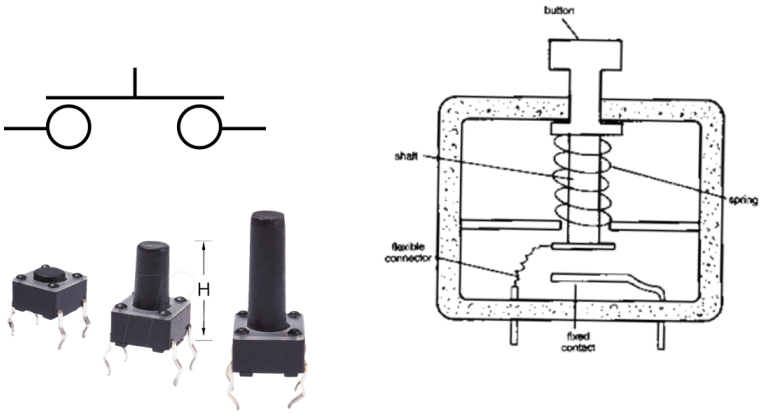
# GPIO (*General Purpose Input Output*) - drugi deo -

## 1 Tasteri

Tasteri su neizostavne komponente u mikrokontrolerskim sistemima jer na najjednostavniji način omogućuju interakciju između korisnika i mikrokontrolera. Na slici 1 dati su prikazi korišćenih tastera u ovoj vežbi.




Slika 1: Simbol tastera (gore levo), fizički izgled tastera (dole levo), unutrašnja struktura tastera (slika desno)

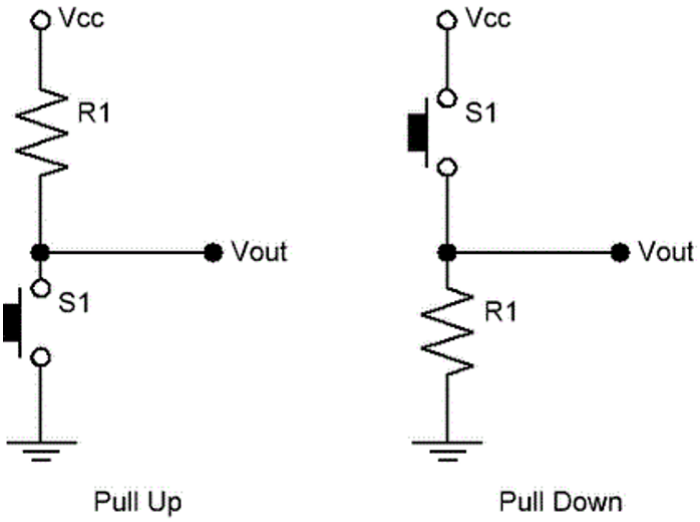
Na EasyPIC v7 PRO razvojnoj ploči, na svakom od pinova mikrokontrolera nalazi se po jedan taster uz pomoć kog se može simulirati ulazni signal na datom pinu. Takođe, pored tastera se nalaze i igličasti konektori uz pomoć kojih se mogu do mikrokontrolera dovoditi spoljašnji signali poput signala sa enkodera, indikatora i sl.

### 1.1 Pull-up i pull-down otpornici

Ako bi se na pin mikrokontrolera povezao samo taster, pojavio bi se problem nedefinisanog stanja kada taster nije pritisnut (*floating state*). Uzročnik postojanja ovog stanja je visoka ulazna impedansa pina mikrokontrolera, čime pin postaje idealna antena za šum iz okoline. Kako bi se ovaj problem rešio uvodi se pojam **pull-up** i **pull-down** otpornika, čija je uloga da definišu stanje na ulaznim pinovima kada tasteri nisu pritisnuti. Pull-up i pull-down otpornici dati su na slici 2. Kada je prisutan pull-up otpornik on definiše stanje logičke jedinice na pinu kada taster nije pritisnut, a kada je taster pritisnut na pinu se uspostavlja stanje logičke nule. Nasuprot njemu pull-down otpornik definiše stanje logičke nule kada je taster otpušten, dok će se pritiskom na taster na pinu uspostaviti stanje logičke jedinice.

