadatak 18.6.5. Napisati program koji omogućava pomeranje jedne diode na PLS7 ekspanzionoj ploči levo/desno u zavisnosti od pritiska levog/desnog tastera. Očitavanje tastera realizovati:

- (a) unutar glavne petlje
- (b) upotreborom prekida usled promene stanja na ulaznom pinu.

Zadatak 18.6.6. Napisati program koji pomoću tastera levo/desno bira cifru sedmosegmentnog displeja na PLS7 ekspanzionoj ploči. Omogućiti da se tasterima gore/dole menja vrednost cifre u opsegu 0-9. Očitavanje tastera realizovati:

- (a) unutar glavne petlje
- (b) upotreborom prekida usled promene stanja na ulaznom pinu.

Zadatak 18.6.7. Napisati program koji na sedmo-segmentnom displeju ispisuje decimalni brojač. Početno stanje brojača je *000*, najveće *999*, a najmanje *-999*. Trenutna vrednost brojača se povećava pritiskom na taster gore, a smanjuje pritiskom na taster dole. U slučajevima kada je stanje brojača negativno, omogućiti njegov prikaz dodavanjem predznaka *"+"* ispred prve značajne cifre. Očitavanje tastera realizovati:

- (a) unutar glavne petlje
- (b) upotreborom prekida usled promene stanja na ulaznom pinu.

Zadatak 18.6.8. Napisati program koji implementira igru sa elektronском kockicom. Na početku igre, korisnik pomoću tastera gore/dole bira broj bacanja kockice, pri čemu je moguće izabrati maksimalno četiri bacanja. Nakon izbora broja bacanja, pritiskom na taster desno se prelazi na bacanje kockice koje je potrebno

realizovati na sledeći način. Na displeju D4 (krajnji desni sedmo-segmentni displej) se smenjuju brojevi iz intervala od 1 do 6, sa početnom frekvencijom od 40Hz. Pritiskom na taster levo, frekvencija obrtanja se smanjuje po narednoj formuli, sve do zaustavljanja:

$$f_{nova} = \frac{f_{stara}}{1.2}$$

Smatrali da je kockica zaustavljena kada frekvencija dostigne vrednost manju od 0.1Hz. Nakon toga, u zavisnosti od broja bacanja koji je korisnik prvobitno izabrao, postupak se ponavlja na ostalim sedmo-segmentnim displejima. Kada je izvršeno poslednje bacanje, vrednosti svih kockica se sabiraju i ukupna suma se prikazuje na sedmo-segmentnom displeju. Ukoliko je suma veća od polovine maksimalne moguće vrednosti koju korisnik može da ostvari, potrebno je uključiti svih osam dioda na PLS7 ploči. U suprotnom, potrebno je uključiti svaku drugu diodu. Maksimalna suma koju korisnik može da ostvari se računa na sledeći način:

$$S_{max} = broj_bacanja \cdot 6$$

Očitavanje tastera realizovati:

- (a) unutar glavne petlje
- (b) upotrebom prekida usled promene stanja na ulaznom pinu.

Prikaz vrednosti na sedmo-segmentnom displeju realizovati:

- (a) unutar glavne petlje
- (b) upotrebom prekida usled isteka tajmer/brojač modula 0.

Zadatak 18.6.9. Napisati program koji pomoću tastera bira sedmo-segmentni displej na PLS7 ekspanzionoj ploči na kom treba da se prikazuje vrednost učitana putem prekidača. Pritiskom na taster se selektuje displej, a nakon puštanja tastera, na selektovanom displeju se prikazuje vrednost. Očitavanje tastera realizovati:

- (a) unutar glavne petlje
- (b) upotrebom prekida usled promene stanja na ulaznom pinu.

Zadatak 18.6.10. Napisati program koji implementira konvertor binarnog broja, koji se zadaje putem prekidača, u decimalni broj koji se prikazuje na sedmo-segmentnom displeju.

18.7 Biblioteka za rad sa PLS7 ekspanzionom pločom

U okviru ovih vežbi biće objašnjena implementacija biblioteke za rad sa PLS7 ekspanzionom pločom. Podsećanja radi, u pitanju je ploča koja sadrži sledeće periferije:

- niz od 4 sedmosegmentna displeja (D1 – D4);
- niz od 8 LED dioda (LD7 – LD0);
- 4 tastera (UP, DOWN, LEFT, RIGHT);
- niz od 8 prekidača (S1 – S8).

