SPI (*Serial Peripheral Interface*) komunikacioni protokol

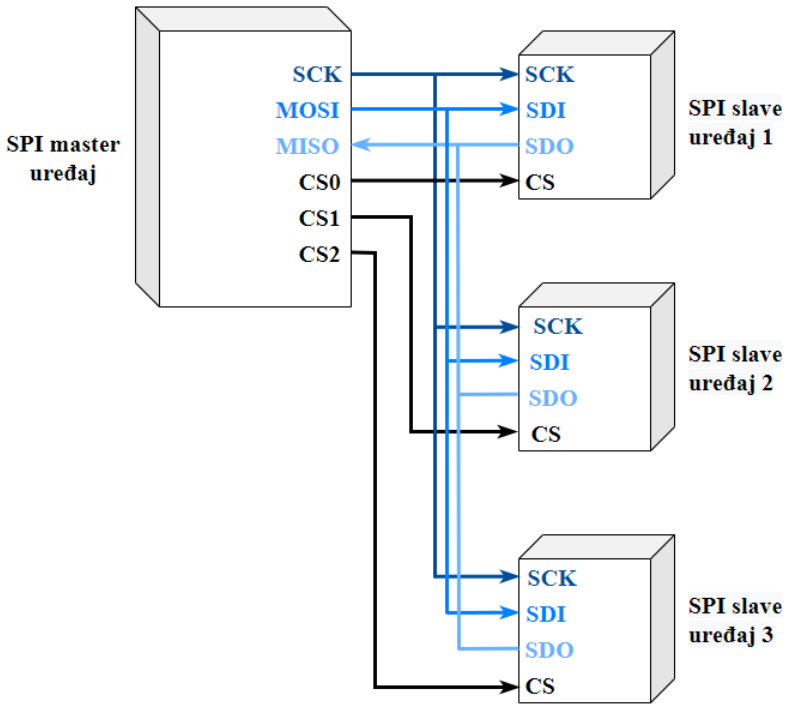
1 SPI komunikacija

SPI (*Serial Peripheral Interface*) predstavlja jedan od najzastupljenijih komunikacionih protokola sinhrone serijske komunikacije, a razvila ga je kompanija *Motorola* 80-ih godina 20. veka. SPI komunikacioni protokol najčešće se koristi pri komunikaciji mikrokontrolera sa periferijama u vidu senzora, Flash memorija, ADC, modula sa SD karticama i sl. Prenos podataka kod SPI komunikacije odvija se po principu full-duplex komunikacije, dakle, u svakom trenutku moguć je bidirekcioni prenos podataka. Kako je SPI sinhrona komunikacija, neophodno je postojanje signala takta kojim se postiže sinhronizacija slanja podataka i odabiranja primljenih podataka.

Uređaji koji komuniciraju putem SPI interfejsa mogu biti *master/main* uređaj i *slave/subnode* uređaji, gde je master/main uređaj onaj uređaj koji upravlja signalom takta i šalje ili prima podatke sa slave/subnode uređaja. Pri SPI komunikaciji može biti više slave uređaja, dok je broj master uređaja ograničen na jedan.

SPI komunikacija realizuje se putem četiri linije:

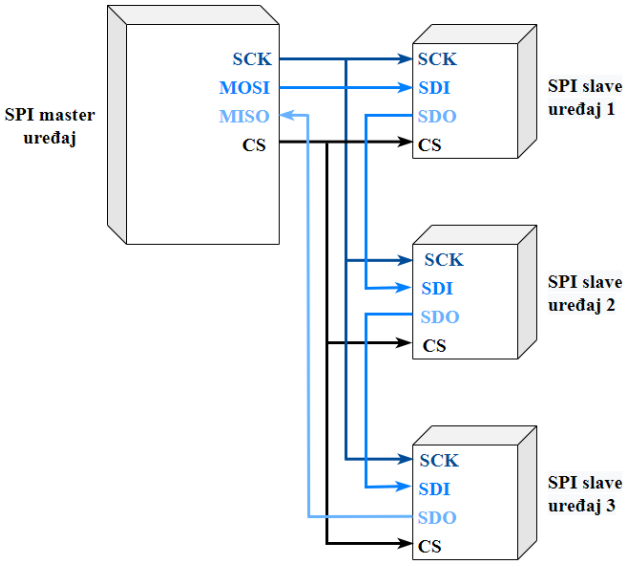
1. CS - *Chip Select* (SS - *Slave Select*)
2. SCK - *Serial Clock*
3. MISO - *Master Input Slave Output*
4. MOSI - *Master Output Slave Input*



Slika 1: Povezivanje master i slave uređaja pri SPI komunikaciji

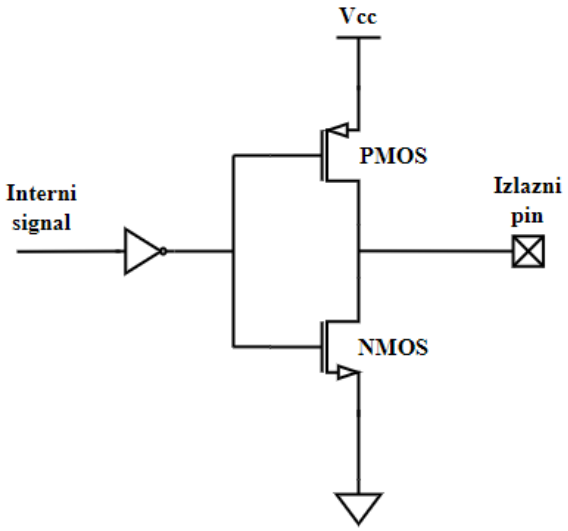
Na slici 1 ilustrovan je princpi povezivanja master i slave uređaja za SPI komunikaciju. SCK linija namenjena je za signal takta, kojim upravlja master uređaj. Razmena podataka između master i slave uređaja vrši se putem MISO i MOSI linija. MOSI linija označava liniju za prenos podataka od mastera ka slave uređaju, dakle, MOSI pin je izlazni pin za master uređaj i potrebno ga je povezati na ulazni pin slave uređaja, koji se još može obeležiti kao SDI (*Serial Data Input*). MISO linija prestavlja liniju putem koje slave uređaj šalje podatke na master uređaj, te je MISO pin ulazni pin master uređaja, a pin na predajnoj strani (slave uređaja) moguće je obeležiti kao SDO (*Serial Data Output*).