Odlika indukovane smetnje je mala energija, gde se prilikom odvođenja struje smanjuje amplituda smetnje. Pošto ulazni pinovi imaju veliku otpornost, gotovo da ne postoji struja u ulazni pin čime smetnja postaje dominantna na pinu. Pull-up odn. pull-down otpornik uspostavljaju tok struje dok ne dođe do željenog stanja

logičke jedinice odn. logičke nule, respektivno. Na ovaj način, uticaj šuma od strane okoline je otklonjen.



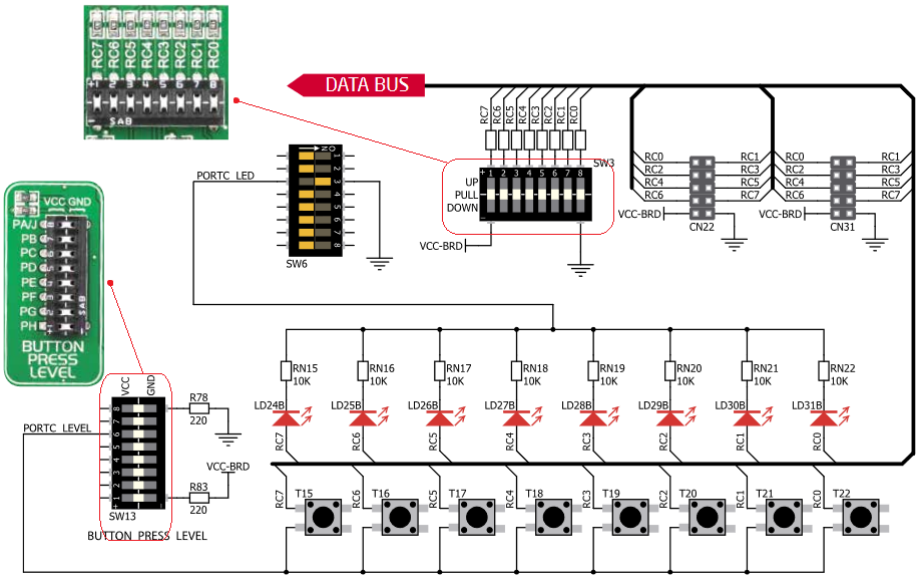

Slika 2: Pull-up i pull-down otpornici

Treba uzeti u obzir da velika otpornost pull-up ili pull-down otpornika može postati samerljiva sa ulaznom otpornošću pina i tako onemogućiti detekciju pritiska tastera. Npr. Ako je ulazna impendansa 1 MΩ, a i pull-up otpornik je 1 MΩ, pri napajanju od 5 V kada taster nije pritisnut na pinu će biti detektovano 2,5 V, zbog pojave naponskog razdelnika. Ovo dovodi do zaključka da je najbolja opcija da bi pull-up odn. pull-down otpornik trebao imati malu otpornost. Sa druge strane što je manja otpornost pull-up odn. pull-down otpornika biće veća i struja kroz njih prilikom pritiska tastera. Iz tih razloga uobičajeno je korišćenje otpornika reda ∼10 kΩ za spore promene (tasteri), dok za brže promene (I2C) otpornosti su ∼1 kΩ ali to varira i od kapacitivnosti prisutnih na pinu. Spram vrednosti pull-up i pull-down otpornici se dele na slabe (velika vrednost otpornosti) i jake (mala vrednost otpornosti) otpornike.

Neki od pinova mikrokontrolera sadrže unutrašnje slabe pull-up (*weak pull-up*) otpornike. Za mikrokontroler PIC18F87K22 portovi *PORTB*, *PORTD*, *PORTE* i *PORTJ* sadrže unutrašnje weak pull-up otpornike, koji mogu biti konfigurisani *SFR* bitovima $\overline{RBPU}$, *RDPU, REPU* i *RJPU*. Treba imati u vidu da je otpornost ovih pull-up otpornika nekoliko destina kΩ. Tačni podaci o njihovim vrednostima se ne mogu pronaći ali se mogu pronaći podaci o struji (min 50 $\mu A$ max 400 $\mu A$), jer su oni realizovani kao strujni izvori (a vrednost zavisi od fabrikacije, temperature itd.). Ovi otpornici su idealni za situcaije kada se na pin povezuje uređaj koji nema jasno definisana 0/1 stanja poput: tastera, prekidača, open collector izlaza itd.

