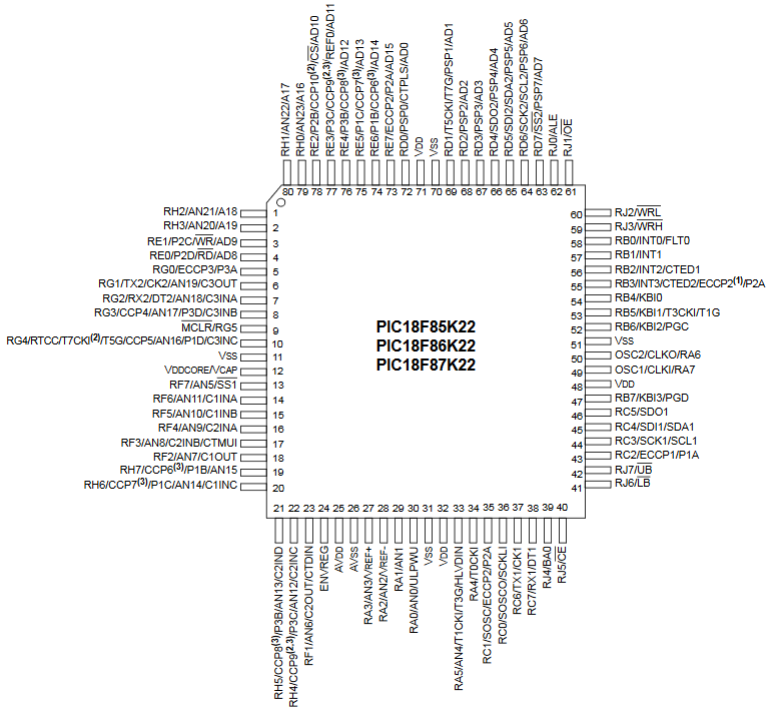
GPIO (*General Purpose Input Output*) - prvi deo -

1 Rad sa ulazno/izlaznim pinovima mikrokontrolera

Da bi mikrokontrolerski sistem stupio u kontakt sa spoljašnjom okolinom neophodno je koristiti GPIO (*General Purpose Input Output*) pinove. Uz pomoć pinova i njima pridodatih modula, mikrokontroler ima mogućnost nadgledanja nekog procesa, kao i preduzimanje odgovarajućih akcija ako je neophodno da se taj proces odigrava prema željenim zahtevima. Prema smeru toka podataka GPIO pinove možemo konfigurisati na dva načina:

- Ulazni pin - smer signala je od spoljašnjeg sveta ka mikrokontroleru
- Izlazni pin - smer signala je od mikrokontrolera ka spoljašnjem svetu

Na slici 1 prikazan je raspored pinova na kontroleru PIC18F87K22 koji će biti podrazumevani kontroler za ove i sve naredne vežbe.



Slika 1: Raspored pinova na PIC18F87K22 mikrokontroleru

Neophodno je uočiti da su svi pinovi kontrolera povezani u celine nazvane portovima. Pošto je korišćeni mikrokontroler 8 bitne arhitekture, svi registri su širine 8 bita, pa je svakom portu pridruženo po najviše 8 pinova. Radi lakšeg korišćenja portovi su označeni slovima: A, B, C itd. Na slici 1 moguće je videti da su svi pinovi, izuzev onih naznačenih sa V_{ss} i V_{dd}, $ENVREG$, V_{DDCORE}/V_{cap}, AV_{ss} i AV_{ss}, nazvani

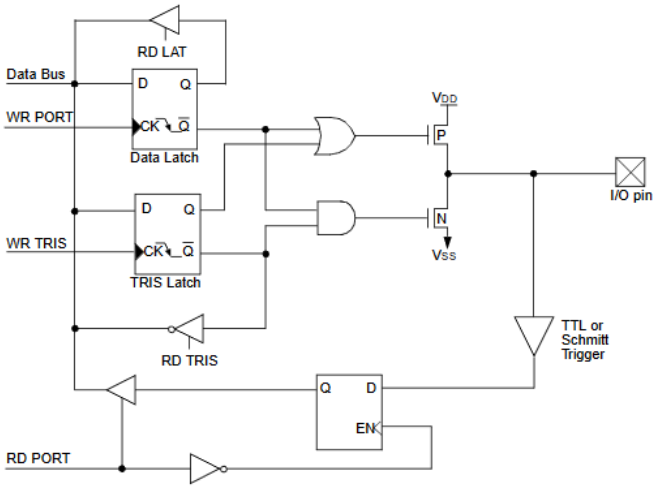
Rxy, gde x predstavlja oznaku porta, a y poziciju pina u datom portu. Važno je napomenuti da numeracija pinova ide od 0. Npr. Pin. *RC6* predstavlja sedmi pin porta C, dok pin *RB2* predstavlja treći pin porta B. Pored oznake *Rxy* moguće je uočiti još oznaka na svakom od pinova npr. *RC4/SDI1/SDA1*, gde *SDI1/SDA1* predstavljaju dodatne funkcionalnosti pinova. O pojedinim funkcionalnostima biće reči u narednim vežbama.