Za početak, biće definisane osnovne funkcionalnosti koje se očekuju od rutina za upravljanje periferijama, u okviru biblioteke **PLS7**. Biblioteka će sadržati dve izvorne datoteke: *PLS7.h* i *PLS7.c*.

18.7.1 LED diode i LED displej 4x7SEG

Niz od osam plavih LED dioda, kao i četiri sedmosegmentna LED displeja u zajedničkom kućištu su izlazne periferije, koji dele zajedničko svojstvo. U pitanju je pet nizova po osam dioda u spoju sa zajedničkom anodom. Njima se upravlja pomoću pinova porta D (PD7...PD0), tehnikom vremenskog multipleksiranja. Ovaj način upravljanja moguć je zahvaljujući tranzistorima Q4...Q0 koji im po potrebi obezbeđuju ili prekidaju struju, a upravljanje tranzistorima se vrši pomoću pinova porta B (PB4...PB0). Kao što je već videno, periodično multipleksno osvežavanje stanja LED dioda i segmenata sedmo-segmentnog displeja je zadatak koji je zgodno prepustiti tajmerskoj prekidnoj rutini, sa odgovarajućim trajanjem periode (tipično *1ms*), pa će stoga implementacija sadržati upravo takvu prekidnu rutinu. U nastavku, na listingu 18.6, je dat predlog makro konstanti i prototipova funkcija za upravljanje izlaznim periferijama.

```
#define LEDS      0
#define D1        1
#define D2        2
#define D3        3
#define D4        4

void pls7WriteDisplay(uint8_t display, uint8_t value);
uint8_t pls7ReadDisplay(uint8_t display);
```

Listing 18.6: Makro konstante i prototipovi funkcija za rad sa izlaznim periferijama

U implementaciji je potrebno za svaki displej kreirati zaseban bafer koji sadrži osobitnu vrednost koja definiše stanja pojedinih segmenata. Upis, odnosno čitanje vrednosti ovih bafera vrši se pomoću funkcija *pls7WriteDisplay()* i *pls7ReadDisplay()*.

Kod obe funkcije, kao parametar `display`, prosleđuje se makro LEDS, D1, D2, D3 ili D4, u zavisnosti od toga koji niz LED dioda se adresira. Parametar `value` u pozivu funkcije `pls7WriteDisplay()` određuje stanja pojedinih dioda, odnosno segmenata na displeju, kao što je prikazano u tabeli 18.4.

Tabela 18.4: Vrednosti parametara funkcije `pls7WriteDisplay()`

display		value							
Makro	Vrednost	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LEDS	0	LD7	LD6	LD5	L4	LD3	LD2	LD1	LD0
D1	1	A1	F1	B1	E1	D1	DP1	C1	G1
D2	2	A2	F2	B2	E2	D2	DP2	C2	G2
D3	3	A3	F3	B3	E3	D3	DP3	C3	G3
D4	4	A4	F4	B4	E4	D4	DP4	C4	G4

Na listingu 18.7 je dat primer korišćenja `pls7WriteDisplay()` funkcije.

```
#include "PLS7.h"

const uint8_t simboli[] PROGMEM = {
    0xf3, 0x5b, 0xd8, 0x3b
}; //look-up tabela sa simbolima za ispis na displej (A,b,C,d)

void main()
{
    pls7Init();

    //D1 <- 'A'
    pls7WriteDisplay(D1, pgm_read_byte(&simboli[0]));
    //D2 <- 'b'
    pls7WriteDisplay(D2, pgm_read_byte(&simboli[1]));
    //D3 <- 'C'
    pls7WriteDisplay(D3, pgm_read_byte(&simboli[2]));
    //D4 <- 'd'
    pls7WriteDisplay(D4, pgm_read_byte(&simboli[3]));

    while(1);
}
```

