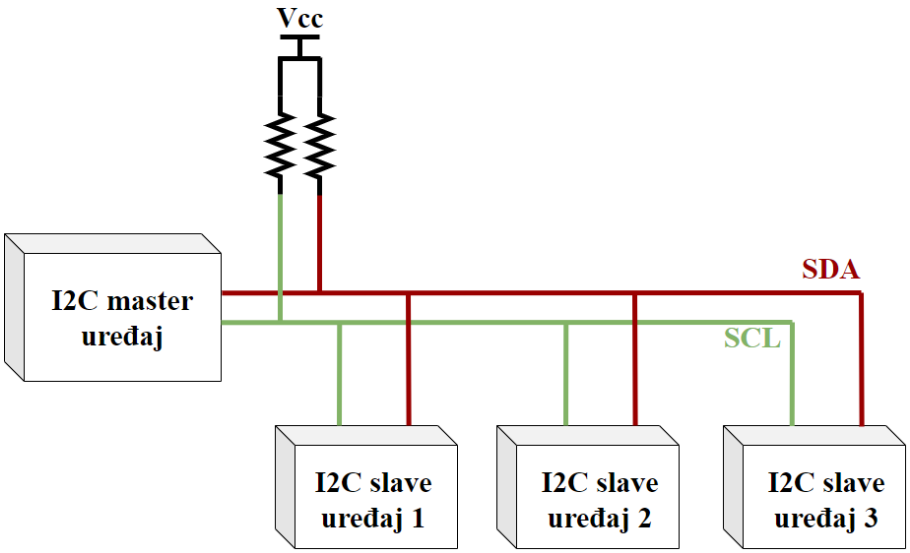
I^{2}C (*Inter-Integrated Circuit*) komunikacioni protokol

1 I^{2}C

I^{2}C (*Inter-Integrated Circuit*) komunikacioni protokol razvijen je 1982. godine od strane kompanije Philips Semiconductor i predstavlja jedan od komunikacionih protokola sinhrone serijske komunikacije. Danas na tržištu postoji veliki broj periferija kojima je osnovni komunikacioni interfejs upravo I^{2}C i ovaj protokol ima široku oblast primene u komunikaciji između mikrokontrolera, senzora, displeja, EEPROM (*Electrically Erasable Programmable Read-Only Memory*) itd. Razmena podataka kod I^{2}C komunikacije ostvaruje se putem I^{2}C magistrale, koju čine dve komunikacione linije – SCL (*Serial Clock*) linija za signal takta, i SDA (*Serial Data*) linija za podatke.

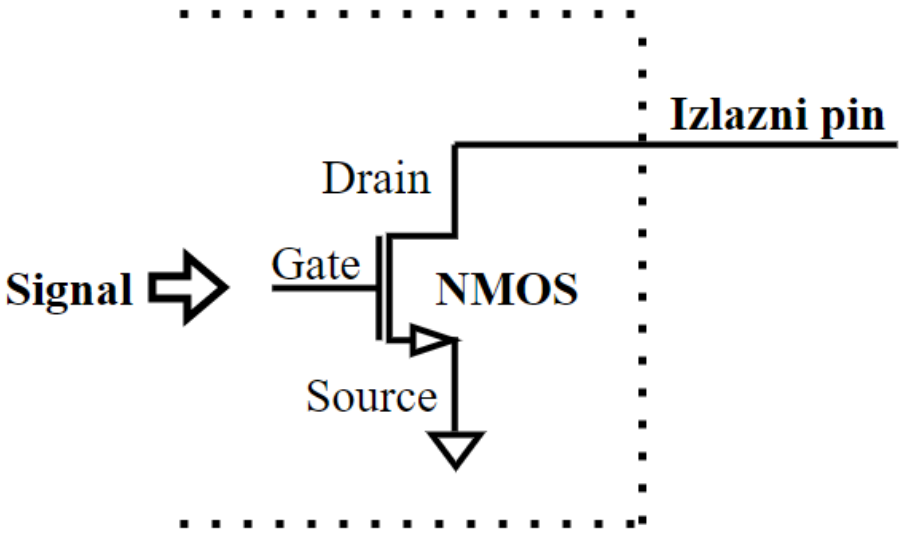


Slika 1: Povezivanje master i slave uređaja na linije I^{2}C magistrale

Za I^{2}C karakteristična je half-duplex komunikacija. Uređaji povezani na I^{2}C magistralu mogu biti master ili slave uređaji. Master predstavlja uređaj koji šalje komande drugim uređajima, dok je slave uređaj koji prima komande od mastera. Izlazni bitovi predajne strane su sinhronizovani sa bitovima koji se odbirkuju na prijemnoj strani uz pomoć signala takta koji je uvek kontrolisan od strane master uređaja. Na I^{2}C magistralu može biti povezano više master i slave uređaja istovremeno. Putem I^{2}C magistrale vrši se razmena podataka između master uređaja i tačno određenog slave uređaja, a kako ne postoji komunikaciona linija namenjena za adresiranje, potrebno je da pre slanja komandi master putem SDA linije pošalje adresu onog slave uređaja sa kojim treba da vrši razmenu podataka. Svaki slave povezan na I^{2}C magistralu mora da ima jedinstvenu adresu, stoga je broj slave uređaja koje je moguće povezati na magistralu ograničen brojem bita slave adrese. Ukoliko se u protokolu koriste n-bitne adrese maksimalan broj slave uređaja je 2^n. Broj master uređaja

teoretski može biti neograničen, međutim, može doći do problema ukoliko dva master uređaja pokušaju da vrše komunikaciju sa istim slave uređajem. U tom slučaju svaki master uređaj mora da detektuje da li je SDA linija u stanju logičke nule ili logičke jedinice pre nego što pošalje poruku. Ukoliko master uređaj detektuje da je magistrala zauzeta, odlaže slanje podataka sve dok ne detektuje stanje logičke 1 na SDA liniji. U slučaju da više master uređaja započne slanje podataka istovremeno, komunikacija se prepušta onom master uređaju koji je prvi postavio logičku 0 na SDA liniju (*arbitration*).