1.1 SFR (*Special Function Registers*) registri I/O portova

Prilikom dizajniranja serije mikrokontrolera, dizajneri (u ovom slučaju Microchip) definišu određeni broj registara specijalne namene - SFR (*Special Function Registers*). Njihova uloga je da se uz pomoć njih pristupa određenim modulima mikrokontrolera. Na ovaj način se omogućuje interakcija između softvera i hardvera unutar mikrokontrolera. Nasuprot SFR postoje i GPR (*General Purpose Registers*) i oni se koriste pri izvršavanju koda za skladištenje promenljivih. Mikrokontroler PIC18F87K22 ima 230 SFR registara.
Da bi se konfigurisali i koristili pinovi mikrokontrolera neophodno je raditi sa tri grupe SFR registara:

- *TRIS* - određuje smer kretanja podataka (signala)
- *PORT* - očitava ili upisuje vrednosti na pin mikrokontrolera
- *LAT* - očitava ili upisuje vrednost u izlazni leč

Na slici 2 prikazana je blok šema pina mikrokontrolera PIC18F serije.



Slika 2: Tipičan izgled svakog pina mikrokontolera

Da bi se u potpunosti razumela šema neophodno je znati da će PMOS tranzistor provoditi za vreme logičke 0 na gejtu, dok će NMOS provoditi za vreme logičke 1 na gejtu.

1.1.1 Upis u TRIS registar

Da bi se odgovarajući pin postavio kao ulazni neophodno je u pripadajući *TRIS* registar upisati logičku 1, a ako se želi podesiti kao izlazni pin neophodno je upisati logičku 0. Radi jednostavnijeg pamćenja 1 podseća na slovo I, pa će pin biti *Input*, dok 0 podseća na slovo O pa se radi o *Output* pinu.

Posmatrajmo sliku 2 tj. linije *WR TRIS* i *Data Bus*. Kada D flip flop dobije silaznu ivicu takta tada se vrednost sa *Data Bus* linije upisuje u *TRIS Latch*. Ako je korisnik upisao 1 (pin je ulazni), na $\overline{Q}$ će se generisati stanje logičke 0. Ovo uzrokuje da će *I* kolo koje kontroliše donji NMOS generisati 0 (jer je na jednom ulazu 0, pa je rezultat logičkog *I* 0). Na ovaj način NMOS tranzistor se drži u isključenom stanju. *Q* izlaz, u ovom slučaju generiše stanje logičke 1, je povezan na *ILI* kolo pa će PMOS dobijati signal logičke jedinice (kada je na ulazu logičkog *ILI* kola barem jedna logička 1 onda je rezultat na izlazu uvek logička 1). Ovo rezultuje da će i PMOS tranzistor biti isključen. Na taj način je moguće vršiti samo očitavanje stanja na pinu odn. pin je ulazni.

U suprotnom ako se u *TRIS Latch* upiše vrednost logičke 0, onda će stanje na izlazu *ILI* i *I* logičkih kola isključivo zavisiti od $\overline{Q}$ *Data Latch* flip flopa. Na ovaj način se kontroliše šta će biti dovedeno na pin, pa je samim tim pin izlazni.
Kako bi se postavila željena vrednost TRIS registra neophodno je znati raspored pinova u registru. Na slici 3 dat je raspored bitova (koji se odnose na pinove) u registru.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0

Slika 3: Raspored bitova u TRISB registru

Svaki bit u *TRISB* registru vezan je za upravljanje tokom podataka na odgovarajućem pinu mikrokontrolera i to od *RB7* do *RB0* respektivno. Npr. ako se želi postaviti da pinovi *RB7, RB6, RB5, RB4* budu ulazni, a *RB3, RB2, RB1* i *RB0* budu izlazni neophodno je *TRISB* postaviti na 0b11110000 (što je ekvivalent 0xF0 odn. 240). Preporuka je da se vrednosti u SFR registre zapisuju u heksadecimalnom formatu, jer je tada kod dosta kompaktniji, a samim tim i čitljivost je bolja. Važno je uočiti da je prefiks za heksadecimalno *0x*, za binaran zapis *0b*, dok za decimalni zapis nije neophodno pisati prefiks.

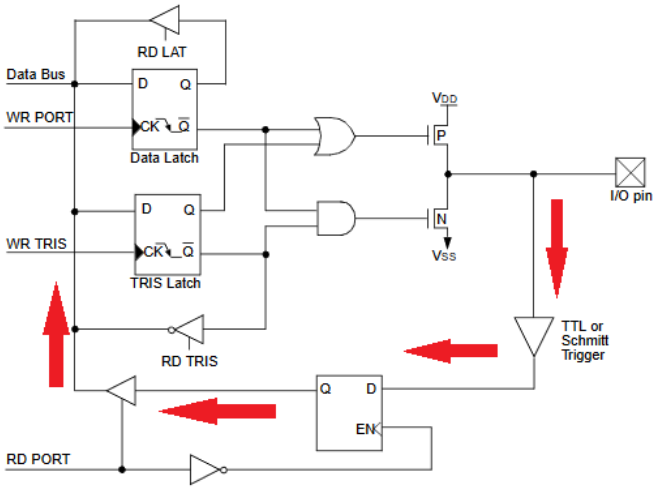
Postavljanje vrednosti u *TRIS* registar za bilo koji port je identično kao i za port B, tako što se na *TRIS* dodaje slovo željenog porta npr. *TRISA*, *TRISB*, *TRISC* itd. Važno je napomenuti da su svi pinovi pri inicijalizaciji postavljeni kao ulazni (stanje visoke impendanse), pa se na taj način sprečava da se greškom ne izazove neka neželjena akcija na pinu mikrokontrolera.