Listing 18.7: Primer korišćenja `pls7WriteDisplay()` funkcije

18.7.2 Tasteri

Tasteri LEFT, RIGHT, UP i DOWN povezani su direktno na pinove porta C (PC3...PC0). Predlog makro konstanti i prototipa funkcije za očitavanje njihovih stanja je dat na listingu 18.8:

```
#define LEFT      0
```

```
#define DOWN      1
#define RIGHT     2
#define UP        3

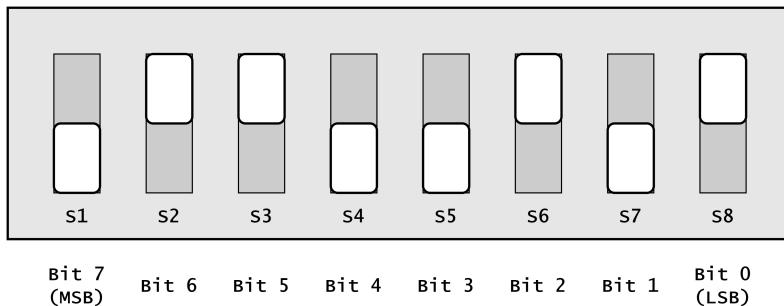
uint8_t pls7ButtonState(uint8_t button);
```

Listing 18.8: Makro konstante i prototip funkcije za rad sa tasterima

Kao parametar `button` prosleđuje se odgovarajuću makro konstantu (`LEFT`, `RIGHT`, `UP` ili `DOWN`), u zavisnosti od toga koji se taster očitava. Povratna vrednost može biti 1 ukoliko je taster pritisnut u trenutku poziva funkcije, odnosno 0 ako je taster tada bio pušten.

18.7.3 Prekidači

Komponenta koja sadrži osam prekidača u zajedničkom kućištu nije direktno povezana na pinove mikrokontrolera, pošto bi za tako nešto bilo potrebno osam pinova, a povezivanjem već opisanih komponenti preostala su samo tri raspoloživa pina. Stoga su prekidači povezani na 8-bitni pomerački registar sa mogućnošću paralelnog upisa. Komunikacija između kontrolera i pomeračkog registra obavlja se putem serijskog protokola, koji podrazumeva prvo paralelni upis stanja prekidača u registar, a potom čitanje sadržaja registra bit po bit, tokom 8 uzastopnih perioda takta.



Slika 18.15: Fizički izgled kućišta komponente sa prekidačima

Na slici 18.15 je prikazan fizički izgled kućišta komponente koja sadrži niz prekidača. Prekidači koji su postavljeni u donji položaj predstavljaju bite koji su u stanju logičke nule, a oni koji su u gornjem položaju su u stanju logičke jedinice. Na listingu 18.9 je dat predlog makro konstanti i prototipovi funkcija za rad sa prekidačima.

```
#define S1    7
#define S2    6
#define S3    5
#define S4    4
```

```
#define S5 3
#define S6 2
#define S7 1
#define S8 0

uint8_t pls7ReadSwitches();
uint8_t pls7SwitchState(uint8_t sw);
```

Listing 18.9: Makro konstante i prototipovi funkcija za rad sa prekidačima

U skladu sa predloženim makro konstantama i funkcijama, ideja je da njihova stanja mogu biti očitana na dva načina. Prvi način omogućava očitavanje stanja pojedinačnih prekidača pozivom funkcije *pls7SwitchState()*. Parametar *sw* predstavlja jednu od makro konstanti (*S1*, *S2*, ..., *S8*) koji odgovara oznaci prekidača na kućištu komponente. Povratna vrednost je 0 ili 1, u zavisnosti od položaja prekidača.

Dруги način omogućava istovremeno očitavanje stanja svih osam prekidača, помоћу funkcije *pls7ReadSwitches()*. U pitanju je funkcija koja vraћа 8-bitnu vrednost u kojoj svaki bit predstavlja stanje prekidača na odgovarajućoj poziciji. При tome treba voditi računa о tome да pozicije bita ne odgovaraju oznakama на kućištu komponente: Stanje krajnjeg levog prekidača označenog sa *S1* predstavljeno je najvišim bitom, односно bitom на poziciji 7. Analogno tome, stanje krajnjeg levog prekidača označenog sa *S8* predstavljeno je najnižim bitom, koji se nalazi na poziciji 0. Veza između oznaka prekidača na kućištu i njihovih bitskih pozicija u okviru povratne vrednosti funkcije takođe je prikazana na slici 18.15. Primer programa koji radi sa prekidačima je dat na listingu 18.10.

```
#include "PLS7.h"

void main()
{
    pls7Init();

    /*citanje stanja prekidaca i prikaz ocitanog stanja
     * prekidaca na LED diodama*/
    while(1)
        pls7WriteDisplay(LEDs, pls7ReadSwitches());
}
```

*Listing 18.10: Primer korišćenja *pls7ReadSwitches()* funkcije*

Pored navedenih funkcija, potrebno je dodati i funkciju za inicijalizaciju periferija koje se koriste prilikom implementacije (portovi mikrokontrolera i odgovarajući tajmer/brojački modul), kao i prekidnu rutinu čija će uloga biti periodično osvežavanje displeja.