CS, odnosno SS, linija namnjena je za odabir slave uređaja sa kojim master uređaj želi da komunicira. Ova linija obično je aktivna na logičku nulu, dok postavljanjem CS linije na logičku jedinicu, master uređaj prekida komunikaciju sa određenim slave uređajem. Kada jedan master uređaj komunicira sa nekoliko slave uređaja, neophodno je onoliko CS linija koliko je slave uređaja, tako da kada master uređaj želi da započne komunikaciju sa određenim slave uređajem obori stanje na logičku nulu na onoj CS liniji na koju je dati slave uređaj povezan. Kako bi se u ovakvim sistemima izbegla upotreba velikog broja pinova master uređaja za CS linije, slave uređaje je moguće povezati u takozvanu *daisy-chain* konfiguraciju (slika 2). Daisy-chain konfiguracija podrazumeva serijsko povezivanje n slave uređaja i zajedničku CS liniju za sve slave uređaje. Slave uređaj koji je direktno povezan na master uređaj, preko svoje SDI linije prima podatke od master uređaja, a svaki preostali slave uređaj prima podatke sa njemu prethodnog slave uređaja u lancu. Dakle, SDO linija jednog slave uređaja vodi se na SDI liniju narednog slave uređaja u lancu. Slave uređaji izvršavaju komande koje primaju preko SDI linije samo kada se stanje CS linije menja sa aktivnog u neaktivno stanje. Sve dok je na CS liniji aktivno stanje, slave uređaj ne obarađuje podatke sa SDI linije, već ih propagira dalje, na svoju izlaznu SDO liniju. Na ovaj način, promenom stanja na zajedničkoj CS liniji, upravlja se izvršavanjem komandi na pojedinačnim slave uređajima u okviru dasiy lanca.



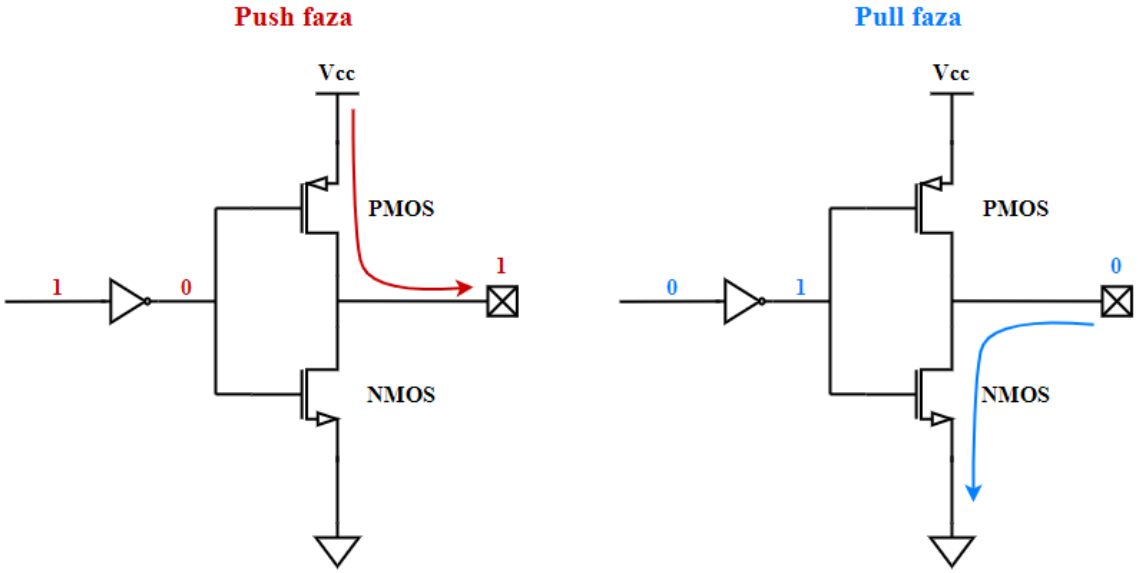
Slika 2: Daisy-chain konfiguracija

Linije za SPI komunikaciju izvode se u *push-pull* konfiguraciji. Push-pull konfiguracija jedna je od najzastupljenijih konfiguracija izlaznih pinova. Na slici 3 prikazan je izgled jednog izlaznog pina u push-pull konfiguraciji, koja podrazumeva upotrebu dva MOSFET tranzistora - PMOS i NMOS tipa, čiji je gate povezan u jednu tačku, i upravljanje stanjem na izlaznom pinu upotrebom određenog internog signala. Na ovaj način omogućena su dva logička nivoa na izlazu - visok naponski nivo (1) i nizak naponski nivo (0).



Slika 3: Push-pull konfiguracija izlaznog pina

Pri uspostavljanju logičkih nivoa na izlaznom pinu razlikuju se dve faze - *push* i *pull* faza (slika 4).