Na EasyPIC v7 PRO svaki pin poseduje prekidač sa tri položaja kojim se odabira pull-up ili pull-down otpornik vrednosti 4,7 kΩ. Srednji položaj isključuje i pull-up i pull-down otpornik, gornji položaj uključuje pull-up otpornik, a donji položaj uključuje pull-down otpornik. Takođe, pored pomenutih prekidača postoje prekidači za kontrolu dovedenog stanja na port pritiskom na tastere pripadajućeg porta. Na ploči ovi prekidači imaju oznaku *Button press level* i nalaze se u donjem levom

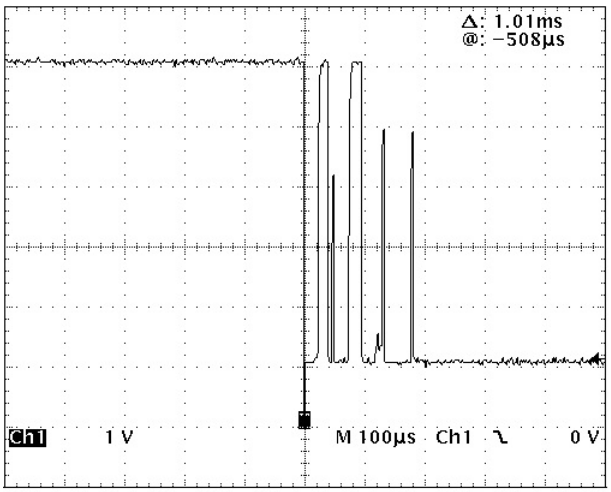
uglu ploče pored dioda porta A. Takođe, ovi prekidači imaju tri stepena: srednji - nedefinisano stanje, levi položaj - $V_{CC}$ i desni položaj - *GND*. Ova dva prekidača su prikazana na slici 3 kao i njima pripadajuća električna šema za PORTC.




Slika 3: Raspored prekidača i šema I/O pinova PORTC

## 1.2 Treperenje kontakata

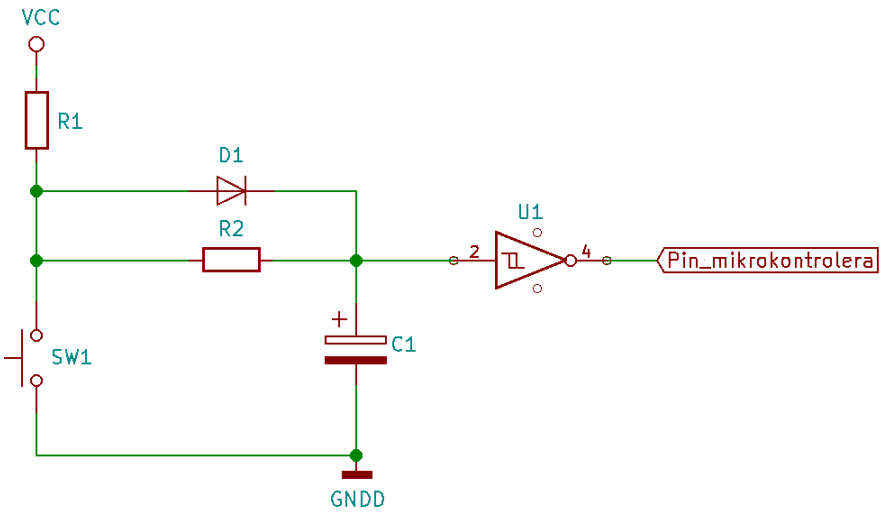
Zbog mehaničke prirode kontakata prilikom pritiska na taster javlja se prelazni proces u vidu šuma. Ovaj prelazni proces se naziva **treperenje kontakata** (*bouncing*). Treperenje kontakata traje nekoliko stotina $\mu s$ pa čak i do nekoliko $ms$ zavisno od vrste korišćenog tastera tj. prekidača. Na slici 4 prikazan je snimak ekrana osciloskopa prilikom pritiska na taster.




