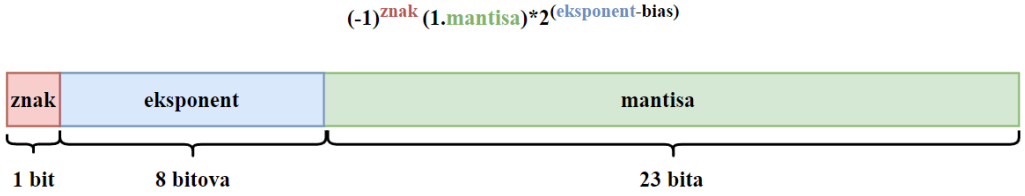
Fixed point i floating point aritmetika

1 Floating point

Aritmetika sa pokretnim zarezom odnosno *floating point* aritmetika predstavlja način za skladištenje podataka u decimalnom formatu. Kompajler *xc8 float* promenljivu definiše kao četvorobajtni podatak. Skladištenje podataka je definisano IEEE 754 standardom i prikazano je na slici 1.



Slika 1: Float IEEE 754

Prvi bit čuva podatak o znaku i ako je on na jedinici onda je broj negativan, dok je u slučaju nule interpretirani broj pozitivan. Eksponent je kodovan sa 8 bita i on se može kretati u opsegu od 0 do 255. Pošto se po IEEE 754 standardu broj čuva u naučnoj (*scientific*) notaciji u formatu $1.mantisa \cdot 2^x$, a kako bi se omogućilo da eksponent bude i negativan, neophodno je oduzeti *bias* od vrednosti eksponenta koja se čuva u *float* promenljivoj. U slučaju 32-bitne *float* promenljive *bias* iznosi 127. Tako je vrednost x definisana kao razlika eksponenta i *bias-a*, pa se zbog toga u eksponent uvek upisuje vrednost uvećana za 127. Na ovaj način je omogućeno da se čuvaju brojevi u opsegu od 2^{-127} do 2^{128}. Mantisa predstavlja broj značajnih cifara koji se nalaze iza decimalne tačke. Za ovaj slučaj veličina mantise je 23 bita.

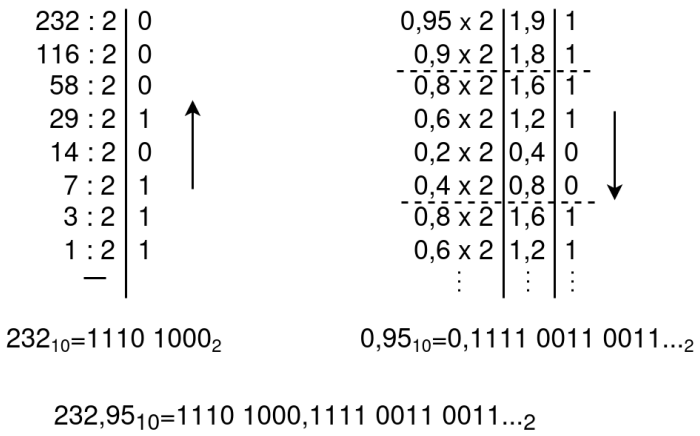
Primer: Konverzija broja 232,95 u IEEE 754 32-bitni format.

Prikaz broja sa pokretnim zarezom u IEEE 754 standardu se može podeliti u tri koraka:

- Korak I: Određivanje vrednosti u binarnom zapisu,
- Korak II: Zapis broja u naučnoj notaciji i određivanje eksponenta,
- Korak III: Određivanje znaka i prikaz broja u IEEE 754 formatu.

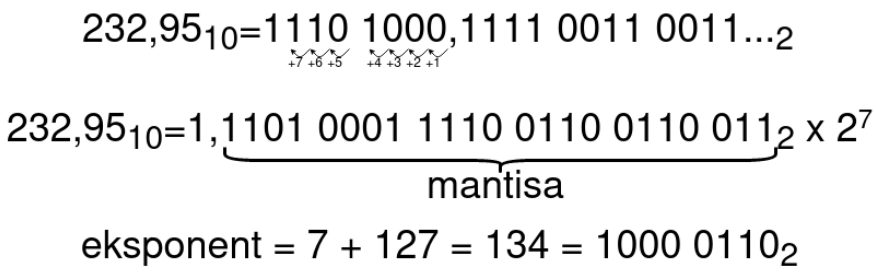
Na slici 2 prikazan je jedan od načina za binarno reprezentovanje broja 232,95. Prvo je broj 232 predstavljen u binarnom formatu. Algoritam za određivanje binarne vrednosti sastoji se iz deljenja vrednosti brojem dva (osnovom binarnog brojevnog sistema), a zatim se za svaku vrednost određuje ostatak. Kada se deljem dođe do tačke da je vrednost deljenika nula, binarni reprezent se dobija očitavanjem vrednosti ostataka u obrnutom redosledu od izračunavanja. Za vrednost 232, binarni reprezent je 0b11101000. Nakon toga određuje se binarni reprezent nakon decimalne tačke. Algoritam za njegovo određivanje sastoji se od množenja sa dva (osnovom binarnog brojevnog sistema), i ako je vrednost veća od jedinice onda se od dobijene vrednosti oduzima jedinica i nastavlja se sa množenjem, dok je u slučaju da je vrednost manja