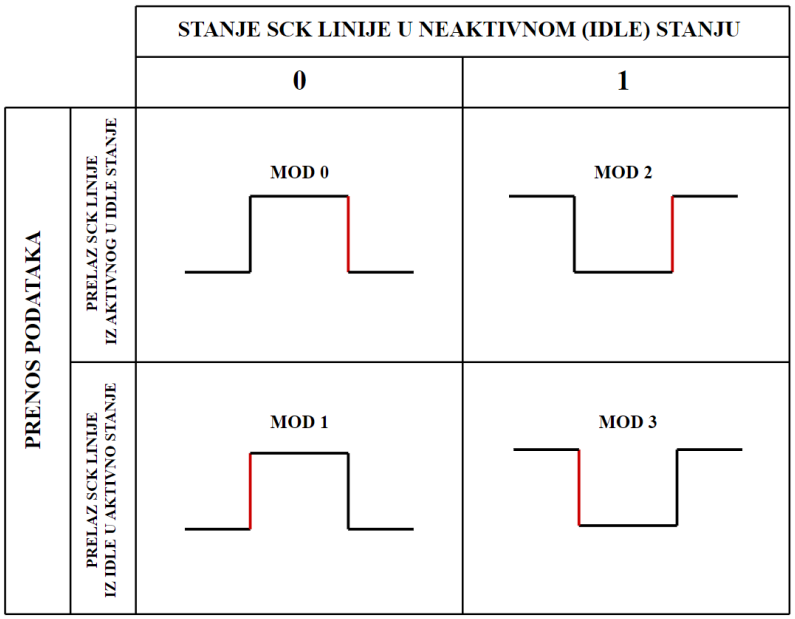
Slika 4: Push faza i pull faza

U push fazi, interni signal uz pomoć koga se upravlja izlazom ima vrednost logičke 1, te njegova invertovana vrednost (logička 0) koja se dovodi na gate MOSFET-a aktivira PMOS tranzistor. Na ovaj način se na izlaznom pinu uspostavlja visok naponski nivo, odnosno, stanje izlaznog pina je logička 1. U okviru pull faze interni signal ima vrednost logičke 0, a nakon invertora na gate MOSFET-a propagira se logička 1, što znači da je aktiviran NMOS. Aktivacijom NMOS tranzistora na izlazni pin dovodi se masa, odnosno, uspostavlja se logička 0.

Ova vrsta konfiguracije izlaznih pinova ne omogućava povezivanje više uređaja na jednu liniju i koristi se kod unidirekcionih linija.

2 Prenos podataka kod SPI komunikacije

Za sinhronu serijsku komunikaciju neophodan je adekvatan signal takta, koji u slučaju SPI komunikacije generiše master uređaj. Dakle, podešavanjem frekvencije signala takta, master uređaj upravlja brzinom prenosa podataka. Signal takta se sa master uređaja propagira dalje putem SCK, na sve slave uređaje koji su povezani sa njim. SCK signal ima dva parametra koja je moguće podešavati - polaritet i fazu. Polaritet signala takta definiše logičko stanje linije u neaktivnom stanje i on može biti *idle low* - u neaktivnom stanju SCK signal je na logičkoj 0 ili *idle high* - u neaktivnom stanju SCK signal je na logičkoj 1. Faza SCK signala definiše da li se odabiranje signala vrši pri promeni takta sa aktivnog na neaktivno stanje ili pri promeni signala takta sa neaktivnog na aktivno stanje. Dodatno, umesto faze signala takta i trenutaka odabiranja signala, moguće je definisati trenutke prenosa podataka - pri prelasku signala takta sa aktivnog na neaktivno stanje ili sa neaktivnog na aktivno stanje. U odnosu na kombinacije ovih parametara razlikuju se četiri moda SPI komunikacije, prikazana na slici 5.



Slika 5: Modovi SPI komunikacije

- Mod 0 - stanje SCK linije u neaktivnom stanju je 0, a prenos podataka se vrši pri promeni signala takta sa aktivnog (1) na neaktivno (0) stanje
- Mod 1 - stanje SCK linije u neaktivnom stanju je 0, a prenos podataka se vrši pri promeni signala takta sa neaktivnog (0) na aktivno (1) stanje
- Mod 2 - stanje SCK linije u neaktivnom stanju je 1, a prenos podataka se vrši pri promeni signala takta sa aktivnog (0) na neaktivno (1) stanje
- Mod 3 - stanje SCK linije u neaktivnom stanju je 1, a prenos podataka se vrši pri promeni signala takta sa neaktivnog (1) na aktivno (0) stanje

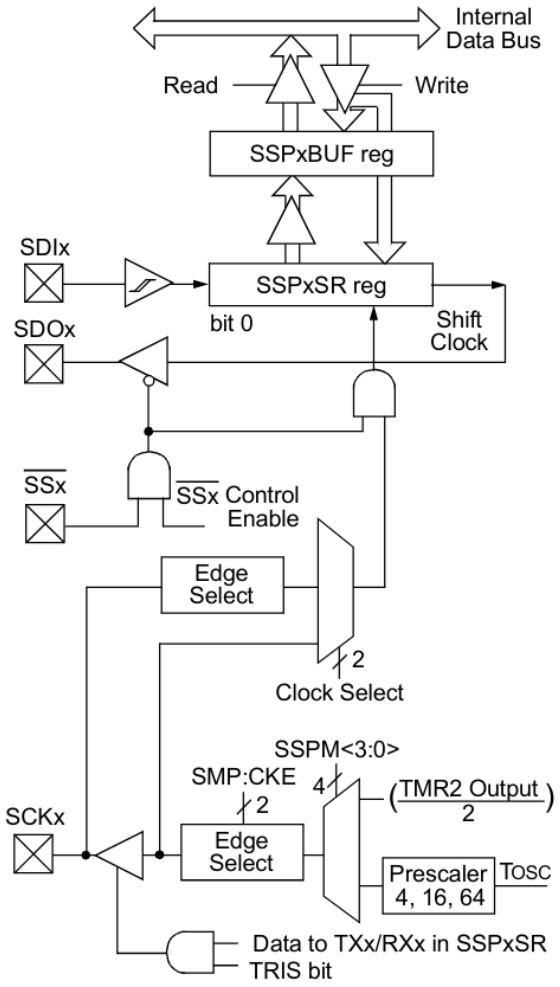
Početak razmene podataka po SPI protokolu inicira master uređaj, aktiviranjem odgovarajuće CS linije, za odabir slave uređaja sa kojim je potrebno izvršiti komunikaciju. CS linija je obično aktivna na logičku 0, dakle, kada master uređaj obori stanje određene CS linije na 0, započinje komunikaciju sa određenim slave uređajem. Master uređaj tada može da šalje podatke na slave uređaj, bit po bit, počevši od MSB, putem MOSI linije, ili da očitava podatke sa slave uređaja, bit po bit, putem MISO linije. Prenos i odabiranje primljenih podataka sinhronizovani su signalom takta.