18.8 Implementacija biblioteke

Na osnovu željenih funkcionalnosti koje su definisane u prethodnom odeljku, prototipovi funkcija i potrebne makro konstante formiraju strukturu datoteke zaglavlja *PLS7.h*. Sadržaj ove datoteke je prikazan na listingu 18.11.

```
#ifndef PLS7_H_
#define PLS7_H_

#include <stdint.h>

void pls7Init();
#define LEDS 0
#define D1 1
#define D2 2
#define D3 3
#define D4 4

void pls7WriteDisplay(uint8_t display, uint8_t value);
uint8_t pls7ReadDisplay(uint8_t display);

#define LEFT 0
#define DOWN 1
#define RIGHT 2
#define UP 3

uint8_t pls7ButtonState(uint8_t button);

#define S1 7
#define S2 6
#define S3 5
#define S4 4
#define S5 3
#define S6 2
#define S7 1
#define S8 0

uint8_t pls7ReadSwitches();
uint8_t pls7SwitchState(uint8_t sw);
#endif
```

Listing 18.11: PLS7.h

Sa druge strane, u datoteci *PLS7.c* nalaziće se implementacije svih navedenih funkcija, kao i definicije neophodnih globalnih promenljivih. Doseg ovih promenljivih je ograničen samo na modul u okviru kojeg su definisane, pa je stoga uobičajena praksa da im se pristupa preko funkcija za očitavanje vrednosti (tzv. *get* funkcije) i za postavljanje vrednosti (tzv. *set* funkcije). Prema tome, zaglavlje datoteke *PLS7.c* sadržaće `#include` direktive koje pozivaju neophodne module, uključujući

i zaglavljje *PLS7.h*, kao i definicije globalnih promenljivih. Njegov prikaz je dat na listingu 18.12.

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

#include "PLS7.h"

//buffer displeja
uint8_t DISP_BUFFER[5] = {0x00, 0xff, 0xff, 0xff, 0xff};
//indeks trenutno aktivnog displeja
uint8_t disp = 4;
```

Listing 18.12: Uključivanje biblioteka i definisanje globalnih promenljivih

Za osvežavanje displeja može biti korišćen neki od 3 tajmer/brojačka modula koji postoje unutar mikrokontrolera ATmega328P. U okviru ove biblioteke, biće korišćen tajmer/brojač modul 1. Primer implementacije prekidne rutine i funkcije za inicijalizaciju je dat na listingu 18.13.

```
ISR(TIMER1_COMPA_vect)
{
    //prekid tajmer/brojac modula 1 usled dostizanja vrednosti
    //registra OCR2A
    if (++disp > 4)
        disp = 0;
    PORTB = ~(1 << (4 - disp)); //uključenje tranzistora
    PORTD = DISP_BUFFER[disp]; //ispis na trenutno aktivan
    //displej
}

void pls7Init()
{
    //inicijalizacija portova:
    DDRB = 0x3f; //PB5-PB0 -> izlazi DDRC = 0x20; //PC5 ->
    //izlaz
    DDRD = 0xff; //port D -> izlaz

    //inicijalizacija tajmera 1:
    TCCR1B = 0x0B; //tajmer 1: CTC mod + fclk = fosc/64
    OCR1A = 249; //perioda tajmera 1: 250 Tclk (OCR1A + 1 =
    250)
    TIMSK1 = 0x02; //dozvola prekida tajmera 1 usled
    //dostizanja vrednosti registra OCR1A
    sei(); //I = 1 (dozvola prekida)
}
```

Listing 18.13: Prekidna rutina i pls7Init() funkcija

Manipulacija sadržajem displeja kao i određivanje kombinacije aktivnih dioda (LED) vrši se pomoću bafera DISP_BUFFER. Implementacija set (*pls7WriteDisplay()*)

i get funkcije (*pls7ReadDisplay()*) je data u nastavku, na listingu 18.14.