od jedinici čuva se vrednost nula i nastavlja se sa množenjem. Npr. množenjem 0,95 sa 2 dobija se broj 1,9 i pošto je njegova vrednost veća od 1, vrednost 1 se čuva a nastavlja se sa množenjem broja 0,9. Nasuprot toga ako je vrednost broja 0,4 nakon množenja sa 2, rezultat je 0,8 i pošto je njegova vrednost manja od 1 čuva se vrednost 0 i nastavlja se sa množenjem broja 0,8. Pošto se posle nekoliko množenja uočava obrazac (množenjem 0,8 dobija se 0,6, zatim 0,2, pa 0,4 i ponovo 0,8) broj 0,95 u binarnom brojevnom sistemu ima beskonačan broj decimala koje se ponavljaju. Vrednost binarno reprezentovanog broja se dobija zapisom sačuvanih vrednosti ali sada u redosledu kako su brojevi i izrčaunavani. Pa je broj 0,95 u binarnom fomatu 0b0,111100110011.... Na osnovu određenih vrednosti za celobrojni i frakcioni deo dobija se binarni reprezent broja 232,95 kao 0b11101000,111100110011....



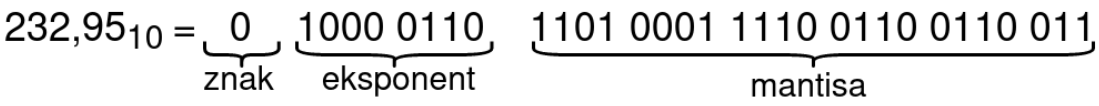
Slika 2: Određivanje vrednosti u binarnom zapisu

U drugom koraku na osnovu binarne vrednosti izvršeno je određivanje mantise i eksponenta, što je i prikazano na slici 3. Kako bi se broj napisao u naučnoj notaciji neophodno je decimalni zarez pomeriti sedam mesta u levo. Sa svakim pomeranjem eksponent se uvećava za jedan, pa će on na kraju pomeranja imati vrednost 7. Da bi se dobila vrednost eksponenta po IEEE 754 standardu vrednost ekspoenta se uvećava za 127, pa je vrednost eksponenta 134 odnosno 0b10000110. Vrednost nakon decimalne tačke predstavlja mantisu pa će za ovaj način prikaza mantisa imati 23 bita i biće jednaka 0b11010001111001100110011.



Slika 3: Određivanje mantise i eksponenta

U trećem koraku je određen znak i pošto je 232,95 pozitivan broj znak je jednak nuli. U slučaju da je reč o -232,95 on bi znak kodovao jedinicom. Konačno binarni reprezent broja 232,95 je 0b01000011011010001111001100110011 po IEEE 754 standardu.

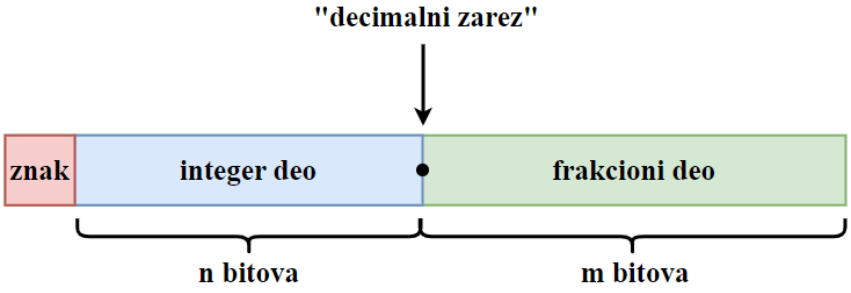


Slika 4: Određivanje znaka i reprezentaciju IEEE 754 standardu

Kada je reč o aritmetici brojeva sa pokretnom tačkom za sabiranje i oduzimanje je neophodno svesti eksponente na isti stepen, a zatim izvršiti sabiranje ili oduzimanje eksponenata. U slučaju množenja i deljenja eksponenti se sabiraju odnosno oduzimaju, dok se mantise množe odnoso dele. Ove operacije mogu biti vremenski zahtevene na mikrokontrolerima koji nemaju *FPU* (*Floating point unit*). *FPU* predstavlja zaseban modul mikrokontrolera čija je namena manipulacija nad brojevima sa pokretnim zarezom. Pošto mikrokontroler PIC18F87K22 nema *FPU* manipulacija sa *float* brojevima može biti duga, pa se zbog toga prezentacija vrši na drugi način.

2 *Fixed point*

Aritmetika sa nepokretnim zarezom (*Fixed-point*) predstavlja format zapisa gde je prezentacija frakcionog dela uvek definisana sa istim brojem bita. Na slici 5 dat je prikaz standardnog zapisa Qn.m (gde n predstavlja broj koji koduje celobrojne bitove, a m deo koji koduje frakcione bitove) broja sa nepokretnim zarezom.