Slika 4: Problem treperenja kontakata

### 1.2.1 Hardversko otklanjanje treperenja kontakata

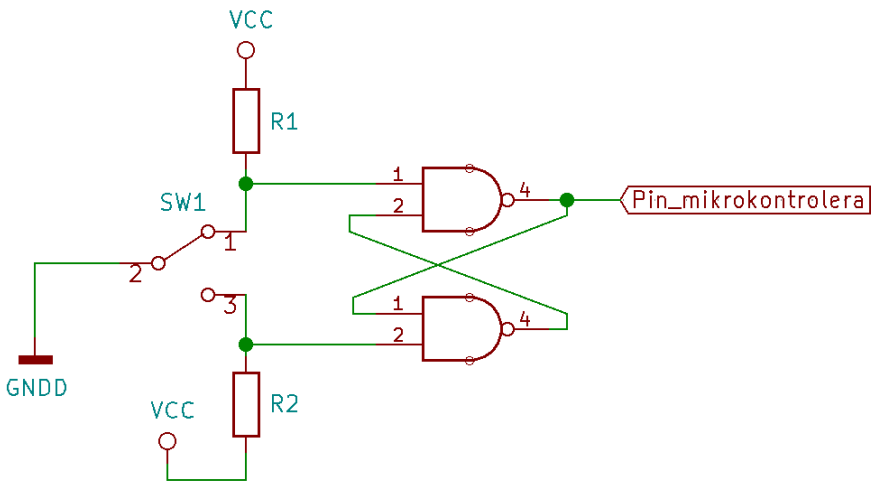
Kako bi se rešio ovaj problem najčešće se koristi RC low-pass filter koji je prikazan na slici . Otklanjanje treperenja kontakata naziva se *debouncing*.




Slika 5: Debouncing RC filtrom

Ovo rešenje se zasniva na integratoru sa vremenskom konstantnom ∼RC. Ako je taster inicijalno otpušten, kondenzator se puni preko $R_1$ i diode. U jednom trenutku Šmit triger će detektovati promenu stanja i ta vrednost postaje stabilna na pinu mikrokontrolera. Uloga diode u ovom kolu je da se ubrza punjenje kondenzatora. U slučaju da se pritisne taster, vrši se pražnjenje kondenzatora kroz otpornik $R_2$, pa će u jednom trenutku Šmit triger detektovati promenu stanja na logičku nulu.

Drugo rešenje koje se koristi za otklanjanje treperenja kontakata primenjivo je samo na dvopoložajne tastere (SPDT-Single Pole Double Throw) i primenom SR leča (realizacija sa dva NAND kola). Ovo kolo prikazano je na slici 6.