Ako se želi promeniti samo jedan bit to je moguće odraditi na sledeći način. Nakon naziva registra, dodaje se *bit* oznaka, praćena tačkom, nakon koje sledi *Rxy*, gde i x predstavlja oznaku željenog porta, a y vrednost od 0 do 7 odn. odgovarajućeg bita u registru. Uzmimo na primer da se želi postaviti RC6 pin kao izlazni, onda bi se

tom pinu jednostavno moglo pristupiti izvršavanjem TRISCbits.TRISC6 = 0, čime pin RC6 postaje izlazni.

1.1.2 Očitavanje iz PORT registra

Na slici 4 prikazan je smer toka podataka prema PORTx registru. Korisnik ima mogućnost da u svakom trenutku izvrši očitavanje logičkih stanja na pinu mikrokontrolera, jednostavnim očitavanjem registra PORTx registra.



Slika 4: Očitavanje PORTx registra

U primeru na slici 5 dat je raspored bitova (koji se odnose na pinove) u PORTB registru.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0

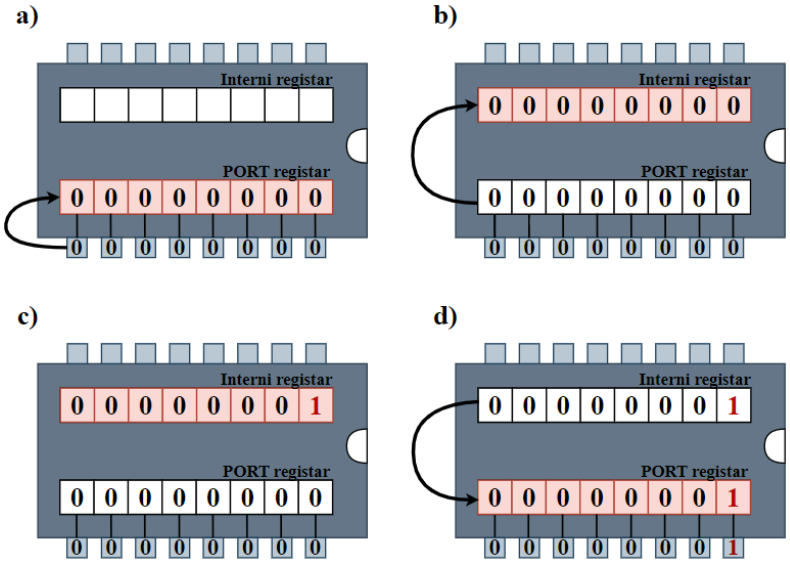
Slika 5: Raspored bitova u PORTB registru

Ako se želi očitati digitalna vrednost sa odgovarajućeg porta, neophodno je skladištiti ili obraditi vrednost PORTx. Ako se želi npr. očitati digitalno stanje sa pina RH0, onda je to moguće odraditi pozivanjem PORTHbits.RH0, što će vratiti vrednost 1 ili 0 u zavisnosti od zatečenog stanja na pinu.

1.1.3 Upis u LAT registar

Pored PIC18F serije, *Microchip* je proslavila i PIC16F serija. Interesantno je da je *LAT* registar primećen kao nedostatak u PIC16F seriji, pa je ubačen u PIC18F seriju. U PIC16F seriji upis i očitavanje sa pina se vrši isključivo korišćenjem *PORT* registra. Naime, PIC serija 8 bitnih kontrolera koristi tzv. ***RMW (Read Modify Write) sekvencu*** kada menja stanje na izlaznom pinu (sa 0 na 1). Na slici 6 prikazana je ilustracija postupka vršenja RMW operacija. Naime, kada se na nekom

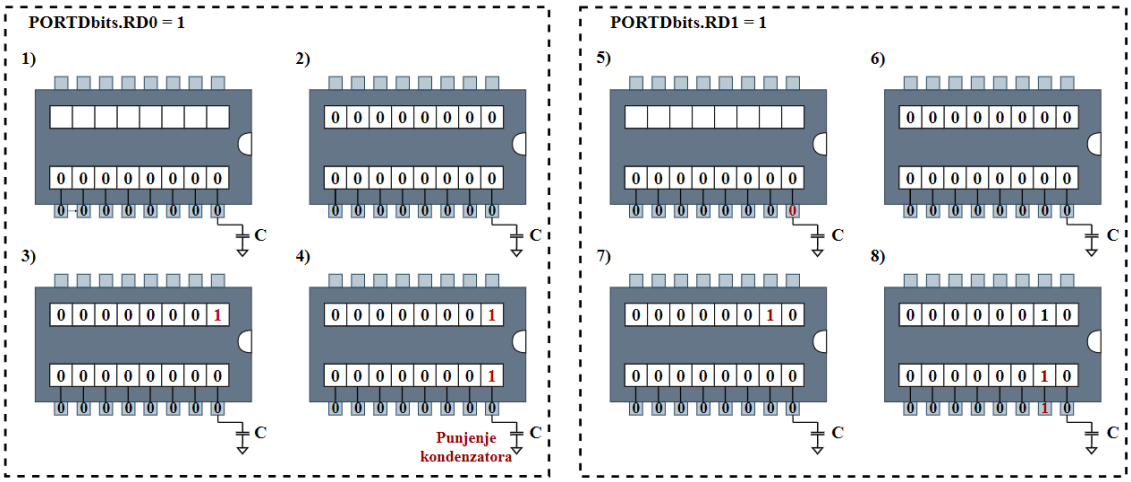
pinu promeni vrednost, mikrokontroler prvo očita vrednost svih 8 bita pripadajućeg PORT registra (slika 6 a)), ta vrednost se prosleđuje mikroprocesoru i smešta u određeni interni registar (slika 6 b)). Mikroprocesor izvršava željenu modifikaciju vrednosti u internom registru (slika 6 c)), a zatim tako modifikovanu vrednost upisuje nazad u pripadajući PORT registar (slika 6 d)).