Neke od prednosti SPI komunikacije jesu veća brzina prenosa podataka, jednostavan način selektovanja slave uređaja, kontinualan prenos bitova, bez generisanja start ili stop bita, full-duplex komunikacija, dok se mane SPI komunikacije ogledaju u upotrebi četiri linije i nedostatku mogućnosti provere prijema podataka i provere nastanka greške.

3 SPI modul PIC18F87K22 mikrokontrolera

Blok šema MSSP modula prilagođena za rad u SPI režimu prikazana je na slici 6.

Za rad sa SPI modulom neophodno je konfigurisati *SSPxCON1* i *SSPxSTAT* registre i razmenjivati podatke sa *SSPxSR* registrom posredstvom *SSPxBUF* baferskog registra. Uloga $\overline{SSx}$ (*Slave Select*) pina postoji samo u slave modu i služi za uključivanje i isključivanje komunikacije sa mikrokontrolerom. Uloga pina *SDIx* je prijem podataka u šift registar *SSPxSR* koji ima ulogu da primi jedan bajt podataka, poslatog od strane master ili slave uređaja. Nasuprot *SDIx* pinu, pin *SDOx* se koristi za slanje podataka ka master ili slave uređaju. Po prijemu bajta u *SSPxSR* registar primljeni podaci se smeštaju u prihvatni baferski registar *SSPxBUF* sa kojim korisnik razmenjuje podatke (isto važi i za slanje podataka). Statusnim bitovima *WCOL* i *SSPOV* u registru *SSPxCON1* moguće je detektovati koliziju prilikom slanja ili primanja podataka. Pin $SCKx$ na kom se generiše takt SPI komunikacije povezan je na multiplekser kojim se podešava izvor takta postavljanjem vrednosti u 4 bita $SSPM < 3:0 >$ *SSPxCON1* registar, tabela 1.



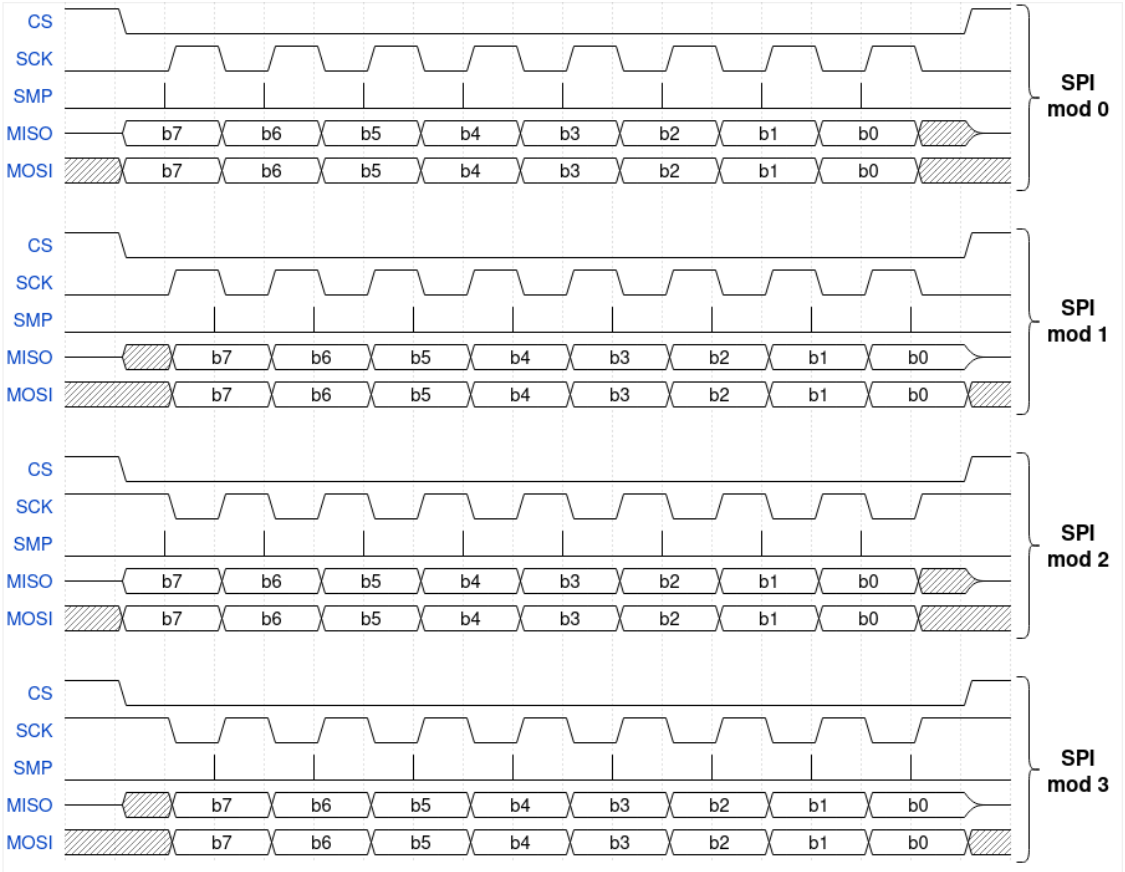
Slika 6: Blok šema SPI modula

$SSPM < 3:0 >$	Takt
0000	$F_{osc}/4$
1010	$F_{osc}/8$
0001	$F_{osc}/16$
0010	$F_{osc}/64$
0011	period TMR2/2
0101	Slave mod, *SSx* pin se ne koristi
0011	Slave mod, *SSx* pin se koristi