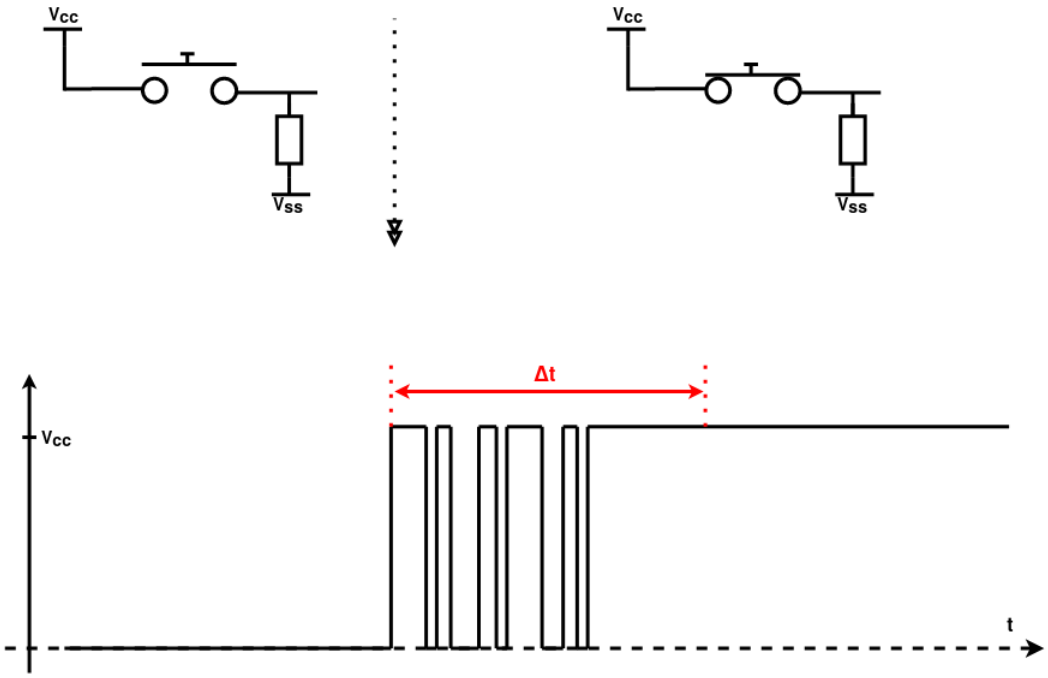

Slika 6: Debouncing SPDT prekidača SR lečom

Ako je na ulazu u gornje NAND kolo logička 1, a na ulazu donjeg NAND kola logička 0 (prekidač postavljen u donji položaj) onda je na ulazu u mikrokontroler logička 0. Ako je na ulazu u gornje NAND kolo logička 0, a na ulazu donjeg NAND kola logička 1 (prekidač postavljen u gornji položaj) onda je na ulazu u mikrokontroler logička 1. Važno je zapaziti da ako su oba ulaza na logičkim jedinicama onda izlaz pamti prethodno stanje. Pošto je nemoguće da dođe do zabranjenog stanja (logičke nule na oba ulaza) ovo kolo će uspešno otklanjati bilo kakvo treperenje kontakata. U ovom slučaju treperenje kontakata će se desiti prilikom pomeranja preklopnika iz jednog položaja u drugi, a tom prilikom na jednom od ulaza će pull-up otpornik držati stabilno stanje logičke jedinice.

Razvijeni su različiti čipovi za otklanjanje treperenja kontakata od kojih su neki: MC14490, LTC6994, MAX6816 itd.

### 1.2.2 Softversko otklanjanje treperenja kontakata

Moguće je implementirati funkciju za otklanjanje treperenja kontakta, koja proverava stanje na pinu pre i posle vremenske zadrške $\Delta t$ i na osnovu tih vrednosti zaključuje da li je na pinu bilo ustaljeno (željeno stanje) stanje. Na slici 7.




Slika 7: Otklanjanje treperenja kontakata softverskim putem

Problem rada u softverskog debouncing-a se ogleda u tome da se oduzima vreme procesoru kako bi vršio jednostavan zadatak provere, a pri tome će morati da čeka neko vreme.

Postoje i drugi vidovi rešenja poput debouncing metode date u primeru 4.

## 1.3 Primeri programa za očitavanje vrednosti sa pinova

***Primer 1.*** *Program za detekciju pritiska tastera.*

U ovom primeru se vrši jednostavna provera da li je na pin 0 porta B dovedena logička 1. Ako jeste onda će ceo PORTD biti postavljen na visok nivo (sve led diode su uključene), a u suprotnom biće postavljen na nizak logički nivo (sve led diode su isključene). Da bi se uočili nedostaci ovakve detekcije neophodno je dodati par izmena u kodu.

Listing 1: Detekcija pritiska tastera

```
#include "mcc_generated_files/mcc.h"

void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        if(PORTBbits.RB0)
        {
            LATD = 0xFF;
        }
        else
        {
            LATD = 0x00;
        }
    }
}
```

***Primer 2.*** *Program za detekciju pritiska tastera - nastavak*

Listing 2: Detekcija pritiska tastera

```
#include "mcc_generated_files/mcc.h"

void main(void)
{
    SYSTEM_Initialize();

    LATD = 0xAA;
    while (1)
    {
        if(PORTBbits.RB0)
        {
            LATD = ~LATD;
        }
    }
}
```