Slika 6: Izvršavanje RMW operacija

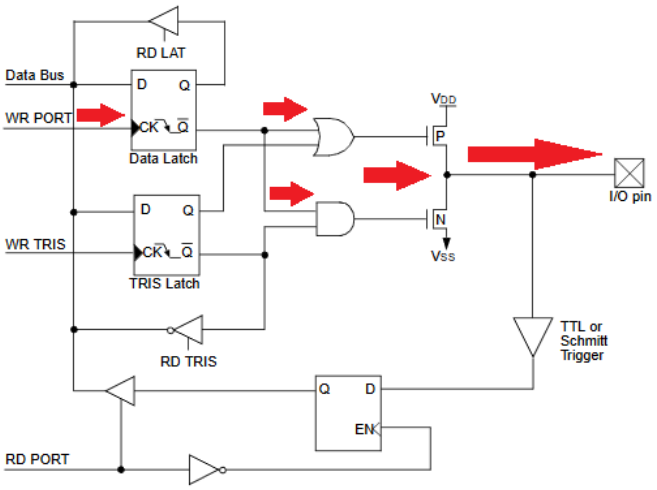
Prethodno opisan postupak izmene stanja na pinu određenog porta može dovesti do neočekivanog ponašanja izlaznog pina, kako operacija očitavanja PORT registra podrazumeva očitavanje stanja na fizičkim pinovima, odnosno, naponskih nivoa na pinovima. Takozvani RMW problem nastaje kada se želi izvršiti više sukcesivnih izmena stanja na odgovarajućem portu jer se očitavanje stanja vrši direktno sa pina. Ako je port opterećen kapacitivno i recimo pin se želi sa logičke 0 postaviti na logičku 1, zbog karakteristike kondenzatora da ne dozvoljava nagle promene napona, pin će neko vreme ostati u stanju logičke 0. Pošto je mikrokontroler znatno brži od ovog prelaznog procesa, dešava se da naredna promena nekog drugog pina na istom portu započinje RMW sekvencu očitavanjem porta sa neustaljenim stanjem. Ovo uzrokuje da nakon *modify* i *write* sekvence biva zanemareno prethodno postavljeno stanje na pinu.

Pretpostavimo da su unete sledeće dve linije koda: PORTDbits.RD0 = 1 i PORTDbits.RD1 = 1. Takođe, pretpostavimo da su inicijalna stanja svih pinova na PORTD logičke 0 i da je pin RD0 kapacitivno opterećen. Kada mikrokontroler izvrši PORTDbits.RD0 = 1 , nakon *RMW* sekvence na PORTD se očekuje 0x01 stanje, ali se dobija 0x00 zbog kapacitivnih karakteristika opterećenja. Ako se odmah zatim pozove PORTDbits.RD1 = 1, nakon *read* komande mikroprocesoru će biti prosleđena vrednost 0x00 umesto 0x01. Što će usloviti da stanje na pinu nakon *modify* i *write* bude 0x02, a očekivana vrednost je 0x03. Prethodno opisan problem ilustrovan je na slici 7.



Slika 7: RMW problem

Ovaj problem je rešavan vremenskom zadrškom, ali to svakako nije efikasno rešenje. Efikasnije rešenje je dodavanjem D flip flopa, kojim se reguliše stanje na izlaznom pinu.



Slika 8: Upis na LATx registra

Ovaj D flip flop pamti poslednju upisanu vrednost bez uticaja opterećenja pina na njegovo logičko stanje. Na ovaj način očitavanjem LAT registra (taktovanjem *RD LAT* bafera) očitava se poslednje upisano stanje u *Data Latch*. Iako na pinu neće biti promene neko vreme zbog osobina potrošača, stanje na izlazu *Q Data Latch* ostaje isto kao željeno. Na ovaj način ne dolazi do problema sa *RMW* sekvencom. Na slici 8 je dat prikaz toka podataka tokom *RMW* sekvence.