Tabela 1: Podešavanje takta prilagođenjem $SSPM < 3:0 >$ bita

Kako bi se upravljalo SPI modovima rada tj. kada će podaci biti poslati koriste se bitovi *CKE* u registru *SSPxSTAT* i *CKP* u registru *SSPxCON1*. Bit *CKP* kontroliše stanje taktnog signala u stanju pripravnosti, ako je vrednost na 0 onda je u stanju pripravnosti na liniji takta *SCKx* prisutan nizak naponski nivo, u suprotnom ako je na 1 onda je prisutan visok naponski nivo. Bit *CKE* podešava tranziciju naponskog

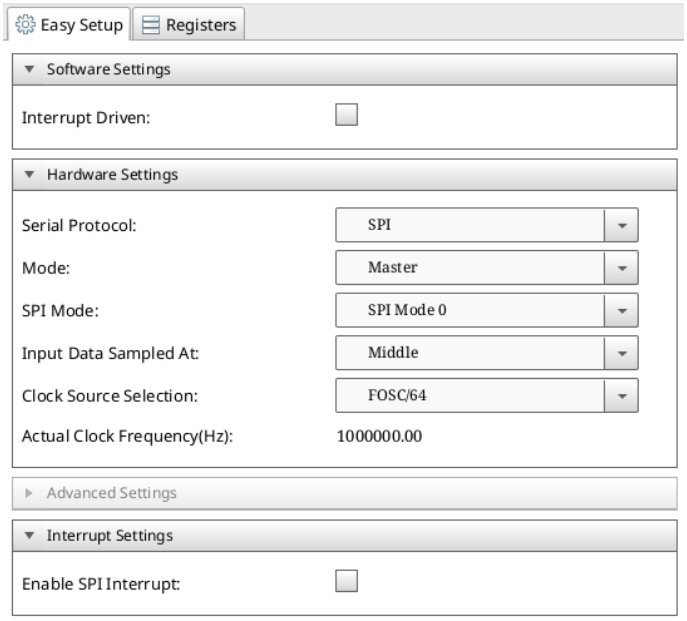
nivoa na koju će podatak biti poslat i ako je vrednost na 0 podaci se šalju na promenu iz aktivnog stanja (suprotno stanju pripravnosti definisanom *CKP* bitom) u stanje definisano stanjem pripravnosti (*CKP* bitom). Ako je bit *CKE* na 1 slanje se obavlja na promenu iz stanja definisanog stanjem pripravnosti u aktivno stanje. U većini mikrokontrolera koji se danas koriste, odabiranje ulaznog signala se vrši na ivicu signala suprotnu od ivice na koju se podaci šalju. Ovaj mikrokontroler podržava mogućnost podešavanja i kada će biti signal odabiran korišćenjem *SMP* bita u registru *SSPxSTAT* i to se može koristiti u master modu. Ako je bit *SMP* na 0 onda se odabiranje vrši kao i kod većine mikrokontrolere između dve ivice takta za slanje. Dakle ako se signali šalju na silazne ivice takta, podaci se odabiraju na uzlaznoj ivici takta i obrnuto. Ako je bit *SMP* na 1 onda se odabiranje vrši sinhrono sa taktom slanja. U svim primerima koji će biti realizovani *SMP* bit će biti podešen na podrazumevanu vrednost tj. 0. Na slici 7 prikazana su sva četiri moda rada SPI modula, gde je bit *SMP* podešen na 0, pa će se odabiranje vršiti između svake dve ivice takta za slanje.



Slika 7: Modovi rada SPI modula i pripadajući signali

4 Implementacija SPI modula u MPLAB-u

Kako bi se koristio SPI modul mikrokontrolera PIC18F87K22, neophodno je odabrati MSSP1 modul iz *Device Resources* → *Peripherals* u *Microchip Code Configurator* alatu. Nakon dodavanja modula dobija se prozor za podešavanje MSSP1 modula kao što je prikazano na slici 8.



Slika 8: Podešavanje SPI modula

Po odabiru MSSP1 modula neophodnoj specificirati korišćeni protokol odabirom *SPI* opcije iz padajućeg menija *Serial Protocol*. Tom prilikom se i dodeljuju funkcionalnosti *SCK1*, *SDI1* i *SDO1* na pinove *RC3*, *RC4* i *RC5*, redom. Kako bi se odabrao mod rada u polju *Mode* odabrana je opcija *Master* jer se očekuje da mikrokontroler bude u modu master uređaja. *SPI Mode* specificira mod rada SPI i u ovom slučaju je odabran mod 0 (*SPI Mode 0*), pa će *SCK1* linija u stanju pripravnosti biti na niskom logičkom nivou i podaci će se slati na svaku silaznu ivicu. Kako bi se podesilo odabiranje signala na polovini intervala između dve ivice slanja, opcija *Input Data Sampled At* se podešava na *Middle*, pa se odabiranje u ovom slučaju vrši na svaku uzlaznu ivicu. Funkcijom *Clock Source Selection* bira se izvor i frekvencija takta *SCK1*. Po odabiru opcije u polju *Actual Clock Frequency(Hz)* prikazana je trenutno podešena vrednost frekvencije. U ovom slučaju vrednost takta je 1 MHz jer je frekvencija oscilatora 64 MHz, a podešena vrednost frekvencije izvora takta je $\frac{F_{osc}}{64}$. Pošto se ne žele koristiti prekidi *Enable SPI Interrupt* i *Interrupt Driven* opcije nisu odabrane.