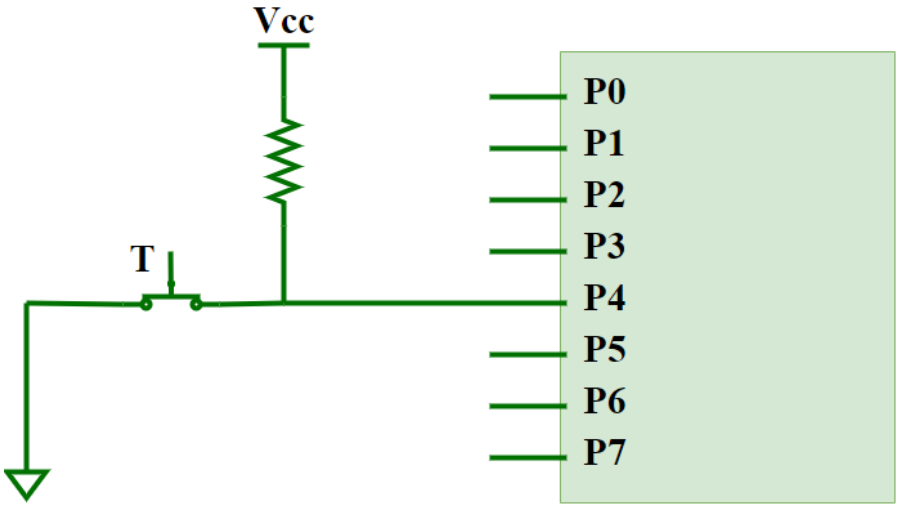
SDA i SCL linije koje čine I^2C magistralu realizuju se u **open-drain**, odnosno, **open-collector** konfiguraciji. Open-drain (collector) konfiguracija jedna je od najčešćih metoda konfigurisanja izlaza integrisanih kola i ona omogućava bidirekcionu komunikaciju putem jedne linije i mogućnost povezivanja više uređaja na jednu liniju, što je upravo i značajno za I^2C komunikacione linije. Termin open-drain odnosi se na upotrebu MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*) tranzistora čiji je drain izvučen van kola kao pin (slika 2), dok je open-collector termin vezan za ovakvu upotrebu BJT (*Bipolar Junction Transistor*) tranzistora. Ova konfiguracija podrazumeva da se umesto propagiranja izlaznog signala određenog napona direktno na izlaz, signal šalje na gate, odnosno base, NMOS, odnosno NPN, tranzistora, čiji je drain (collector) izvučen van kola. Emitter i source ovakvih tranzistora povezani su na masu. Pin koji je označen kao open-drain/collector ponaša se kao prekidač koji se uključuje i isključuje u zavisnosti od signala koji se dovodi na tranzistor.



Slika 2: Open-drain konfiguracija

Kada su izlazni pinovi u ovakvoj konfiguraciji mogu zauzeti dva stanja – nizak naponski nivo i stanje visoke impedanse. Kada je tranzistor isključen na pinu je visoka impedansa i stanje signala nije definisano. Iz tog razloga u open-drain/collector konfiguraciji neophodna je upotreba **pull-up otpornika** za definisanje visokog naponskog nivoa kada je tranzistor isključen, i oni se mogu videti na slici 1. Pull-up otpornik predstavlja otpornik preko koga se linija povezuje na napon napajanja i na

taj način, ukoliko nema spoljašnjeg uticaja na stanje linije, putem pull-up otpornika definiše se stanje logičke 1. Na slici 3 prikazan je primer povezivanja tastera preko pull-up otpornika na određeni pin. U slučaju kada ne bi bilo pull-up otpornika, kada taster nije pritisnut, pin P4 bi bio u stanju visoke impedanse i ponašao se kao antena za smetnje iz okoline. Uvođenjem pull-up otpornika, u slučaju kada taster nije pritisnut, na pin se dovodi napon Vdd, odnosno, očitava se stanje logičke 1, dok u slučaju kada je taster pritisnut, pin se vodi na masu i očitava se stanje logičke 0. Pri odabiru vrednosti za pull-up otpornik potrebno je voditi računa o povećanju disipacije snage za male vrednosti otpornosti, odnosno o smanjenju napona na pinu, za prevelike vrednosti. Kod I^2C linija najčešće vrednosti za pull-up otpornik su između 4,7 kΩ i 10 kΩ. Takođe, u kombinaciji sa parazitnim kapacitivnostima linija, pull-up otpornik formira RC filter, i u odnosu na vrednosti kapacitivnosti (maksimalno 400 pF) i otpornosti, može značajno da utiče na signal na pinu, o čemu se pri projektovanju sistema mora voditi računa.

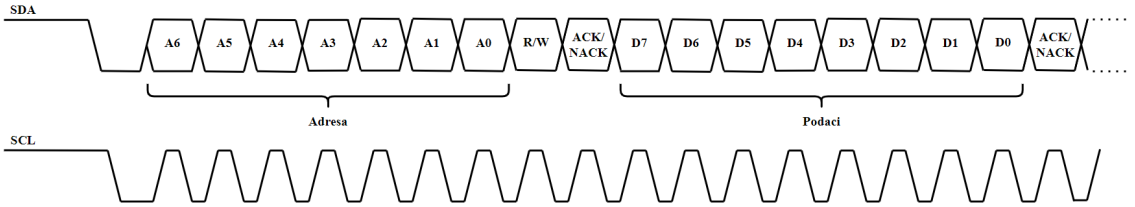


Slika 3: Pull-up otpornik

Ukoliko je signal na gate-u NMOS tranzistora u stanju logičke 0, tranzistor je isključen i na pin se, preko pull-up otpornika dovodi napon napajanja. U slučaju da je NMOS uključen pin se vezuje na masu. Kada su linije I^2C postavljene u navedenoj konfiguraciji uređaji koji se na njih vezuju mogu jedino da obore signal na liniji na logičku 0, ili da ne utiču na stanje na liniji, i tada pull-up otpornik podiže napon na liniji na napon napajanja. Kako nijedan od povezanih uređaja ne forsira visok naponski nivo na liniji, ne dolazi do slučaja da uređaji istovremeno forsiraju dva različita stanja. Na ovaj način više uređaja može biti povezano na jednu liniju – kada uređaji nisu aktivni pull-up otpornik održava napon napajanja na liniji, a ukoliko je jedan ili više uređaja aktivno napon na liniji se obara.

2 Prenos podataka kod I^{2}C komunikacije

Podaci se kod I^{2}C komunikacionog protokola prenose u vidu poruka (slika 4) koje se sastoje od slave adrese, nekoliko okvira podataka, start bita, stop bita, bita za čitanje/pisanje i ACK (*Acknowledge*)/NACK (*Not Acknowledge*) bita.



Slika 4: Poruka u I^{2}C komunikacionom protokolu

Kako bi razmena podataka otpočela potrebno je da master promeni stanje SDA linije sa logičke 1 na 0, a zatim posle određenog vremena, i stanje SCL linije sa logičke 1 na 0 (slika 5). Navedene promene na SDA i SCL liniji označavaju start stanje u I^{2}C protokolu. U nekim slučajevima moguće je generisanje start stanja više puta bez generisanja stop stanja (*Repeated Start*), i ovakav vid komunikacije koristi se pri promeni smera toka podataka ili uzastopnim pokušajima slanja podataka i sl.