Da bi upis u LAT registar imao smisla neophodno je pin proglasiti za izlazni. Ako je pin ulazni onda su izlazi *I* i *ILI* logičkih kola postavljeni na stanje takvo da oba MOSFET tranzistora budu isključena. Ako je pin izlazni onda će u zavistnosti od stanja na $\overline{Q}$ biti uključen PMOS u slučaju 0 (na izlaz se prosleđuje V_{DD} odn. logička

1) i NMOS u slučaju 1 (na izlaz se prosleđuje V_{SS} odn. logička 0). Treba uzeti u obzir da je stanje na $\overline{Q}$ *Data Latch* komplementarno sa stanjem upisanim kroz *Data Bus*. Ovo rezultuje da će za upisano stanje logičke 0 na D ulazu *Data Latch*, na izlazu *I* kola uključuje NMOS pa je stanje na pinu logička 0. Za upisano stanje logičke 1 na D ulazu *Data Latch*, na izlazu *ILI* kola uključuje PMOS pa je stanje na pinu logička 1. Upis vrednosti na pin se vrši upisom u LATx registar. Raspored pinova u LATB registru dat je na slici 9.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LATB	LATB7	LATB6	LATB5	LATB4	LATB3	LATB2	LATB1	LATB0

Slika 9: Raspored bitova u LATB registru

Ako se želi upisati digitalna vrednost na odgovarajući port, neophodno je upisati vrednost u LATx, npr. LATE = 0xAA. Ako se želi postaviti digitalno stanje na pinu RF3, onda je to moguće odraditi upisom ili LATFbits.LATF3 ili LATFbits.LF3 koji će upisati vrednost 1 ili 0 u *Data Latch* a samim tim i uključiti odgovarajući MOSFET, čime se postavlja željeno logičko stanje na izlazni pin mikrokontrolera.

2 Primeri programa za kontrolu i upis na I/O pinove

Softver napisan za mikrokontroler naziva se **firmver**, on zapravo predstavlja spregu između hardvera i softvera. U ovim vežbama, za implementiranje firmvera, koristiće se razvojno okruženje *MPLAB* kompanije Microchip. Uz MPLAB razvojno okruženje, u svrhe kompajliranja koda, koristiće se MPLAB XC8 kompajler, namenjen za 8-bitne PIC mikrokontrolere.

Struktura svakog programa sastoji se iz dva dela (listing 1):

- Inicijalizacije - izvršava se pre početka programa i uloga je da se hardver pravilno konfiguriše kako bi se izvršile sve akcije.
- Beskonačne petlje - koja predstavlja akcije koje mikrokontroler treba da izvrši.

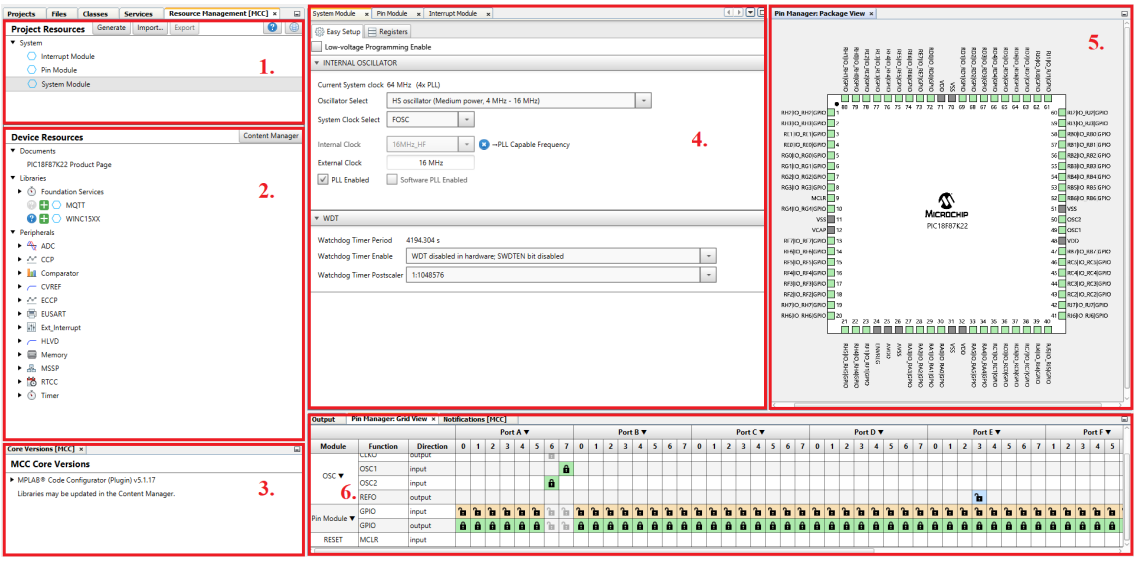
Proces koji mikrokontroler nadgleda ili kontroliše nikada ne treba da se završi, jer je mikrokontroler komponenta koja je predviđena da u svakom momentu radi. Iz tih razloga uvodi se beskonačna petlja.

Listing 1: Izled osnovne petlje

```
1 //Start
2 void main(void)
3 {
4     //Inicijalizacioni kod
5     SYSTEM_Initialize();
6
7     while (1)
8     {
```

```
 9            //Kod koji se izvrsava
10        }
11 }
```

U okviru MPLAB razvojnog okruženja moguće je dodati MCC (*MPLAB Code Configurator*) grafičko okruženje, čija je namena konfigurisanje određenih periferija i funkcionalnosti mikrokontrolera za specifičnu namenu. Pomoću ovog alata inicijalna podešavanja mikrokontrolera, kao i dodavanje odgovarajućih biblioteka je znatno olakšano. Na slici 10 prikazan je izgled MCC okruženja, gde je moguće uočiti nekoliko funkcionalnih celina. Polje obeleženo brojem 1 na slici 10 odnosi se na module koje je moguće podešavati - *Pin Module*, *Interrupt Module* i *System Module*. Podešavanja svakog od navedenih modula vrše se u okviru polja obeleženog brojem 4, na slici 10. U okviru polja 2 prikazani su svi dostupni resursi određenog mikrokontrolera, u našem slučaju PIC18F87K22, dok je sam mikrokontroler prikazan u okviru polja 5. Podešavanja pinova mikrokontrolera moguće je vršiti u delu Code Configuratora obeleženog brojem 6, u odeljku *Pin Manager: Grid View* ili u okviru polja 5 (*Pin Manager: Package View*), desnim klikom na pin. Deo slike 10 označen brojem 3 odnosi se na prikaz informacija o verziji MCC-a.