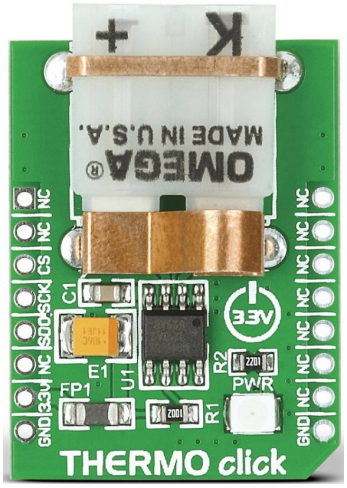
Nakon generisanja koda dobiju se fajlovi *spi1.h* i *spi1.c* sa implementacijama funkcija za manipulaciju nad SPI1 modulom. Funkcija *SPI1_Initialize* inicijalizuje modul i to podešavanjem *SSP1STAT* registra na 0x40 bit *CKE* se postavlja na 1 što je preduslov za korišćenje SPI moda 0 i bit *SMP* se postavlja na 0 kako bi se odabiranje vršilo pomereno za pola periode u odnosu na slanje signala. Registar *SSP1CON1* postavlja se na vrednost 0x02, čime se *CKP* pin postavlja na 0 odnosno da je stanje

pripravnosti pina za takt nisko stanje što je neophodno za podešavanje moda 0 i podešava se takt na $\frac{F_{osc}}{64}$. Zatim se podešava pin *RC3* kao izlazni tj. konfiguriše se *SCK1* pin kao izlazni. Na kraju se bit *SSPEN* u registru *SSP1CON1* postavlja na 0 čime se isključuje SPI1 modul. Pre svakog slanja podataka neophodno je pozvati funkciju *SPI1_Open* prosleđivanjem parametra (enumeracija *SPI1_DEFAULT*), čime se pored opisane inicijalizacije i uključuje SPI1 modul. Nasuprot pomenute funkcije, funkcija *SPI1_Close* isključuje SPI1 modul. Za slanje podataka koriste se funkcije *SPI1_WriteByte* i *SPI1_WriteBlock*. Funkcija *SPI1_WriteByte* šalje prosleđeni bajt podataka posredstvom SPI1 modula. Funkcija *SPI1_WriteBlock* prima kao parametar pokazivač na niz koji se želi proslediti, kao i broj elemenata koji se želi poslati. Za očitavanje podataka sa SPI1 modula koriste se funkcije *SPI1_ReadByte* i *SPI1_ReadBlock*. Funkcija *SPI1_ReadByte* očitava jedan bajt podataka pristiglih *SDI* linijom i vraća ga po završetku funkcije. Funkcija *SPI1_ReadBlock* prima pokazivač na niz u koji se smeštaju očitane vrednosti, kao i drugi parametar koji daje informaciju o broju podataka koji se trebaju primiti. Pošto će očitani podaci biti smešteni u prosleđeni niz, ova funkcija nema povratnih vrednosti. Četiri funkcije za čitanje i pisanje su blokirajuće funkcije, samim tim se SPI1 modul koristi samo u *half-duplex* modu rada. Kako bi se koristio *full-duplex* mod rada implementirane su funkcije *SPI1_ExchangeByte* i *SPI1_ExchangeBlock*. Funkcija *SPI1_ExchangeByte* prima jedan bajt koji šalje preko *SDO* linije, a zatim i očitava jedan bajt primljen peko *SDI* linije. Funkcija *SPI1_ExchangeBlock* se koristi za razmenu više podataka preko SPI1 linija, gde se kao prosleđeni parametar prenosi niz iz kojeg se prvobitno očitavaju podaci, a zatim se očitani podaci smeštaju u isti taj niz. Kao drugi parametar prenosi se podatak o broju pdoataka koji će biti razmenjeni.

5 Primeri

Primer 1: *THERMO click*

Thermo click predstavlja pretvarački modul koji vrednost napona na krajevima termopara K tipa, povezanog na odgovarajući konektor, pretvara u temperaturu. Glavna komponenta pretvarača je čip MAX31855K. Pločica sa pripadajućim konektorom, čipom i MikroBUS konektorom prikazana je na slici 9.



Slika 9: THERMO click

Komunikacija sa pretvaračem se odvija preko SPI protokola, gde se po spuštanju linije *CS* na nizak nivo, sa generisanim taktom u SPI modu 0 doprema četiri bajta podataka. Raspored podataka dat je na slici 10. Od bita 31 do bita 18 nalazi se podatak u o temperaturi vrućeg kraja termopara, bit 17 kao i bit 3 su rezervisani i uvek će se očitavati kao 0, bit 16 signalizira da li je došlo do greške (bit je na 1) i ako jeste onda će jedan od bitova B2, B1 i B0 biti na jedinici čime se signalizira da li jedan od provodnika termopara u kratkom spoju sa naponom napajanja, masom ili ne postoji ništa povezano na konektoru (otvorena veza). Takođe, pošto čip ima kompenzaciju hladnog spoja moguće je očitati i temperaturu na kojoj se nalazi čip tj. hladan spoj termopara, očitavanjem od bita 15 do bita 4.

14-bitni podatak o temperaturi termopara						12-bitni podatak o temperaturi hladnog kraja termopara							
B31	B30		B18	B17	B16	B15	B14		B4	B3	B2	B1	B0
Znak	2^{10} 1024 °C	...	2^{-2} 0,25 °C	Rezervisan	Greška	Znak	2^{6} 64 °C	...	2^{-4} 0,0625 °C	Rezervisan	Kratak spoj ka V_{cc}	Kratak spoj ka GND	Otvorena veza

Slika 10: Raspored podataka u primljenom bajtu

U listingu koda 1 dat je kod za očitavanje pretvarača MAX31855K. Pre komunikacije vrši se inicijalizacija periferija. Pošto je *Thermo click* povezan na konektor 1 razvojnog sistema, neophodno je *Chip Select* pin konfigurisati na pin *RE0*, gde mu je u ovom primeru dodeljena i znakovna konstanta CS. Važno je istaći da se korisnik mora starati o *CS* pinu tj. pre komunikacije mora odabrati slave uređaj sa kojim želi da razmenjuje podatke. Odabiranje se vrši pozivom funkcije *CS_SetLow* tj. spuštanjem pina CS na nizak naponski nivo. Nakon biranja slave uređaja poziva se funkcija *SPI1_Open* koja omogućuje komunikaciju preko SPI1 modula, nakon čega se vrši očitavanje 4 bajta koje prosleđuje MAX31855K pretvarač. Zatim se pozivom *SPI1_Close* zatvara komunikacija i pin se vraća na visoko stanje pozivom *CS_SetHigh* čime se čip deselektuje. Prva dva bajta sadrže podatak o temperaturi, a pošto je podatak o temperaturi 14-bitni neophodno ga je skladištiti u 16-bitnu