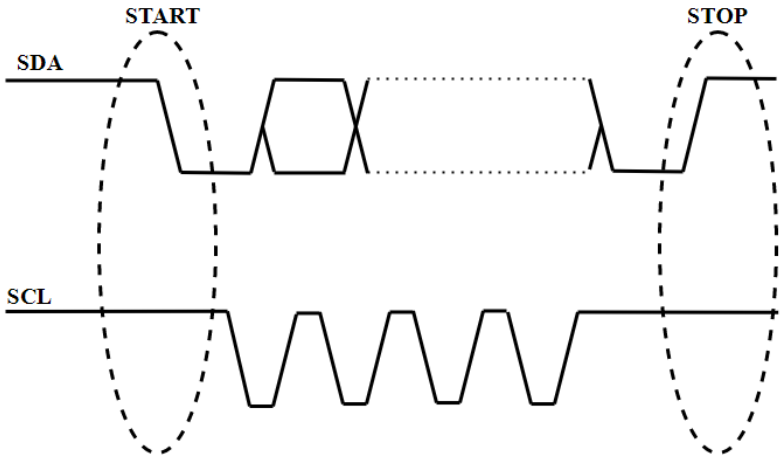
Nakon što je komunikacija započeta, potrebno je da master pošalje sedam do deset bitova koji označavaju adresu određenog slave uređaja sa kojim master želi da vrši razmenu podataka. Kada master uređaj postavi adresu na SDA liniju, svaki od slave uređaja poredi datu adresu sa svojom adresom, koja je najčešće fabrički podešena. Ukoliko prosleđena adresa odgovara adresi nekog od slave uređaja povezanih na magistralu, on započinje komunikaciju sa masterom, dok drugi uređaji od tog trenutka ne primaju komande od mastera, sve dok se komunikacija sa datim slave uređajem ne završi.

Nakon adekvatnog adresiranja slave uređaja, potrebno je da master pošalje jedan bit, označen kao *Read/Write*, koji naznačava smer toka podataka, odnosno, da li se vrši upis podataka na slave uređaj, ili se podaci primaju od slave uređaja. Ukoliko je vrednost ovog bita logička 0, master treba da šalje podatke na slave, a ukoliko je vrednost logička 1, master treba da prima podatke od slave uređaja.

Kod I^{2}C protokola postoji mehanizam za proveru da li su podaci primljeni u vidu ACK, odnosno NACK bita. Kada master uređaj pošalje podatke o adresi slave uređaja i bit za čitanje/upis pull-up otpornik podiže stanje SDA linije na visok naponski nivo. Ukoliko na magistrali postoji slave uređaj sa prosleđenom adresom, i ukoliko je primio bit za čitanje/upis, potrebno je da spusti naponski nivo SDA linije na nizak napon i ovo se označava kao ACK bit, odnosno, slave uređaj naznačava master uređaju da razmena podataka može da započne. U suprotnom, stanje SDA linije ostaje logička 1 i master prima NACK bit, što označava neuspešnu identifikaciju slave uređaja.

Kada master uređaj detektuje prvi ACK bit započinje razmena podataka u smeru koji je definisan bitom za čitanje/upis. Ukoliko se vrši upis podataka u slave uređaj

master šalje podatke, počevši sa MSB (*Most Significant Bit*), gde nakon svakog bajta slave uređaj mora da odgovori sa ACK ili NACK bitom. Blok bitova podataka praćenih ACK/NACK bitom ponavlja se sve dok se svi potrebni podaci ne prenesu. Kraj upisa podataka na slave uređaj označava se promenom stanja SDA linije sa logičke 0 na 1, dok je SCL linija u stanju logičke 1 (slika 5) - stop stanje. U suprotnom smeru, slave uređaj šalje podatke na master, koji zatim mora da postavi ACK/NACK bit na SDA liniju. Kada su svi podaci primljeni master šalje NACK bit i slave uređaj prestaje da šalje podatke, a SDA linija se vraća u neaktivno stanje.



Slika 5: Početak i kraj poruke I^2C komunikacionog protokola

Prenos podataka kod I^2C komunikacije može se vršiti različitim brzinama u odnosu na koje se definiše nekoliko različitih modova:

1. Standard mode – brzina prenosa podataka je 100 kbps.
2. Fast mode – brzina prenosa podataka je 400 kbps.
3. High speed mode – brzina prenosa podataka je 3.4 Mbps.
4. Ultra fast mode – brzina prenosa podataka je 5 Mbps.

Brzinu prenosa podataka moguće je menjati na master uređaju, koji diktira brzinu komunikacije na osnovu frekvencije signala takta.

U slučaju da je slave uređaju potrebno više vremena za skladištenje primljenih podataka ili slanje podataka, u odnosu na vreme prijema podataka, kako master ne bi detektovao lažne podatke, slave uređaj može da drži SCL liniju na niskom naponskom nivou i master uređaj čeka na podatke sve dok se stanje SCL linije ne promeni na visok naponski nivo. Na ovaj način signal takta je produžen i ovaj proces naziva se *clock stretching*.

3 I²C modul PIC18F87K22 mikrokontrolera

U sastavu PIC18F87K22 mikrokontrolera nalazi se MSSP (*Master Synchronous Serial Port*) modul koji predstavlja serijski komunikacioni interfejs i namenjen je za SPI i I²C komunikaciju. Ovakav modul omogućava komunikaciju PIC18F87K22 mikrokontrolera sa periferijama kao što su A/D i D/A konvertori, EEPROM ili drugi mikrokontroleri. PIC18F87K22 mikrokontroler sadrži dva ovakva modula – MSSP1 i MSSP2.

U slučaju upotrebe MSSP modula za I²C komunikaciju (slika 6) koriste se po dva pina mikrokontrolera – MSSP1 modul koristi pin RC3 kao SCL, a pin RC4 kao SDA, dok MSSP2 modul koristi pin RD6 kao SCL i pin RD5 kao SDA.

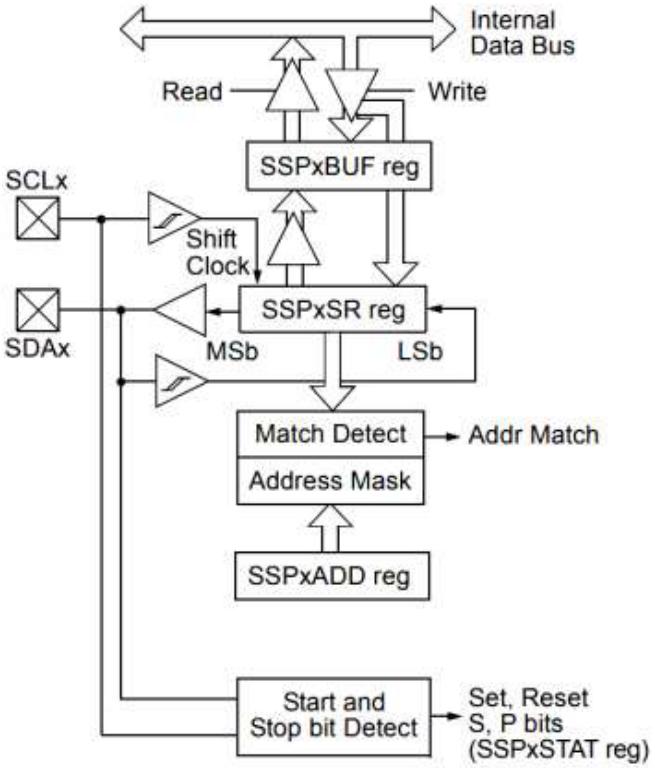
MSSP moduli datog mikrokontrolera za I²C komunikaciju konfigurišu se putem sedam registara:

1. SSPxSTAT (MSSPx Status Register) – statusni registar,
2. SSPxCON1 (MSSPx Control Register 1) – kontrolni registar,
3. SSPxCON2 (MSSPx Control Register 2) – kontrolni registar,
4. SSPxBUF (Serial Receive/Transmit Buffer Register) – bafer u koji se podaci upisuju ili iz koga se podaci čitaju,
5. SSPxSR (MSSPx Shift Register) – registar za pomeranje bita,
6. SSPxADD (MSSPx Address Register) – registar koji sadrži slave adresu,
7. SSPxMSK (I²C Slave Address Mask Register) – registar koji sadrži masku slave adrese kada je modul konfigurisan u 7-bitnom modu.