Slika 10: MPLAB Code Configurator

Na osnovu podešavanja mikrokontrolera u MCC-u generiše se odgovarajući kod u C programskom jeziku, koji sadrži sve neophodne funkcije za konfigurisanje mikrokontrolera, a u njega je dalje potrebno dodati korisnički kod. U okviru listinga 1 moguće je uočiti funkciju *SYSTEM_Initialize*, koja vrši inicijalizaciju celokupnog sistema (pinova, oscilatora, periferija mikrokontrolera itd.) na način koji je definisan kroz podešavanja u okviru MCC-a.

Za svaki od predstojećih primera bilo je potrebno izvršiti određena podešavanja u okviru MCC-a. Naime, u okviru *System Module* polja potrebno je podesiti oscilator, koji je neophodan za normalno funkcionisanje mikrokontrolera. Kako bi bio omogućen rad sa spoljašnjim oscilatorom, koji se nalazi na razvojnoj ploči, odabran je HS (*High Speed*) oscilator (4 MHz do 16 MHz), gde je frekvencija spoljašnjeg

oscilatora postavljena na 16 MHz. Sistemski takt dovodi se sa datog spoljašnjeg oscilatora, stoga se kao *System Clock Select* postavlja *FOSC*. Dodatno, kako bi postigli sistemski takt frekvencije 64 MHz, potrebno je omogućiti PLL (*Phase Locked Loop*) množitelj frekvencije. U okviru *Pin Manager: Grid View* polja neophodno je podesiti određene pinove mikrokontrolera kao ulazne, odnosno, izlazne, klikom na ikonicu katanca u odgovarajućem polju. Ova podešavanja je takođe moguće izvršiti u okviru polja *Pin Manager: Package View* - desnim klikom na pin i odabirom *input* ili *output* opcije. Klikom na taster *Generate*, u odeljku *Project Resources* konfiguriše se odgovarajući kod na osnovu izvršenih podešavanja u MCC-u.

Primer 1: *LED blinking - port (1. način)*

U datom primeru se žele pinovi porta B postavljati naizmenično na visoko, odn. nisko stanje. Pre izvršavanja neophodno je inicijalizovati port B kao izlazni, podešavanjem pinova u okviru MCC-a, odnosno, upisom vrednosti 0x00 u TRISB registar. Jedan od mogućih načina za implementaciju programa za uključivanje i isključivanje porta dat je u listingu 2. Na početku je potrebno postaviti vrednosti svih pinova porta B na logičku 0, upisom vrednosti 0x00 u LATB registar. Tada su pripadajuće LED porta B isključene. Zatim se generiše pauza od 1 s, upotrebom funkcije *__delay_ms*. Funkcija *__delay_ms* je korisna kada se želi napraviti pauza određeno vreme. To je realizovano izvršavanjem asemblerske *NOP* (*No OPeration*) funkcije određeni broj puta. Problem je što mikrokontroler poseduje mikroprocesor sa jednim jezgrom, pa će svako pozivanje funkcije *__delay_ms* praktično blokirati izvršavanje koda. Ovo se prevazilazi složenijim mehanizmima, o kojima će biti reči na nekim od narednih vežbi. U sledećem koraku u LATB registar se upisuje vrednost 0xFF, što znači da se na svaki od pinova porta B dovodi logička 1 (LED su uključene), nakon čega sledi još jedna pauza od 1 s.

Listing 2: LED blinking - port

```
1 void main(void)
2 {
3     SYSTEM_Initialize();
4
5     while (1)
6     {
7         LATB = 0x00;
8         __delay_ms(1000);
9
10        LATB = 0xFF;
11        __delay_ms(1000);
12    }
13 }
```

Primer 2: *LED blinking - port (2. način)*

Prethodni primer moguće je preglednije napisati korišćenjem bitskog (eng. *bitwise*) operatora za negaciju bita $\sim$, a odgovarajući kod dat je u okviru listinga 3. Jednostavnim negiranjem svih bita LATB registra vrši se naizmenično invertovanje stanja na pinovima porta B.