```
void pls7WriteDisplay(uint8_t display, uint8_t value)
{
    DISP_BUFFER[display] = value;
}

uint8_t pls7ReadDisplay(uint8_t display)
{
    return DISP_BUFFER[display];
}
```

Listing 18.14: pls7WriteDisplay() i pls7ReadDisplay() funkcija

Makro konstante za pristup tasterima rešene su na takav način, da vrednost konstante koja je definisana makroom odgovara poziciji pina u okviru porta C na koji je odgovarajući taster povezan. Time je implementacija funkcije *pls7ButtonState()* za očitavanje stanja tastera maksimalno pojednostavljena. Ona je prikazana na listingu 18.15.

```
uint8_t pls7ButtonState(uint8_t button)
{
    if (PIN & (1 << button))
        return 0;
    else
        return 1;
}
```

Listing 18.15: pls7ButtonState() funkcija

Konačno, za očitavanje stanja prekidača, realizovan je serijski protokol koji očitava stanje 8-bitnog pomeračkog registra *74HC165*. Funkcija *pls7ReadSwitches()*, kao povratnu vrednost, vraća 8-bitnu promenljivu čiji biti predstavljaju stanja svih 8 prekidača, dok funkcija *pls7SwitchState()* vraća stanje pojedinačnog prekidača, koji se prosleđuje kao ulazni parametar pomoću makro konstanti S1...S8. Implementacija pomenutih funkcija je prikazana na listingu 18.16.

```
//makroi za kontrolu pinova preko kojih je kontroler povezan
//sa 74HC165:
#define SCL_HI (PORTC |= (1<<5))
#define SCL_LO (PORTC &= ~(1<<5))
#define SDA (PIN & (1 << 4))
#define SHLD_HI (PORTB |= (1<<5))
#define SHLD_LO (PORTB &= ~(1<<5))

uint8_t pls7ReadSwitches()
{
    uint8_t i, tmp = 0, mask = 0x80;
    //impuls za upis stanja prekidaca u register
    SHLD_HI;
```

```

SHLD_L0;
SHLD_HI;

for (i = 0; i < 8; i++)
{
    if (SDA) //provera stanja ulaznog pina tmp |= mask;
    mask >>= 1;

    SCL_L0;
    SCL_HI; //generisanje aktivne ivice takta

}
return tmp;
}

uint8_t pls7SwitchState(uint8_t sw)
{
    uint8_t tmp = pls7ReadSwitches();
    return (tmp & (1 << sw)) >> sw;
}

```

Listing 18.16: pls7ReadSwitches() i pls7SwitchState() funkcija

18.8.1 Primer korišćenja funkcija biblioteke PLS7

U nastavku, na listingu 18.17 je naveden jednostavan primer koji demonstrira rad skoro svih funkcija realizovanih u okviru biblioteke za upravljanje PLS7 ekspanzijonom pločom.

```

#include "PLS7.h"

int16_t main()
{
    pls7Init();

    while(1)
    {
        pls7WriteDisplay(LEDs, pls7ReadSwitches());

        if (pls7ButtonState(LEFT))
            pls7WriteDisplay(D1, 0x0c);
        else
            pls7WriteDisplay(D1, 0xfe);

        if (pls7ButtonState(DOWN))
            pls7WriteDisplay(D2, 0xa4);
        else
            pls7WriteDisplay(D2, 0xfe);
    }
}

```

```
    if (pls7ButtonState(RIGHT))
        pls7WriteDisplay(D3, 0x27);
    else
        pls7WriteDisplay(D3, 0xfe);

    if (pls7ButtonState(UP))
        pls7WriteDisplay(D4, 0xc4);
    else
        pls7WriteDisplay(D4, 0xfe);
}