Podešavanjem I²C moda MSSP modula u SSPxCON1 registru SCL i SDA pinovi konfigurišu se kao open-drain, i modul se konfiguriše kao master, slave sa 7-bitnom adresom, slave sa 10-bitnom adresom, slave sa 7-bitnom adresom i start i stop bitom, slave sa 10-bitnom adresom i start i stop bitom ili kao master kontrolisan firmverom.

Ukoliko je modul konfigurisan kao slave uređaj potrebno je da SCL i SDA pinovi budu podešeni kao input u odgovarajućim TRIS registrima. Nakon start bita, u SSPxSR registar pristiže osam bitova, koji se odbirkuju na uzlaznu ivicu signala takta. Bitovi na pozicijama od jedan do sedam se porede sa bitovima u SSPxADD registru, i ukoliko su vrednosti iste generiše se prekid i postavlja se SSPxIF flag bit. Tada se generiše ACK bit i sadržaj SSPxSR registra prenosi se u SSPxBUF registar. U odnosu na to da li master upisuje ili čita podatke iz slave uređaja, postavlja se odgovarajući bit SSPxSTAT registra. Ukoliko slave uređaj prima podatke sa SDA linije, oni se pomeraju u SSPxSR registar, a po generisanju ACK bita, prebacuju se u SSPxBUF registar. Slave uređaj koji šalje podatke drži SDA liniju na visokom naponskom nivou (NACK) sve dok podaci nisu spremni za slanje, tako da master uređaj ne može da generiše novu četvrtku takta. Podaci koji se šalju smeštaju se u SSPxBUF registar, odakle se šalju u SSPxSR registar. Iz SSPxSR registra bit po bit se pomera na SDA liniju na svaku opadajuću ivicu takta. Nakon svakog prijema ili prenosa podataka generiše se prekid i postavlja se flag bit SSPxIF, koji mora biti vraćen na inicijalnu vrednost softverski.

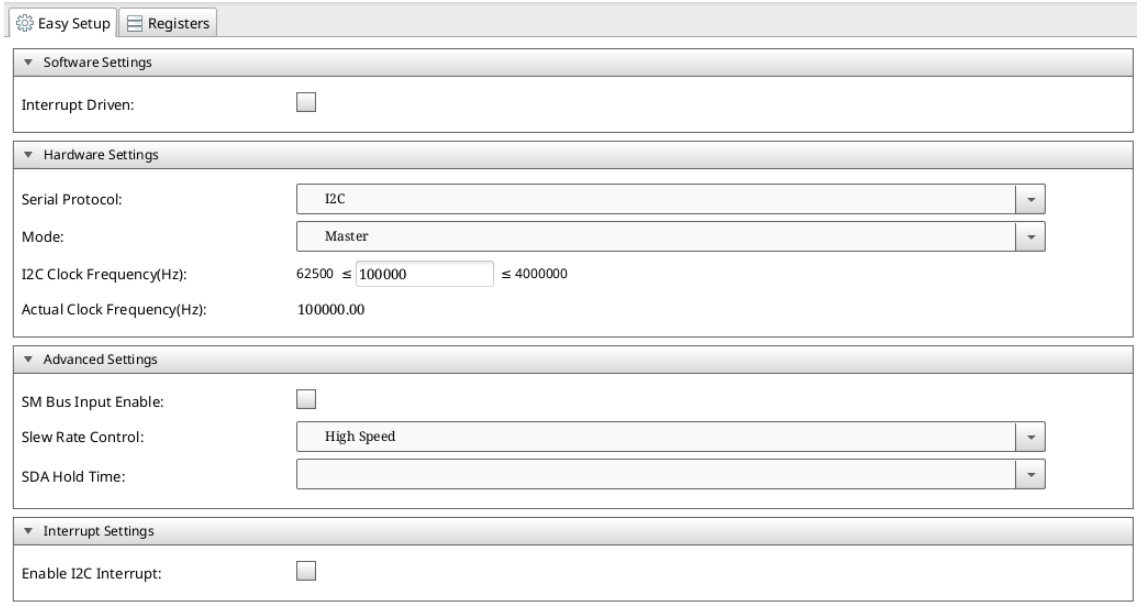
U slučaju kada je modul konfigurisan tako da radi kao master uređaj neophodno je da generiše signal takta, kao i start i stop bitove. Slanje podataka od strane master uređaja vrši se tako što se podaci upisuju u SSPxBUF, a zatim baud rate generator počinje sa brojanjem. Na svaku opadajuću ivicu signala takta na SCL liniji, po jedan bit podataka ili adrese se pomera na SDA liniju. Nakon prenosa svih osam bitova neophodno je da master oslobodi SDA liniju kako bi odgovarajući slave uređaj mogao da odgovori sa ACK/NACK bitom. Takođe, nakon devetog takta baud rate generator prestaje sa radom do trenutka kada sledeći bajt podataka ne bude upisan u SSPxBUF, SCL linija ostaje u stanju logičke 0, a SDA linija zadržava poslednje stanje. Kada master uređaj prima podatke sa slave uređaja, baud rate generator počinje sa brojanjem i na svaku promenu stanja na SCL pinu bit po bit se upisuje u SSPxSR registar. Nakon opadajuće ivice osmog takta sadržaj SSPxSR registra upisuje se u SSPxBUF, postavlja se odgovarajući flag bit, baud rate generator prestaje sa brojanjem i SCL ostaje u stanju logiče 0.



Slika 6: I^{2}C modul PIC18F87K22 mikrokontrolera

4 Implementacija I^{2}C modula u MPLAB-u

Kako bi se koristio I^{2}C 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 7.