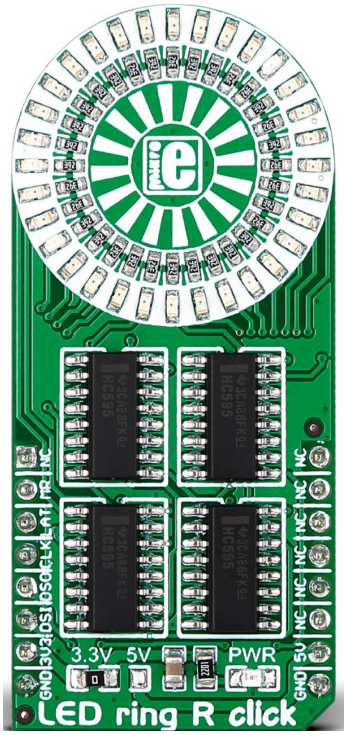
promenljivu. U promenljivu *termopar* se smešta ova 16-bitna vrednost, a zatim se proverava da li je najviši bit na jedinici maskom 0x8000, tj. da li je očitana temperatura vrućeg kraja termopara negativna i ako jeste ceo rezultat se množi sa -1. Pošto će u 16-bitnoj promenljivoj poslednja dva bita sadržati bitove B17 i B16 sa slike 10 tj. rezervisani bit i bit greške neophodno ih je odstraniti što je odrađeno desnim pomeranjem za dva mesta. Na sličan način se dobija 12-bitni podatak o temperaturi hladnog kraja s tim što se mora na kraju odstraniti 4 bita podataka i to B3 koji je rezervisan i B2, B1 i B0 koji prenose informaciju o grešci. Na kraju se podaci šalju preko UART protokola i nakon pauze od 500 ms se vrši ponovno očitavanje. Važno je naglasiti da se podaci o temperaturi hladnog kraja (ambijenta) množe sa vrednošću LSB bita 0,0625 C, dok se određivanje vrednosti vrućeg kraja množi sa 0,25 C.

Listing 1: Thermo click

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

void main(void)
{
    SYSTEM_Initialize();

    uint8_t data[4];
    int16_t termopar;
    int16_t ambijent;
    while (1)
    {
        CS_SetLow();
        SPI1_Open(SPI1_DEFAULT);
        SPI1_ReadBlock(data, 4);
        SPI1_Close();
        CS_SetHigh();

        termopar = (data[0]<<8) | data[1];
        if(termopar & 0x8000)
        {
            termopar *= -1;
        }
        termopar >>= 2;
        ambijent = data[2]<<8 | data[3];
        if(ambijent & 0x8000)
        {
            ambijent *= -1;
        }
        ambijent >>= 4;
        printf("Temperatura termopara: %.2f\n", termopar*0.25);
        printf("Temperatura ambijenta: %.2f\n", ambijent*0.0625);