return 0;
}
```

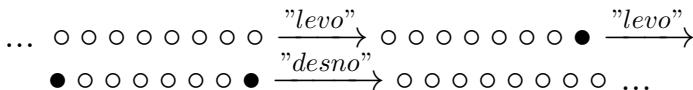
Listing 18.17: Primer korišćenja biblioteke za PLS7

18.9 Zadaci za vežbu II

Zadatak 18.9.1. Proširiti biblioteku **PLS7** sa funkcijom koja omogućava očitavanje svih tastera i shodno tome vraća odgovarajuću povratnu vrednost. Prototip funkcije je dat u nastavku.

- `int8_t readButtons();`
 - **Opis:** Funkcija koja vrši očitavanje stanja tastera na PLS7 ekspanzionoj ploči.
 - **Povratna vrednost:** Odgovarajuća makro konstanta dodeljena svakom tasteru (definisana u okviru zaglavlja *PLS7.h*), odnosno vrednost -1 ukoliko nijedan taster nije pritisnut.

Zadatak 18.9.2. Na početku je uključeno svih osam dioda PLS7 ekspanzionate ploče. Prilikom svakog pritiska tastera levo, potrebno je naizmenično, sa obe strane, umanjivati broj uključenih dioda za 1. Pritiskom na taster desno, ponovo se uključuju sve diode. Za realizaciju zadatka koristiti gotove funkcije iz biblioteke **PSL7**. Primer tranzicije je dat u nastavku.



○ – dioda je uključena ● – dioda je isključena

Zadatak 18.9.3. Na nasumičan način ispisati vrednost iz intervala 0-255 na četiri cifre sedmo-segmentnog displeja. Nakon toga, potrebno je postaviti prekidače u odgovarajući položaj i pritisnuti taster levo. Ukoliko je vrednost učitana putem prekidača identična sa onom koja je prikazana na sedmo-segmentim displejima, potrebno je uključiti sve diode na PLS7 ekspanzionoj ploči. U suprotnom, potrebno je uključiti svaku drugu diodu. Za realizaciju zadatka koristiti gotove funkcije iz biblioteke **PSL7**.

Zadatak 18.9.4. Napisati program koji implementira jednostavan digitron na PLS7 ekspanzionoj ploči. Na početku je potrebno postaviti prekidače koji koduju binarnu vrednost prvog operanda. Zatim se ova vrednost učitava pritiskom na taster desno. Nakon toga, potrebno je postaviti vrednost drugog operanda pomoću prekidača. Na sličan način, data vrednost se učitava pritiskom na taster levo. Konačno, pritiskom na taster gore ili dole, vrši se operacija sabiranja odnosno oduzimanja, a rezultat se ispisuje na sedmo-segmentnom displeju. Za realizaciju zadatka koristiti gotove funkcije iz biblioteke **PSL7**.