Slika 7: Podešavanja MSSP1 modula

Nakon dodavanja MSSP1 modula neophodno je odabrati da radi u I^2C modu, odabiranjem iz padajućug menija *Serial Protocol* opcije *I2C*. Pored *I2C* MSSP1 modul može raditi i u *SPI* modu. Takođe, neophodno je odabrati i master mod rada, odabirom iz padajućeg menija *Mode* opcije *Master*, gde će mikrokontroler biti glavni uređaj na I^2C magistrali. U polje *I2C Clock Frequency(Hz)* se unosi frekvencija takt signala koji će se generisati na *SCL* liniji tj. ovo polje definiše brzinu prenosa podataka. Nakon unosa brzine, u polju *Actual Clock Frequency(Hz)* biće prikazana konfigurisana vrednost frekvencije. U poljima za dodatna podešavanja *Software Settings* i *Interrupt Settings* moguće je uključiti prekide prilikom korišćenja I^2C modula. U primerima koji slede prekidi neće biti korišćeni. U dodatnim podešavanjima *Advanced Settings* moguće je uključiti *SM Bus* protokol koji se bazira na I^2C protokolu i može biti podržan od nekih senzora. Pošto će u primerima koji slede biti korišćeni senzori koji koriste samo I^2C protokol ova opcija neće biti korišćena. Prilikom dodavanja MSSP1 modula i njegove konfiguracije u I^2C modu, pinovi *RC4* i *RC3* se podešavaju kao *SDA* i *SCL* linije.

Kada se generiše kod dobija se fajl *i2c1_master.c* u kom se nalaze smeštene funkcije za kontrolu I^2C protokola. Kako je korišćenje ovih funkcija komplikovano, MCC generator koda generiše i folder *examples* sa izvornim fajlom *i2c1_master_example.c*. U ovom fajlu generisane su funkcije za jednostavniju manipulaciju na I^2C magistrali.

Funkcija *I2C1_Read1ByteRegister* očekuje dva 8-bitna parametra, prvi koji predstavlja adresu senzorskog modula na koji se podaci žele poslati i drugi registar iz kog se žele očitati podaci. Na osnovu prosleđenih podataka i manipulacija I^2C magistralom kao povratna vrednost dobija se 8-bitni podatak dobijen od perifernog uređaja sa kojim se komunicira. Ova funkcija generiše start bit, za kojim sledi slanje adrese u modu upisa (*write* mod), nakon čega slave uređaj daje signal da je prisutan na liniji (ACK). Pošto se senzor odazove, prosleđuje se adresa registra koji se želi očitavati, nakon čega ponovo slave uređaj potvrđuje prijem podataka. Master uređaj, zatim,

generiše ponovni start (*repeated start*) i prosleđuje adresu ali sada poslednji bit postavlja u mod čitanja (*read*), nakon čega slave uređaj odgovara sa ACK bitom i sa 8 bita podataka. Po prijemu podataka master uređaj vraća liniju na visoko stanje (generiše NACK), što daje signal slave uređaju da je gotova komunikacija, pa master uređaj generiše stop bit. Na slici 8 prikazan je način razmene poruka master i slave uređaja korišćenjem *I2C1_Read1ByteRegister* funkcije.

MASTER	START	ADRESA UPIS		ADRESA REGISTRA		START	ADRESA ČITANJE			NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		

Slika 8: Razmena podataka *I2C1_Read1ByteRegister* funkcijom

Funkcija *I2C1_Read2ByteRegister* očekuje dva 8-bitna parametra, prvi koji predstavlja adresu senzorskog modula na koji se podaci žele poslati i drugi registar iz kog se žele očitati podaci. Na osnovu prosleđenih podataka i manipulacija I^2C magistralom kao povratna vrednost dobija se 16-bitni podatak dobijen od perifernog uređaja sa kojim se komunicira. Ova funkcija se ponaša isto kao i *I2C1_Read1ByteRegister* funkcija samo što se umesto nakon prijema prvog bajta podataka umesto NACK bita, šalje ACK bit što daje signal slave uređaju da prosledi sledeći bajt. Po prijemu drugog bajta master uređaj generiše NACK bit, za kojim se generiše stop bit i prekida se izvršavanje. Na slici 9 prikazan je način razmene poruka master i slave uređaja korišćenjem *I2C1_Read2ByteRegister* funkcije.

MASTER	START	ADRESA UPIS		ADRESA REGISTRA		START	ADRESA ČITANJE			ACK		NACK	STOP
SLAVE			ACK		ACK			ACK	DATA		DATA		

Slika 9: Razmena podataka *I2C1_Read2ByteRegister* funkcijom

Funkcija *I2C1_ReadDataBlock* se koristi za očitavanje više podataka u jednoj sekvenci očitavanja. Ova funkcija očekuje četiri parametra i to prvi parametar koji prestavlja 8-bitnu vrednost adrese modula sa kojim se želi komunicirati, drugi 8-bitni parametar predstavlja memorijski registar modula od kog se želi čitati, treći parametar predstavlja pokazivač na *uint8_t* niz u koji će biti smešteni očitani podaci i četvrti parametar predstavlja broj koliko se parametara želi očitati. Ova funkcija nema povratnih vrednosti jer će očitane vrednosti biti smeštene u prosleđenom nizu. Funkcija *I2C1_ReadDataBlock* se ponaša isto kao i *I2C1_Read1ByteRegister* i *I2C1_Read2ByteRegister* funkcije, ali nakon prijema bajta podataka (*DATA*) šalje ACK bit sve dok ne primi dovoljan broj bajta koje je korisnik specificirao nakon koga se šalje NACK bit i stop bit.

Funkcija *I2C1_Write1ByteRegister* očekuje tri 8-bitna parametra, prvi koji predstavlja adresu senzorskog modula na koji se podaci žele poslati, drugi registar u koji se žele upisati podaci i treći podatak koji predstavlja podatke koji će se upisati u

prosleđeni registar. Na osnovu prosleđenih podataka vrši se manipulacija I^2C magistralom i funckija završava izvršavanje bez povratnih vrednosti. Prvo se šalje start bit za kojim sledi adresa slave uređaja u modu upisa gde slave uređaj odgovara ACK bitom. Zatim se šalje adresa registra slave uređaja u koji se želi upisati jedan bajt i čeka se ACK odgovor slave uređaja. Na kraju se šalje bajt podataka i čeka se ACK odgovor slave uređaja, nakon čega se završava slanje podataka generisanjem stop bita. Na slici 10 prikazan je način razmene poruka master i slave uređaja korišćenjem *I2C1_Write1ByteRegister* funkcije.

MASTER	START	ADRESA UPIS		ADRESA REGISTRA		DATA		STOP
SLAVE			ACK		ACK		ACK	

Slika 10: Razmena podataka *I2C1_Write1ByteRegister* funkcijom

Funkcija *I2C1_Write2ByteRegister* očekuje dva 8-bitna i jedan 16-bitni parametra, prvi koji predstavlja adresu senzorskog modula na koji se podaci žele poslati, drugi registar u koji se žele upisati podaci i treći 16-bitni podatak koji predstavlja dva bajta podataka koji će se upisati u željeni registar. Na osnovu prosleđenih podataka vrši se manipulacija I^2C magistralom i funckija završava izvršavanje bez povratnih vrednosti. Funkcija *I2C1_Write2ByteRegister* se ponaša kao i *I2C1_Write1ByteRegister* s tom razlikom da ona šalje umesto jednog bajta dva bajta. Nakon svakog poslatog bajta slave uređaj odgovara ACK bitom, gde se nakon drugog bajta i primljenog ACK bita šalje stop signal. Na slici 11 prikazan je način razmene poruka master i slave uređaja korišćenjem *I2C1_Write2ByteRegister* funkcije.