Naizgled kodovi dati u listingu 1 i listingu 2 su identični, a jedinu razliku predstavlja inverzija stanja na portu D odn. inverzija u registru LATD. Inicijalna upisana vrednost u LATD je 0xAA odn. naizmenično uključene i isključene diode. Prilikom pritiska na taster velikom brzinom se vrši prebacivanje iz 0xAA i njoj invertovane vrednosti (0x55), čime se stiče utisak da su pri pritisku tastera sve diode uključene. Nakon otpuštanja tastera dolazi do zadržavanja stanja ili 0xAA ili 0x55. Ovaj efekat je posledica brzine provere uslova "Da li je taster RB0 pritisnut?". Po pritisku tastera ovaj uslov će biti tačan, pa će kao uzrok imati stalnu inverziju dok korisnik

ne otpusti taster. Po otpuštanju u LATD ostaje zapamćena poslednja invertovana vrednost.

U velikom broju slučajeva u interesu je samo izvršiti izraze pod uslovom, za jedan pritisak tastera. Iz tih razloga se uvodi detekcija ulazne odn. silazne ivice.

***Primer 3.*** *Program za detekciju silazne ivice tastera.*

Da bi se detektovala silazna ivica neophodno je ispuniti sledeće uslove:

- Neophodno je znati da je postojalo stanje logičke jedinice.
- Nakon stanja logičke jedinice neophodno je detektovati prvo stanje logičke nule.

Između ova dva uslova desila se silazna ivica signala. Da bi se ovo implementiralo u kodu, najjednostavnije je uvesti dodatnu promenljivu. U ovom slučaju uvedena je promenljiva *flag* kao bool tip. Da bi se omogućilo korišćenje bool tipa, neophodno je korsititi *stdbool* biblioteku. U svetu programiranja pojam *flag* označava promenljivu, najčešće binarnu (dve moguće vrednosti), uz pomoć koje se reguliše tok programa. Kada se *flag* setuje to sugeriše da je ispunjen neki željeni uslov tj. desio se odgovarajući događaj. U primeru 3 *flag* promenljiva vrši detekciju stanja logičke 1 na pinu RB0. Drugi *if* uslov proverava da li se desio *flag* (bilo je visoko stanje na pinu) i da li je na pinu RB0 stanje logičke 0. Ako je ovaj uslov ispunjen uspešno je detektovana silazna ivica. Da bi se zabranila ponovna detekcija na logičkoj 0, neophodno je *flag* postaviti na *false* stanje. Na ovaj način promenljiva *flag* će biti setovana na *true* tek ponovnim dovođenjem logičke 1 na RB0 pin. Na ovaj način osigurava se ponovna detekcija silazne ivice na pinu RB0.

Prilikom otpuštanja tastera može se uočiti da dolazi do detekcije više silaznih ivica. Ovo je direktna posledica treperenja kontakata.

Listing 3: Detekcija silazne ivice tastera

```
#include "mcc_generated_files/mcc.h"
#include<stdbool.h>

void main(void)
{
    SYSTEM_Initialize();

    bool flag;
    LATD = 0x00;
    flag = false;

    while (1)
    {
        if(PORTBbits.RB0)
        {
            flag = true;
        }
        if(flag && !PORTBbits.RB0)
        {
            LATD++;
            flag = false;
        }
    }
}
```

***Primer 4.*** *Program za detekciju silazne ivice tastera i otklanjanje treperenja kontakata metodom pomeranja bita.*

Listing 4: Detekcija silazne ivice tastera i otklanjanje treperenja kontakata

```
#include "mcc_generated_files/mcc.h"
#include <stdbool.h>
#include <stdint.h>

static bool debounce(uint8_t *port, uint8_t pin)
    {
    static uint32_t temp = 0;
    temp<<=1;
    temp|=(*port&(1<<pin))>>pin;
    return temp == 0x80000000 ? true : false;
    }

void main(void)
{
    SYSTEM_Initialize();

    while (1)
    {
        if(debounce(&PORTJ, 4))
        {

            LATD++;
        }
    }
}
```