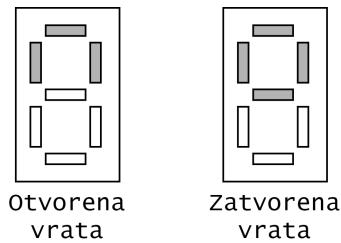
Zadatak 18.9.5. Napisati program koji implementira štopericu na PLS7 ekspanzionoj ploči. Na ciframa sedmo-segmentnog displeja se prikazuje vreme u formatu *MM.SS*, gde M označava minute, a S sekunde. Pritiskom na taster levo započinje merenje vremena, dok se pritiskom na taster desno zaustavlja isto. Pritiskom na taster gore vrši se resetovanje štopericice. Za realizaciju zadatka koristiti gotove funkcije iz biblioteke **PSL7**.

Zadatak 18.9.6. Napisati program koji implementira digitalni sat na PLS7 ekspanzionoj ploči. Neophodno je da sat, neprekidno od spuštanja na ploču, ispisuje proteklo vreme na sedmo-segmentnom displeju. Trenutno vreme je potrebno prikazati u formatu *MM.SS*, gde M označava minute, a S sekunde. Pritisakom na tastere levo i desno smanjuje se, odnosno povećava trenutna vrednost sekundi, a pritiskom na tastere gore i dole povećava se i smanjuje trenutna vrednost minuta. Za realizaciju zadatka koristiti gotove funkcije iz biblioteke **PSL7**.

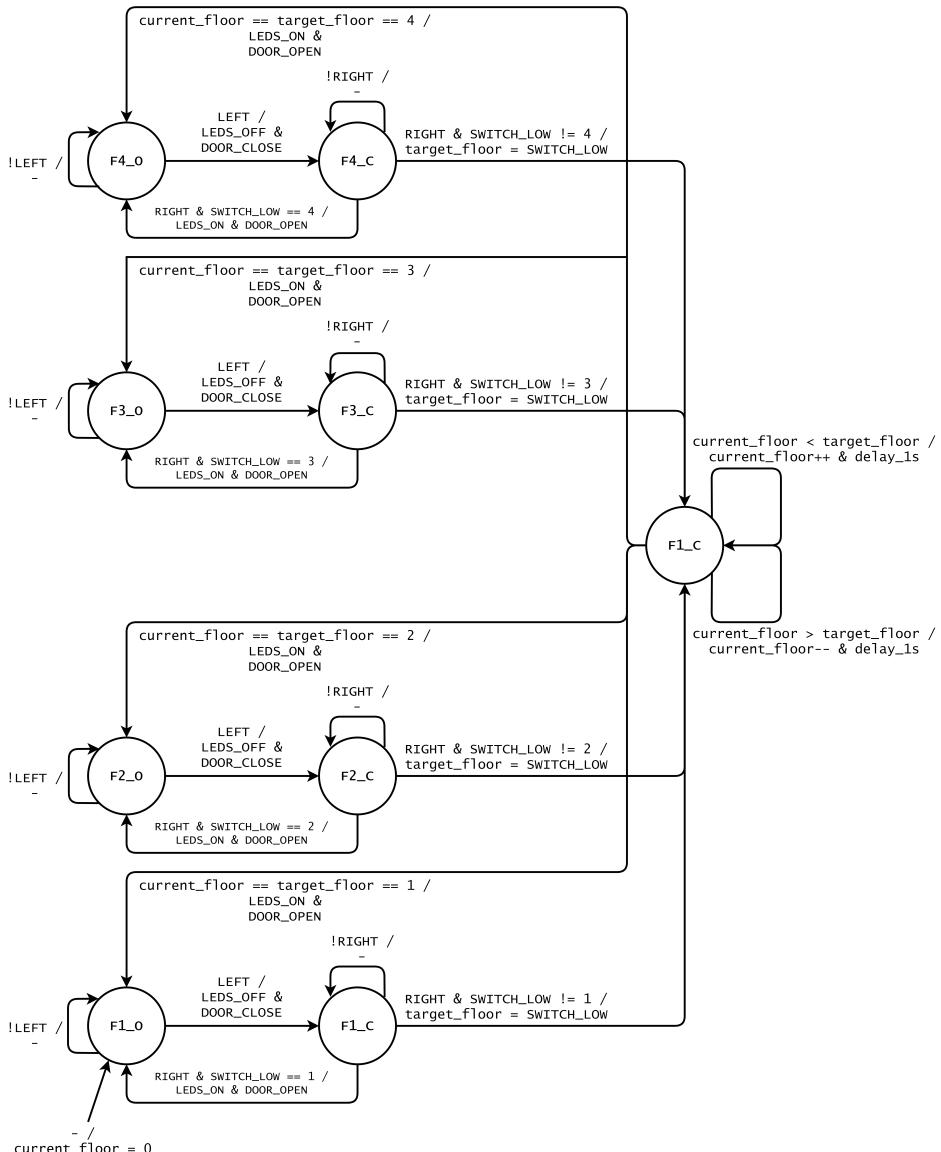
18.10 Zadaci za vežbu III

Zadatak 18.10.1. Napisati program koji simulira rad lifta upotrebom PLS7 ekspanzione ploče. Položaj lifta potrebno je prikazati na sedmo-segmentnom displeju tako što svaka cifra predstavlja po jedan sprat. Lift prikazati na odgovarajućem spratu (cifri sedmo-segmentnog displeja), gde je moguće da lift ima otvorena ili zatvorena vrata, kao što je ilustrovano na slici 18.16.