MASTER	START	ADRESA UPIS		ADRESA REGISTRA		DATA		DATA		STOP
SLAVE			ACK		ACK		ACK		ACK	

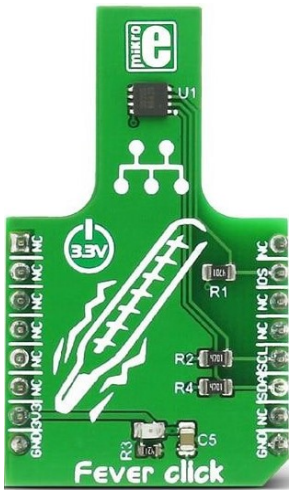
Slika 11: Razmena podataka *I2C1_Write2ByteRegister* funkcijom

Pošto se u memoriji podaci skladište tako da svaka adresa adresira jedan bajt, promenljiva koja čuva više bajta skladišti se na više uzastopnih memorijskih lokacija. Ovaj mikrokontroler podatke čuva u *little endian* poretku, pa se tako npr. dvobajta promenljiva 0xAABB u memoriji čuva tako što se na nižoj adresi čuva 0xBB, a na sledećoj 0xAA vrednost. Pošto funkcija *I2C1_Write2ByteRegister* u svojoj implementaciji koristi pokazivač na prosleđene parametre, posredstvom I^2C protokola prvo će biti poslat LSB bajt, a zatim MSB bajt prosleđenog 16-bitnog parametra. Ovo može prouzrokovati probleme, ako je drugačije specificirano uputsvom čipa sa kojim se želi komunicirati.

5 Primeri

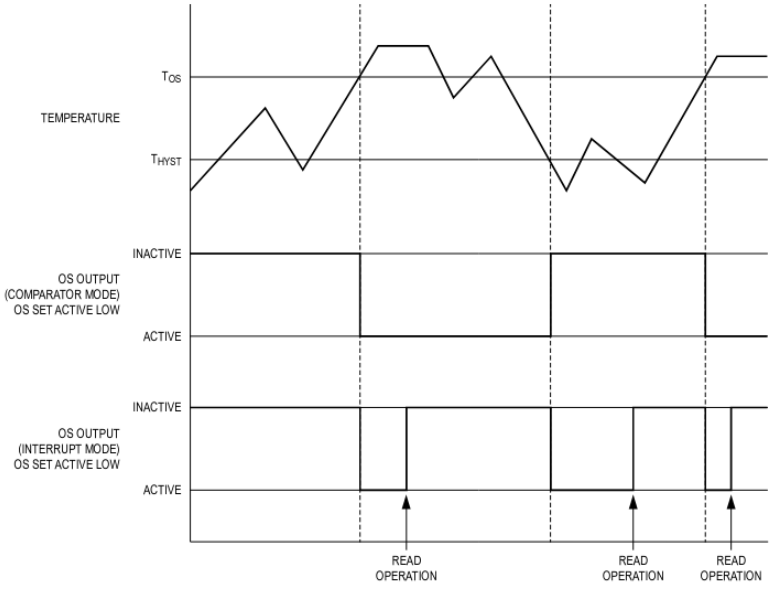
Primer 1: *Implementacija Fever click pločice*

Na slici 12 prikazana je click pločica sa pripadajućim senzorskim čipom MAX30205.



Slika 12: *Fever click* modul

Na *Fever click* pločici izveden je vod za napajanje (3,3 V), signali za I^2C protokol (SDA i SCL) i OS pin. U internim registrima čipa MAX30205 moguće je postaviti temperaturu histerezisa T_{hyst} registar, kao i temperaturni limit T_{os}. Na osnovu podešavanja ova dva registra moguće je definisati mod rada OS pina i to u komparatorskom ili prekidnom (*interrupt*) modu. Na slici 13 prikazan je izlaz OS pina u zavisnosti podešavanja registara T_{hyst} i T_{os} od promene temperature i to u oba moda rada.



Slika 13: Modovi rada OS pina

Podrazumevano izlaz OS pina je aktivan na niskom naponskom nivou. U komparatorskom modu rada, poredi se vrednost temperature sa vrednošću T_{os} registra, kada temperatura poraste preko temperature specificirane u ovom registru izlaz OS se aktivira tj. njegovo stanje se postavlja na nizak naponski nivo. OS ostaje aktivan sve dok temperatura ne padne ispod napona specificiranog T_{hyst} i tada prelazi u visoko stanje (stanje logičke jedinice). Drugi mod rada, generiše silaznu ivicu sa svakim prelaskom preko T_{os} praga odnosno padom ispod T_{hyst} praga uzimajući u obzir histerezis. Stanje se vraća na visok nivo sa svakim očitavanjem temperaturnog registra.

U *header* fajlu *MAX30205.h*, listing 1, nalaze se predprocesorske direktive *#define* gde *MAX30205_ADDR* predstavlja sedmobitnu adresu slave uređaja, a zatim se definišu adrese registara u memoriji slave uređaja koje će biti izmenjene. Na kraju se vrši definicija protopivo funkcija koje su implementirane u *source* fajlu *MAX30205.c*, listing 2. Takođe, na početku fajla nalazi se predprocesorksa direktiva *#ifndef* (vrši proveru da li nije definisana odgovarajući makro) koja se koristi da spreči višestruko uključivanje *header* fajla. Prvi put kada se fajl uključi direktivom *#include* iz bilo kog drugog fajla, generiše se marko *MAX30205_H_*, pa će sa svakim narednim uključivanjem *#ifndef* iskaz biti netačan jer je makro već definisan. Važno je istaći da se važenje *#ifndef* završava direktivom *#endif*. Nasuprot *#ifndef* postoji i *#ifdef* direktiva koja će imati tačnu vrednost ako je makro napravljen. Ovako će prilikom kompajliranja svaki fajl biti uključen samo jednom jer bi se u suprotnom javila greška prilikom kompajliranja.