        __delay_ms(500);
    }
}
```

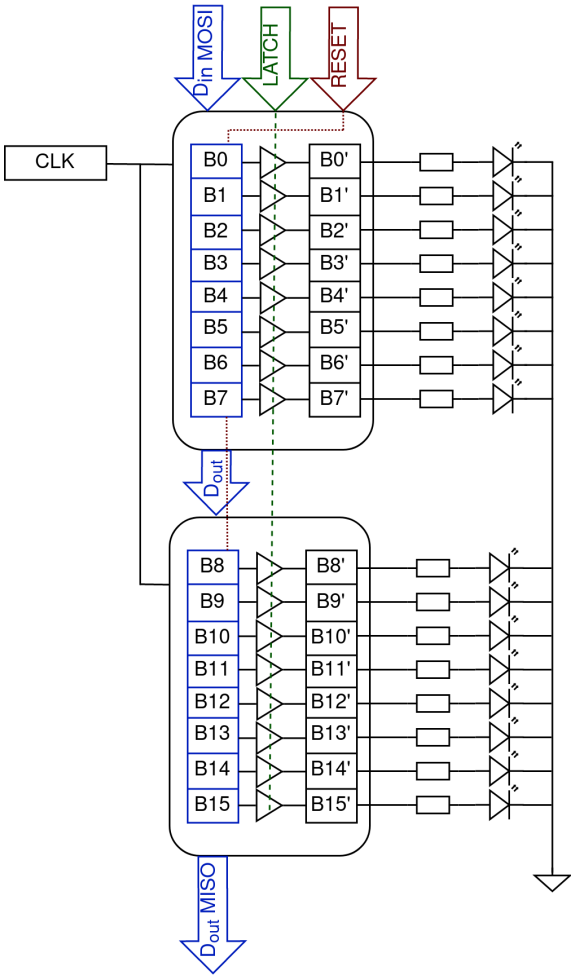
Primer 2: *LED Ring R click*

Na slici 11 prikazan je *LED ring R click* modul.



Slika 11: LED Ring R click

Pošto je SPI protokol kompatibilan sa radom pomeračkih (*shift*) registara moguće ga je korsititi i za upravljanje istim. *LED Ring R click* predstavlja modul sa četiri osmobitna pomeračka registra povezanih u *daisy chain* konfiguraciju. Na svaki od pinova ovih registara povezana je po jedna LED, čime je moguće nezavisno uključivati do 32 diode. Korišćeni su pomerački registri SN74HC595 koji ima mogućnost rada i na 3,3 V i na 5 V, podrazumevano se čipovi napajaju sa 3,3 V. Na slici 12 prikazano je povezivanje u *daisy chain* konfiguraciju dva pomeračka registra. Svaki od čipova sadrži bafere koji odvajaju pomerački registar od izlaznih. Korisnik uključivanjem bafera kontrolnom linijom *LATCH* kopira rezultat bita *Bn* u bit *Bn'* koji na svom izlazu ima povezanu LED. Važno je istaći da je *LATCH* linija aktivna na visoko stanje. U pomerački registar se upisuje tako što se postavi odgovarajuće stanje na D_{in} *MOSI* liniju, a zatim se izvrši tranzicija naponskog nivoa na *CLK* liniji. U slučaju sa slike 12 signali će biti upisani u pomerački registar tek nakon 16 perioda taktnog signala i prvo se postavlja MSB bit, a poslednji LSB bit. Ako se želi obrisati sadržina pomeračkih registara, da se ne bi upisivalo 16 nula u napravljeni 16-bitni registar, brisanje je omogućeno linijom *RESET*. Korisnik upisuje vrednosti u svih 16 bita i po završetku upisa *LATCH* pinom daje signal da se sadržina pomeračkog registra prekopira na izlazne pinove. Slična metodologija povezivanja primenjena je na modulu *LED ring R click*, samo su umesto dva povezana četiri pomeračka registra.



Slika 12: Pomerački registri u *daisy chain* konfiguraciji

U listingu koda 2 dat je kod za očitavanje AD konvertora i na osnovu očitavanja, proporcionalno ulaznom naponu, uključuju se LED na izlaznim pinovima pomeračkih registara. Nakon inicijalizacije *RST* pin se postavlja na visok nivo čime je kolo u aktivnom stanju tj. nije resetovano. Podešavanjem *LAT* pina na nizak nivo, omogućuje se upis u pomerački registar a da se pri tom ne utiče na izlazne pinove. Pošto je AD konvertor 12-bitni tj. opseg je od 0 do 4095, a vrednost koja se može smestiti u pomerački registar je od 0 do 31, opseg je preskaliran celobrojnim deljenjem sa 128 tj. pomeranjem 7 mesta u desno i dobijena vrednost se smešta u promenljivu *cnt*. Sadržina *cnt* promenljive predstavlja broj dioda koji se treba uključiti počevši od diode na mestu LSB-a. U liniji 18 *for* petlja generiše maske za svaku celobrojnu vrednost u opsegu od 0 do *cnt*, pa će npr. za vrednost u *cnt=5* pet najnižih bitova promenljive *led* biti na jedinici dok će preostali biti na nuli. Ovako generisanu 32-bitnu promenljivu neophodno je proslediti na izlaz pomeračkih registara. Kako bi se podaci pripremili za slanje neophodno je 32-bitnu promenljivu *led* konvertovati u niz od četiri osmobitna elementa korišćenjem funkcije memcpy. Pozivom funkcije *SPI1_WriteBlock* prenose se četiri bajta čime se upisuje vrednost u sva četiri pomeračka registra. Po završetku slanja, daje se kratak impuls na *LAT* pinu čime se upisane vrednosti iz pomeračkog registra prebacuju u izlazne registre.

Listing 2: LED Ring R click

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

void main(void)
{
    SYSTEM_Initialize();

    uint8_t data[4];
    uint8_t cnt;
    uint32_t led;
    RST_SetHigh();
    LAT_SetLow();
    while (1)
    {
        cnt = (uint8_t)(ADC_GetConversion(channel_AN0)>>7);

        led=0;
        for(uint8_t i=0; i <= cnt; i++)
        {
            led |= ((uint32_t)1 << i);
        }
        memcpy(data,&led,4);

        SPI1_Open(SPI1_DEFAULT);
        SPI1_WriteBlock(data, 4);
        SPI1_Close();
        LAT_SetHigh();
        LAT_SetLow();


        __delay_ms(500);
    }
}
```

Primer 3: *THERMO+LED Ring R click*

U listingu koda 3 dat je kod koji kombinuje prethodna dva primera. Prvo se vrši očitavanje četiri bajta iz kojih se izvlači podatak o temperaturi vrućeg kraja termopara. Kako bi promena od 1 C odgovarala jednoj uključenoj diodi, neophodno je iz vrednosti za temperaturu vrućeg kraja odstraniti podatak o delu iza decimalne tačke (definisanim sa dva bita). Zbog toga se očitana vrednost umesto pomeranja u desno za dva bita (rezervisani i bit o grešci), sada pomera četiri puta kako bi se odbacili i bitovi iza decimalne tačke (bitovi koji koduju 2^{-1} i 2^{-2}). Po očitavanju vrednosti i konverziji u celobrojnu vrednost, vrši se upis u pomerački registar i daje se signal za prebacivanje sadržaja iz pomeračkog u izlazni registar.

Listing 3: THERMO+LED Ring R click

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

void main(void)
{
    SYSTEM_Initialize();

    uint8_t data[4];
    uint8_t cnt;
    uint32_t led;
    int16_t termopar;
    RST_SetHigh();
    LAT_SetLow();
```

```
14      while (1)
15      {
16
17          CS_SetLow();
18          SPI1_Open(SPI1_DEFAULT);
19          SPI1_ReadBlock(data, 4);
20          SPI1_Close();
21          CS_SetHigh();
22
23          termopar = (data[0]<<8) | data[1];
24
25          cnt = (uint8_t)(termopar >> 4);
26
27          led=0;
28          for(uint8_t i=0; i < cnt; i++)
29          {
30              led |= ((uint32_t)1 << i);
31          }
32          memcpy(data,&led,4);
33
34          SPI1_Open(SPI1_DEFAULT);
35          SPI1_WriteBlock(data, 4);
36          SPI1_Close();
37          LAT_SetHigh();
38          LAT_SetLow();
39
40
41          __delay_ms(500);
42      }
43  }
```