U ovom primeru napravljena je funkcija *debounce*, čija je uloga da vrši otklanjanje treperenja kontakata pri detekciji silazne ivice. Algoritam se zasniva na jednostavnom pomeranju bita u promenljivoj *temp* gde se pri svakom pozivu funkcije *debounce* očitana vrednost sa željenog pina (u ovom slučaju *RJ4*) upisuje na mesto najnižeg bita u promenljivoj *temp* (u ovom slučaju 32-bitna promenljiva). Izraz nakon *return* predstavlja "if u jednom redu", gde se prvo postavlja iskaz koji se proverava, zatim ?, pa vrednost koja se upisuje ako je tačan iskaz, za kojim sledi : i na kraju vrednost ako je dati iskaz netačan. U ovom slučaju se poredi da li je temp jednak sa 32-bitnim brojem (*0b1000 0000 0000 0000 0000 0000 0000 0000*), a to će biti slučaj samo kada je taster otpušten i kada mikrokontroler detektuje 31 uzastopnu nulu. Ujedno, to će označavati i kraj treperenja kontakata i predstavljati ustaljeno stanje na pinu. Važno je napomenuti da promenljiva *temp* mora biti *static*. To znači da pri sledećem pozivu ove funkcije u RAM memoriji ostaje upamćena njena prethodna vrednost. Ako bi se deklarisala kao obična promenljiva (npr. *uint32_t temp = 0;*) onda bi njena vrednost bila stalno 0, a samim tim bi algoritam bio neupotrebljiv.

***Primer 5.****Program za detekciju uzlazne ivice tastera i otklanjanje treperenja kontakata metodom vremenske zadrške.*
Ovaj kod se zasniva na algoritmu provere stanja pina pre i posle vremenske zadrške. Funkciji *debounce* prosleđuju se pokazivač na PORT registar koji se želi proveriti, pin prosleđenog porta, vreme trajanja vremenske zadrške u ms i stanje koje se želi proveriti (true - stanje logičke jedinice odn. *false* - stanje logičke nule). Nakon toga se proverava da li je na prosleđenom pinu bilo prisutno prosleđeno stanje i u slučaju da nije funkcija vraća *false*. U slučaju da je rezultat true, realizuje se vremenska

zadrška trajanja definisanog prosleđenim parametrom vreme. Važno je istaći da funkcija *__delay_ms()* očekuje konstantu vrednost, stoga se uvodi *while* petlja sa zadatkom realizacije vremenske pauze. Petlja će izvršiti pauzu od 1 ms onoliko puta koliko je to definisano prosleđenim parametrom vreme. Nakon izvršene pauze još jednom se proverava da li je stanje na pinu jednako prosleđenom stanju i ako jeste funkcija vraća vrednost *true*. U suprotnom ako stanja nisu jednaka, funkcija će se završiti vraćanjem *false*. Detekcija uzlazne ivice vrši se slično kao u primeru 3. s tim što se sada promenljiva *flag* postavlja na vrednost *true* dok je taster otpušten, kako bi se stekli uslovi za detekciju uzlazne ivice. U prvom trenutku kada se detektuje stanje logičke jedinice i ako je pri tome *flag* postavljeno na *true* onda je detektovana uzlazna ivica. Promenljivu *flag* neophodno je postaviti na vrednost *false* kako bi se tek po otpuštanju tastera stekli uslovi za generisanje nove uzlazne ivice.

Program za detekciju uzlazne ivice tastera i otklanjanje treperenja kontakata sa vremenskom zadrškom dat je u listingu 5.

Listing 5: Detekcija uzlazne ivice tastera i otklanjanje treperenja kontakata

```
#include "mcc_generated_files/mcc.h"
#include <stdbool.h>
#include <stdint.h>

bool debounce(uint8_t *port, uint8_t pin,uint16_t vreme, bool stanje)
{
    if((*port&(1<<pin)) == stanje<<pin)
    {
        while(vreme--)
            __delay_ms(1);
        if((*port&(1<<pin)) == stanje<<pin)
            return true;
    }
    return false;
}

void main(void)
{
    SYSTEM_Initialize();

    LATD = 0x00;
    bool flag;
    flag = true;
    while (1)
    {
        if(debounce(&PORTJ, 4, 10,true)&&flag)
        {
            LATD++;
            flag = false;
        }
        if(debounce(&PORTJ, 4, 10, false))
        {
            flag = true;
        }
    }
}
```