Listing 1: *Header* fajl implementacije MAX30205 senzora

```
#ifndef MAX30205_H_
#define MAX30205_H_

#include <xc.h>
#include "mcc_generated_files/examples/i2c1_master_example.h"

#define MAX30205_ADDR (0x90 >> 1)

#define MAX30205_TMP    0x00
#define MAX30205_CFG    0x01
#define MAX30205_THYST  0x02
#define MAX30205_TOS    0x03

void config_sensor(void);
float get_temperature(void);
void config_hyst(uint16_t temp);
void config_os(uint16_t temp);

#endif
```

Listing 2: Izvorni (*source*) fajl implementacije MAX30205 senzora

```
#include "MAX30205.h"

void config_sensor(void)
{
    I2C1_Write1ByteRegister(MAX30205_ADDR, MAX30205_CFG, 0x00);
}

void config_hyst(uint16_t temp)
{
    I2C1_Write2ByteRegister(MAX30205_ADDR, MAX30205_THYST, temp);
}
```

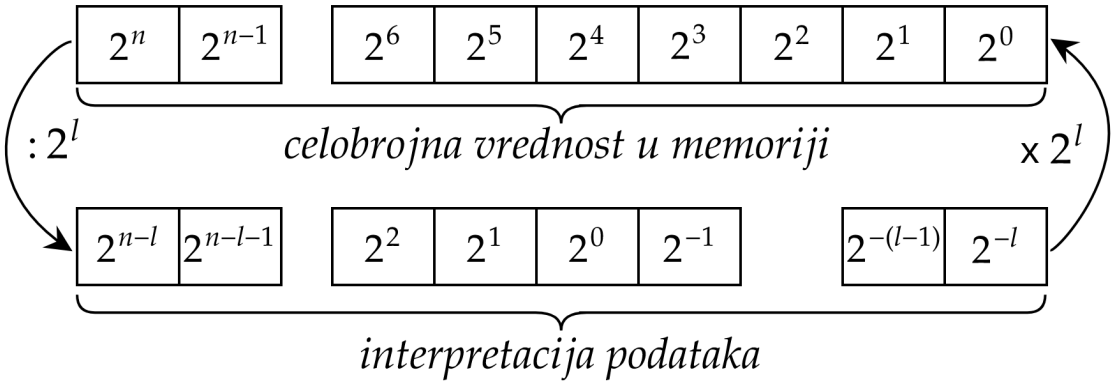
```
12
13 void config_os(uint16_t temp)
14 {
15     I2C1_Write2ByteRegister(MAX30205_ADDR,MAX30205_TOS,temp);
16 }
17
18 float get_temperature(void)
19 {
20         int16_t temp = I2C1_Read2ByteRegister(MAX30205_ADDR, MAX30205_TMP);
21         float temperature = temp  * 0.00390625;
22         return  temperature;
23 }
```

U fajlu *MAX30205.c* prvo je implementirana *config_sensor* funkcija sa zadatkom konfiguracije senzorskog modula. Prosleđena vrednost konfiguracionom registru je 0x00 čime je konfigurisan senzor i to: da bude u budnom stanju, da radi u komparatorskom modu rada, polaritet OS pina je invertovan (aktivno stanje je na logičku nulu), sa svakim prelaskom iznad i ispod T_{os} i T_{hyst} generiše se promena na OS pinu, ako dođe do generisanja logičke nule na *SDA* liniji doći će do resetovanja I^2C veze i senzor je podešen da kontinualno vrši merenje. Kako bi se konfigurisali registri T_{os} i T_{hyst}, a i kako bi se očitavala vrednost trenutne temperature, neophodno je poznavati format njihovog smeštanja u memoriju i on je prikazan na slici 14.

UPPER BYTE								LOWER BYTE							
D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
S	MSB 64°C	32°C	16°C	8°C	4°C	2°C	1°C	0.5°C	0.25°C	0.125°C	0.0625°C	0.03125°C	0.015625°C	0.0078125°C	0.00390625°C
	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}

Slika 14: Format podataka u temperaturnim registrima

Pošto mikrokontroler $(u)int16\_t$ dvobajtnu vrednost može interpretirati samo kao celobrojnu vrednost (u opsegu od 0 do 65535 za neoznačeni ili od -32768 do 32767 za označeni tip), neophodno je izvršiti adekvatnu transformaciju kako bi broj bio interpertiran kao što je prikazano na slici 14. Na slici 15 prikazana je transformacija za bilo koji registar. U ovom slučaju, kako bi se izvršila transformacija iz temperature u celobrojnu vrednost koja se upisuje u registre T_{os} i T_{hyst} neophodno je vrednost temperature pomnožiti sa 2^8 ($l = 8$) i tu vrednost upisati u registre T_{os} i T_{hyst}. Ovako konvertovana vrednost prosleđuje se funkcijama *config_hyst* i *config_os* gde prosleđena dvobajtna promenljiva upisuje u željeni registar. Kao što je napomenuto funkcija *I2C1_Write2ByteRegister* upisuje niži pa viši bajt prosleđene promenljive, stoga treba obratiti pažnju jer senzor prvo očekuje viši pa niži bajt. Ovaj problem će biti rešen jednostavnom zamenom bajta prilikom prosleđivanja parametara.



Slika 15: Transformacija podataka iz celobrojnog u decimalni

Funkcija *get_temperature* očitava dvobajtu promenljivu iz memorije slave uređaja. Ovaj podatak je neophodno konvertovati u decimalnu vrednost i to se vrši deljenjem sa 2^8 jer je $l = 8$. U ovom slučaju očitana vrednost je pomnožena sa $0,00390625$ što odgovara vrednosti 2^{-8}. Ova vrednost se kao *float* tip vraća kao rezultat izvršavanja funkcije.

U listingu koda 3 dat je glavni program za očitavanje senzora u kom se nakon inicijalizacije senzora i odgovarajućih parametara, očitava temperatura i njena vrednost se ispisuje zaokružena na dve decimale. Nakon 1 s ciklus se ponavlja. Konfigurisana vrednost T_{os} je 28 degC što odgovara vrednosti u registru $28 * 2^8 = 7168 = 0x1C00$, pa se funkciji *config_os* prosleđuje kao parametar $0x001C$ čime će se prvo poslati $0x1C$ pa $0x00$ preko *SDA* linije I^2C magistrale.

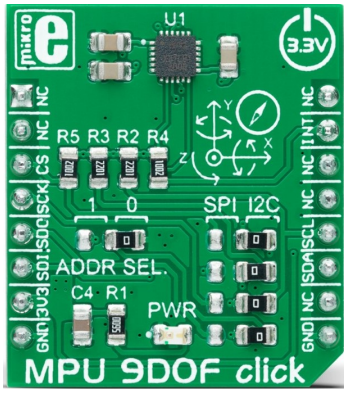
Listing 3: Glavni program

```
#include "mcc_generated_files/mcc.h"
#include "MAX30205.h"

void main(void)
{
    SYSTEM_Initialize();
    config_sensor();
    config_os(0x001C);
    config_hyst(0x001A);
    while (1)
    {
        float temp = get_temperature();
        printf("Temperatura je: %.2f\n",temp);
        __delay_ms(1000);
    }
}
```

Primer 2: *Implementacija MPU 9DOF click pločice*

MPU 9DOF click predstavlja pločicu sa čipom MPU9250 tj. 3-osnim akcelerometrom, 3-osnim žiroskopom i 3-osnim magnetometrom i prikazana je na slici 16. Važno je napomenuti da ovaj čip ima veliki broj funkcionalnosti, pa će u ovom primeru biti prikazan samo deo njih tj. biće očitane i prikazane sirove vrednosti po sve tri ose akcelerometra i žiroskopa.