Listing 3: LED blinking - port

```
1 void main(void)
2 {
3     SYSTEM_Initialize();
4
5     while (1)
6     {
7         LATB = ~LATB;
8         __delay_ms(1000);
9     }
10
11 }
```

Primer 3. *LED blinking - pin*

Program za uključivanje i isključivanje pina pripadajućeg porta prikazan je u listingu 4. Za pristup pojedinačnim pinovima porta koristi se LATxbits struktura, gde x označava određeni port, i operator tačke (”.”). Pomoću operatora tačke (”.”) nakon čega se prosleđuje LATx0, LATx1, LATx2, LATx3, LATx4, LATx5, LATx6 ili LATx7 (Lx0, Lx1, Lx2, Lx3, Lx4, Lx5, Lx6 ili Lx7), moguće je pristupiti bilo kojem bitu odgovarajućeg LAT registra. U ovom slučaju pristupa se bitu na drugom mestu na portu A (RA1 pin). Kako bi odgovarajuća LED bila isključena drugom bitu LATA registra dodeljuje se vrednost logičke 0, zatim sledi pauza od 1 s, te uključivanje LED postavljanjem vrednosti bita na logičku 1, a na kraju se generiše još jedna pauza od 1 s.

Listing 4: LED blinking - pin

```
1 void main(void)
2 {
3     SYSTEM_Initialize();
4
5     while (1)
6     {
7         LATAbits.LATA1 = 0;
8         __delay_ms(1000);
9
10        LATAbits.LATA1 = 1;
11        __delay_ms(1000);
12    }
13 }
```

Primer 4. *LED blinking - binarno brojanje*

Dati primer implementira binarno brojanje koje je pri tome vizualizovano odgovarajućim uključivanjem i isključivanjem LED određenih portova. Program za uključivanje i isključivanje LED portova po principu binarnog brojanja prikazan je u listingu 5. Promenljive *counter* i *counter2* predstavljaju osmobitne promenljive unsigned int tipa, čije su vrednosti inicijalno postavljene na 0. Na portu B i C paralelno se prikazuje postfiksno i prefiksno inkrementiranje, redom. Kod se izvršava sa leva na desno, stoga će u LATB registar prvo biti upisana vrednost *counter* promenljive, koja će zatim biti inkrementirana, dok će u LATC registar biti upisana inkrementirana vrednost *counter2* promenljive.

Listing 5: LED blinking - binarno brojanje

```
1 void main(void)
2 {
3     SYSTEM_Initialize();
4
5     uint8_t counter = 0, counter2 = 0;
6
7     while (1)
8     {
9         LATB = counter++;
10        LATC = ++counter2;
11        __delay_ms(1000);
12    }
13 }
```

Primer 5. *LED blinking - zavesa*

U ovom primeru uključivanjem i isključivanjem LED svih portova simulirano je spuštanje zavese. Odgovarajući kod prikazan je u okviru listinga 6. Da bi kod iz listinga 6 bio razumljiviji neophodno je razumeti pojam maske, odn. maskiranja bita.

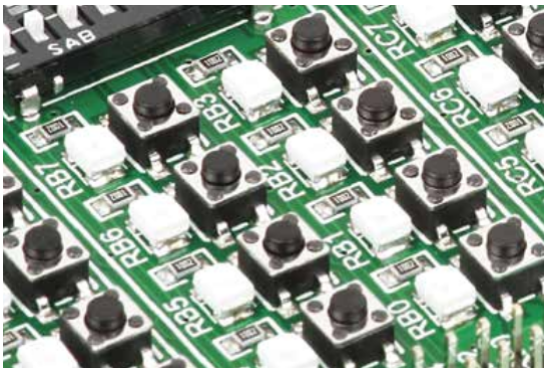
Maskiranje bita se najčešće primenjuje kada se ne zna trenutno stanje registra, a želi se izmeniti vrednost samo jednog bita. Npr. pretpostavimo da je napravljena osmobitna promenljiva *abcd efgh*, gde je svaka vrednost bita u ovom bajtu predstavljena jednim slovom. Ako se želi promeniti vrednost bita *e* na vrednost 1, neophodno je koristiti bitsko ili (u C-u operator |). Zatim je neophodno napraviti odgovarajuću masku, tj. treba napraviti konstantu identične dužine gde se svi bitovi postavljaju na 0, izuzev bita koji se želi postaviti na 1. U ovom slučaju za promenljivu *abcd efgh* odgovarajuća maska je *0x08* (*0b0000 1000*), pa je rezultat operacije *abcd efgh |0x08 = abcd 1fgh*. Ovo za rezultat ima da su svi bitovi nepromenjeni izuzev bita na poziciji *e*. Ako se želi neki od bitova podesiti na *0* neophodno je koristiti bitsko i (u C-u operator &) i primeniti masku gde se svi bitovi postavljaju na 1 izuzev bita koji se želi promeniti. Ako se želi bit d promeniti na 0 onda je neophodno korisiti masku 0xEF (0b1110 1111), pa bi konačni izraz za postavljanje bita d na 0 glasio *abcd efgh & 0xEF = abc0 efgh*.

Česta je greška da se mešaju logičko i (&&) sa bitksim i (&), kao i logičkog ili (‖) sa bitskim ili (|). U C-u nenulta vrednost se tretira kao logičko tačno dok nula predstavlja logičko netačno. Npr. Rezultat izraza *0x0F & 0xF0* je 0x00, dok je rezultat iste te operacije *0x0F && 0xF0* jednak logički tačnom iskazu (vrednost različita od nule).

Pored pomenutih operatora koriste se još i ˆ za ekskluzivno ili, >> za pomeranje bita u desno za određeni broj mesta kao i << za pomereanje određeni broj mesta u levo.