Funkcije *DOOR_OPEN* i *DOOR_CLOSE* otvaraju i zatvaraju vrata. Ponašanje lifta implementirati kao mašinu stanja opisanu dijagramom sa slike 18.17. *LEDS_ON* označava uključivanje svih dioda, dok *LEDS_OFF* označava gašenje svake druge diode. *LEFT* i *RIGHT* označavaju vrednosti istoimenih tastera, a *SWITCH_LOW* vrednost donja tri prekidača. *Current_floor* i *target_floor* su promenljive čije su inicijalne vrednosti 0. Napraviti pauzu dužine 1s prilikom pojave *delay_1m*, merenu pomoću Tajmera 0.



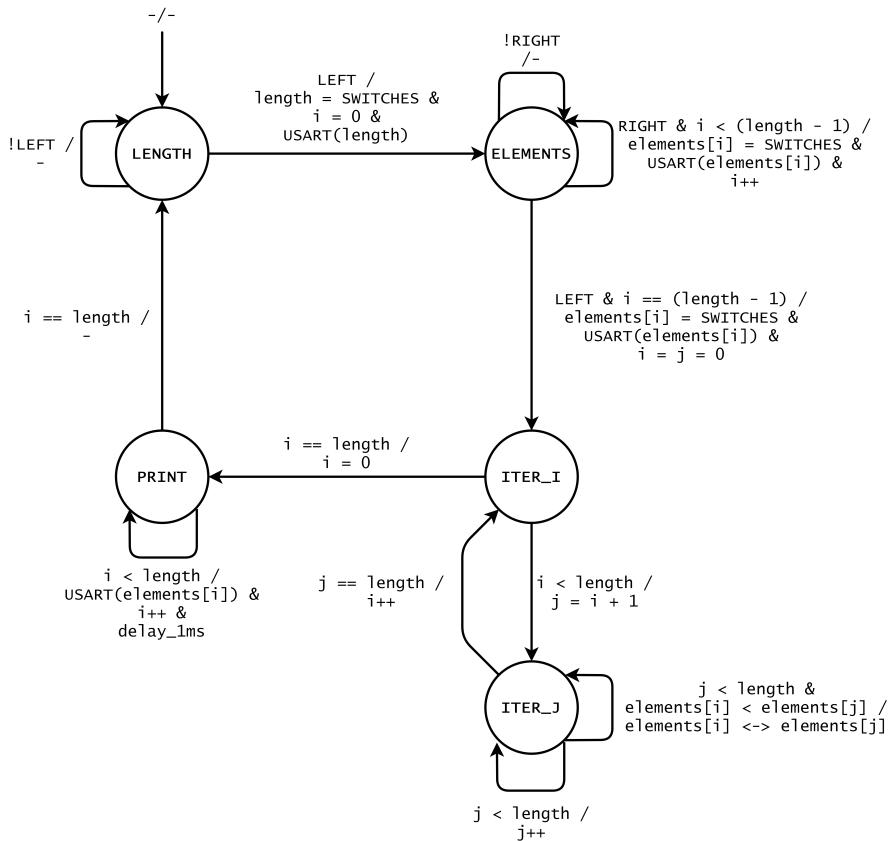
Slika 18.16: Prikaz lifta na sedmo-segmentnom displeju



Slika 18.17: Mašina stanja koja simulira rad lifta

Zadatak 18.10.2. Napisati program koji omogućava unos niza, njegovo sortiranje i prikazivanje pomoću mašine stanje prikazane na slici 18.18. Dužina unetog niza se unosi na početku, pritiskom na taster levo, kao vrednost učitana preko prekidača. Prekidači predstavljaju odgovarajuće bite unesene vrednosti. Nakon unosa dužine, preko tastera desno i prekidača se unose vrednosti niza. Nakon što je poslednja vrednost niza uneta, niz se sortira. *SWITCHES* označava očitavanje prekidača. *LEFT* i *RIGHT* označavaju vrednosti tastera. *USART* označava ispis vrednosti

upotreboom serijske komunikacije. Simbol \leftrightarrow predstavlja zamenu (eng. *swap*) mesta elemenata.



Slika 18.18: Mašina stanja koja implementira sortiranje niza

Zadatak 18.10.3. Modifikovati mašinu stanja iz Zadatka 18.10.2 tako da se svi ispisi (dužina i elementi niza) sada izvrše na sedmo-segmentnom displeju umesto putem serijskog terminala. Pri tome, uzeti u obzir da vrednosti dužine i elemenata niza mogu biti višecifrene.

Zadatak 18.10.4. Kreirati mašinu stanja koja omogućava unos broja preko prekidača i proveru da li je taj broj prost. Nakon unosa broja, prikazati ga preko sedmosegmentnog displeja Tokom provere, uključiti sve diode. Nakon provere, diodama signalizirati da li je broj prost ili ne. Ukoliko jeste, uključiti po jednu diodu na krajevima, u suprotnom, uključiti središnje dve diode. Program je potreban implementirati kao mašinu stanja.