Slika 16: *MPU 9DOF click* modul

U fajlu *MPU9250.h*, datog u listingu 4, napravljen je tip strukture MPU9250_t sa zadatkom da čuva sirove vrednosti koje će biti očitane iz registara akcelerometra i žiroskopa. Takođe, definisani su makroi koji čuvaju vrednosti sedmobitne adrese, kao i adrese pojedinih registara u koje će se upisvati ili očitavati vrednost. Na kraju definisani su prototipovi funkcija koje će biti implementriane u fajlu *MPU9250.h*, datog u listingu 5.

Listing 4: *Header* fajl implementacije MPU9250 senzora

```
1
2 #ifndef MPU9250_H_
3 #define MPU9250_H_
4
5 #include <xc.h>
6 #include "mcc_generated_files/examples/i2c1_master_example.h"
7
8 typedef  struct {
9         int16_t accel_x_raw;
10        int16_t accel_y_raw;
11        int16_t accel_z_raw;
12
13        int16_t gyro_x_raw;
14        int16_t gyro_y_raw;
15        int16_t gyro_z_raw;
16
17 }MPU9250_t;
18
19 #define MPU9250_ADDR 0x68
20
21 #define WHO_AM_I 0x75
22 #define PWR_MGMT_1 0x6B
23 #define PWR_MGMT_2 0x6C
24 #define ACCEL_CONFIG 0x1C
25 #define GYRO_CONFIG 0x1B
26 #define ACCEL_XOUT_H 0x3B
27 #define GYRO_XOUT_H 0x43
28
29 uint8_t MPU9250_who_am_i(void);
30
31 void MPU9250_init(void);
32 void MPU9250_read_accel(MPU9250_t *sensor);
33 void MPU9250_read_gyro(MPU9250_t *sensor);
34
35 #endif
```

Listing 5: Izvorni (*source*) fajl implementacije MPU9250 senzora

```
#include <xc.h>
#include "MPU9250.h"

uint8_t MPU9250_who_am_i(void)
{
    return I2C1_Read1ByteRegister(MPU9250_ADDR, WHO_AM_I);
}

void MPU9250_init(void)
{
    I2C1_Write1ByteRegister(MPU9250_ADDR, PWR_MGMT_1, 0x00);
    I2C1_Write1ByteRegister(MPU9250_ADDR, PWR_MGMT_2, 0x00);
    I2C1_Write1ByteRegister(MPU9250_ADDR, ACCEL_CONFIG, 0x10);
    I2C1_Write1ByteRegister(MPU9250_ADDR, GYRO_CONFIG, 0x08);
}

void MPU9250_read_accel(MPU9250_t *sensor)
{
    uint8_t tmp[6];

    I2C1_ReadDataBlock(MPU9250_ADDR, ACCEL_XOUT_H, tmp, 6);

    sensor->accel_x_raw = tmp[0]<<8 | tmp[1];
    sensor->accel_y_raw = tmp[2]<<8 | tmp[3];
    sensor->accel_z_raw = tmp[4]<<8 | tmp[5];
}

void MPU9250_read_gyro(MPU9250_t *sensor)
{
    uint8_t tmp[6];

    I2C1_ReadDataBlock(MPU9250_ADDR, GYRO_XOUT_H, tmp, 6);

    sensor->gyro_x_raw = tmp[0]<<8 | tmp[1];
    sensor->gyro_y_raw = tmp[2]<<8 | tmp[3];
    sensor->gyro_z_raw = tmp[4]<<8 | tmp[5];
}
```

Funkcija *MPU9250_who_am_i* očitava vrednost registra *WHO_AM_I* koji sadrži fabrički upisanu vrednost *0x71*. Na ovaj način moguće je testirati ispravnost komunikacije sa senzorom. Funkcija *MPU9250_init* podešava niz registara kako bi se senzor pripremio za slanje podataka. Upisom u registar *PWR_MGMG_1* vrednosti *0x00*, senzor se postavlja u budno stanje i podešava se da uvek bude u njemu, isključuje se stanje pripravnosti za žiroskop i podešava se takt za senzor na frekvenciju 20 MHz. Upisom vrednosti *0x00* u *PWR_MGMG_2* registar uključuje se merenje po x, y i z osi i akcelerometra i žiroskopa. Upisom u *ACCEL_CONFIG* registar vrednosti *0x10* isključuje se mod autotestiranja i podešava se opseg detekcije gravitacionog ubrzanja ±8g. Kako bi se konfigurisao žiroskop u registar *GYRO_CONFIG* upisana je vrednost *0x08*, čime je takođe isključeno autotestiranje, opseg žiroskopa podešen na 500 °/s i isključuje se filtriranje podataka digitalnim filterom. Funkcija *MPU9250_read_accel* očitava vrednosti koristeći *I2C1_ReadDataBlock* funkciju koja ima mogućnost da očita više sukcesivnih registara. Pošto se podatak o ubrzanju po svakoj od osa čuva u dvobajtnom registru, ova funkcija će očitati šest registara počevši od registra sa adresom *ACCEL_XOUT_H* koji predstavlja viši bajt rezultata očitavanja ubrzanja po x osi. Ovi podaci se smeštaju u niz *tmp* čiji se pokazivač prosleđuje funkciji *I2C1_ReadDataBlock* kao i broj bajta koji se želi očitati. Primljeni podaci se smeštaju u strukturu *sensor* koja je prosleđena kao pokazivač, pa se stoga za pristup poljima strukture koristi operator ->. Prva dva primljena podatka

se smeštaju u polje koje čuva sirove podatke za x osu, naredna dva u polje koje čuva sirove podatke za y osu i poslednja dva u polje koje čuva sirove podatke za z osu. Na isti način je realizovana funkcija *MPU9250_read_gyro* ali ona očitava podatke sa žiroskopa.

U listingu koda 6 izvršena je inicijalizacija senzorskog modula, a zatim se pozivaju funkcije za očitavanje sirovih vrednosti sa akcelerometra i žiroskopa. Važno je istaći da se parametar prenosi po adresi, jer funkcija očekuje pokazivač na strukturu. Nakon izmena podataka funkcija printf posredstvom UART-a ispisuje sirove podatke, očitane sa akcelerometra i žiroskopa, u terminalu *Data Visualizer* alata.

Listing 6: Glavni program

```
#include "mcc_generated_files/mcc.h"
#include "MPU9250.h"

void main(void)
{
    SYSTEM_Initialize();
    MPU9250_init();
    MPU9250_t sensor;
    while (1)
    {
        MPU9250_read_accel(&sensor);
        MPU9250_read_gyro(&sensor);

        printf("AccelX: %d | AccelY: %d| AccelZ: %d\n",
               sensor.accel_x_raw, sensor.accel_y_raw, sensor.accel_z_raw);
        printf("**************************************\n");
        printf("GyroX: %d | GyroY: %d| GyroZ: %d\n",
               sensor.gyro_x_raw, sensor.gyro_y_raw, sensor.gyro_z_raw);
        __delay_ms(1000);
    }
}
```