U listingu 6 uključena je biblioteka *stdint*. Ova biblioteka je korisna jer se uz pomoć nje mogu veoma lako deklarisati promenljive različitih veličina bez preteranog razmišljanja o veličini određenog tipa. Ime tipa se formira rečju *int*, zatim sledi dužina u bitima i na kraju *_t*. Ako se želi neoznačeni tip neophodno je dodati *u* na početak. Npr. ako se želi napraviti neoznačena celobrojna osmobitna promenljiva, korišćeni tip podataka bi bio *uint8_t*. Ova biblioteka je korisna jer čini kod lako prenosivim na druge arhitekture bez bilo kakvih izmena u deklaraciji promenljivih

zato što u nekim sistemima int može biti 16 bita, dok na drugim sistemima može biti 32 bita što može uvesti sistem u nestabilno stanje. Zatim je deklarisana funkcija *reset_ports*. Ova funkcija nema povratnih vrednosti, dok ključna reč *static* označava da će ova funkcija biti vidljiva samo unutar ovog fajla. Svaka funkcija napisana u C-u je globalna, a ponekad je dobro da ona bude vidljiva samo iz fajla gde je ona deklarisana i u tim slučajevima se koristi ključna reč *static*. Pravilo je da funkcije koje neće biti korišćene izvan izvornog (*source*) fajla budu deklarisane kao statičke. Uloga funkcije *reset_ports* je da sve bitove, svih portova podesi na vrednost 0. U *main* funkciji u inicijalizacionom delu napravljene su dve osmobitne promenljive *counter* i *value*. Treba uočiti da je promenljiva *value* konstantna, a ona se proglašava dopisivanjem ključne reči *const* pre definisanja tipa promenljive. U mikrokontrolerskim sistemima je veoma važna ušteda memorijskog prostora, pre svega RAM memorije. Ako se neka promenljiva deklariše kao konstantna (njena vrednost neće biti promenjena), kompajler će je smestiti u flash memoriju, čime ona neće biti upisana u RAM. Naizgled ovde je mala ušteda prostora, ali prilikom rada sa većim brojem podataka (npr. slike) na ovaj način se može uštedeti dosta prostora. Na slici 11, dat je prikaz porta na koji su povezane LED diode koje signaliziraju stanje bita na portu.



Slika 11: Izgled LED dioda na portu

Važno je uočiti da su diode *Rx0* i *Rx4* (gde x menja slovnu oznaku porta), čine par LED dioda najbližih ivici ploče. Kako bi se te dve diode aktivirale neophodno je u *LATx* upisati vrednost *0x11*, pa se iz tih razloga vrednost promenljive *value* postavlja na *0x11*. Nakon inicijalizacije svih portova kao izlaznih izuzev porta A, vrši se postavljanje svih portova na nizak nivo (funkcija *reset_ports*). Nakon inicijalizacije u beskonačnoj petlji se izvršava for petlja u 4 koraka i to korak za svaki od par dioda:

1. $Rx0$ i $Rx4$
2. $Rx1$ i $Rx5$
3. $Rx2$ i $Rx6$
4. $Rx3$ i $Rx7$

Unutar *for* petlje razlikuju se dve vrste izraza x = *value* << *counter* kao i x |= *value* << *counter* gde x odgovara vrednosti pripadajućeg *LAT* registra. Promenljiva *counter* uzima vrednosti iz skupa $x \in \{0,1,2,3\}$, pa će pomerene vrednosti promenljive *value* (*counter* puta) imati vrednosti *0x11*, *0x22*, *0x44*, *0x88* redom za vrednosti iz skupa x. Na ovaj način se postiže da se uključuje par susednih LED dioda. Drugi izraz koristi maske *0x11*, *0x22*, *0x44*, *0x88* redom za svaki korak, pa se na taj način ne gase prethodno uključene LED diode pa se samo pridodaju novo uključene diode. Ovaj način uključivanja dioda se može posmatrati kao efekat spuštanja zavese. Između svaka dva koraka vrši se pauza u trajanju od 1 *s*. Nakon izvršenja *for* petlje vrši se resetovanje svih *LAT* registara, a zatim se ponovo ulazi u *for* petlju.

Listing 6: LED blinking - zavesa

```
#include <stdint.h>

static void reset_ports(){
    LATA = 0x00;
    LATB = 0x00;
    LATC = 0x00;
    LATD = 0x00;
    LATE = 0x00;
    LATF = 0x00;
    LATG = 0x00;
    LATH = 0x00;
    LATJ = 0x00;
}

void main(void)
{
    SYSTEM_Initialize();

    uint8_t counter;
    const uint8_t value = 0x11;
    reset_ports();
    while (1)
    {
        for(counter = 0; counter<4; counter++){
            LATA = LATA | (value << counter);
            LATB = LATB | (value << counter);
            LATC = LATC | (value << counter);
            LATD = LATD | (value << counter);
            LATE = LATE | (value << counter);
            LATF = value << counter;
            LATG = value << counter;
            LATH = value << counter;
            LATJ = value << counter;
            __delay_ms(1000);
        }
        reset_ports();
        __delay_ms(1000);
    }
}
```

Dodatne napomene:

- MCU datasheet `http://ww1.microchip.com/downloads/en/devicedoc/39960d.pdf`.
- EasyPIC PRO v7 user manual `https://download.mikroe.com/documents/full-featured-boards/easy/easypic-pro-v7/easypic-pro-v7-manual-v101.pdf`