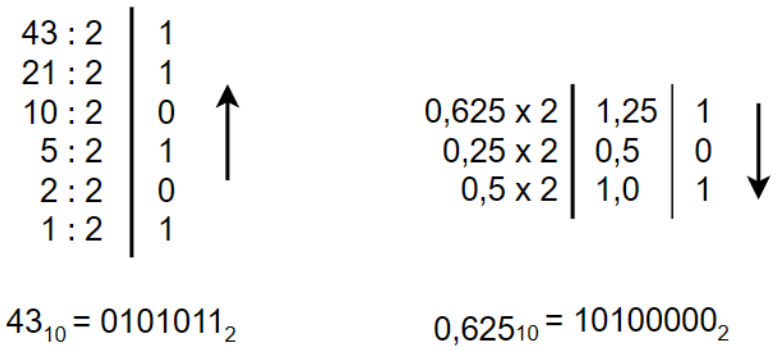


Slika 5: Fixed point

Važno je istaći da se sada znak ne koduje samo upisom 1 ili 0 u prvi bit već se, ako je broj negativan, mora odrediti komplement dvojke. Mikrokontroleri nemaju standardizovan format za skladištenje *fixed-point* brojeva pa se on realizuje korišćenjem dostupnih celobrojnih tipova.

Primer: Konverzija broja 43,625 u fixed point Q7.8 format.

Prikaz broja u fixed point zapisu zahteva određivanje vrednosti pre i posle decimalnog zareza, gde je Qn.m formatom određen broj bita koji je potrebno upotrebiti za reprezentaciju celog dela - n, i frakcioni deo - m. U prvom koraku vrši se konverzija broja 43 u binarni zapis, deljenjem sa 2 i posmatranjem ostatka pri deljenju sa 2, gde se binarni zapis očitava u suprotnom smeru od vršenja operacija deljenja. Na ovaj način za broj 43 dobijena je binarna vrednost 0b0101011. Važno je napomenuti da je broj 43 bilo neophodno predstaviti sa 7 bitova, kako je naznačeno Q7.8 formtom. U sledećem koraku posmatra se vrednost iza decimalnog zareza, koja se konvertuje u binarni zapis množenjem sa 2 i posmatranjem broja dobijenog množenjem sa 2. Ukoliko je rezultat množenja sa 2 manji od 0, u binarni zapis uvrštava se 0 i nastavlja se sa množenjem date vrednosti i 2, a ukoliko je rezultat množenja veći od 1, u binarni zapis uvrštava se 1 i nastavlja se množenje 2 i date vrednosti umanjenje za 1. Binarna vrednost očitava se u smeru vršenja operacija množenja. Ovim postupkom vrednost 0,625 predstavljena je u binarnom zapisu kao 0b10100000, gde je upotrebljeno 8 bitova za reprezentaciju vrednosti. Na slici 6 prikazani su postupci konverzije brojeva 43 i 0,625 u binarni zapis.



Slika 6: Konverzije brojeva 43 i 0,625 u binarni zapis

Ukoliko, na primer, posmatramo broj predstavljen binarnim zapisom po fixed point formatu Q4.4 0b0111.0001, do broja u dekadnom zapisu možemo doći kao:

$$\begin{aligned} 0111.0001_2 &= 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} \\ &= 2^2 + 2^1 + 2^0 + 2^{-4} \\ &= 7,0625 \end{aligned}$$

Drugi način jeste konverzija binarnog broja 0b01110001 u dekadni i množenje sa vrednošću LSB datog fixed point Q4.4 formata (2^{-4}).

$$\begin{aligned} 01110001_2 &= 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= 2^6 + 2^5 + 2^4 + 2^0 \\ &= 64 + 32 + 16 + 1 \\ &= 113 \end{aligned}$$

$$113 \cdot 2^{-4} = 113 \cdot 0,0625 = 7,0625$$

Dakle, broj u decimalnom zapisu možemo predtsaviti kao $INT \cdot 2^{-m}$, gde je INT ceo broj a m broj frakcionih bitova.

Ukoliko posmatramo dva broja predstavljena u fixed point formatu Qn.m - FP_A i FP_B, nad njima je moguće vršiti operacije sabiranja, oduzimanja, množenja i deljenja, tako da se dobije rezultat, takođe u fixed point formatu FP_C. Date brojeve moguće je predstaviti i kao proizvod celog broja - INT_x i vrednosti LSB za dati format - 2^{-m}.

- **Sabiranje**
  Rezultat sabiranja dva fixed point broja, u fixed point formatu:

$$\begin{aligned} FP_C &= FP_A + FP_B \\ FP_C &= INT_A \cdot 2^{-m} + INT_B \cdot 2^{-m} \\ FP_C &= (INT_A + INT_B) \cdot 2^{-m} \end{aligned}$$

  Integer broj rezultata se dalje može dobiti kao:

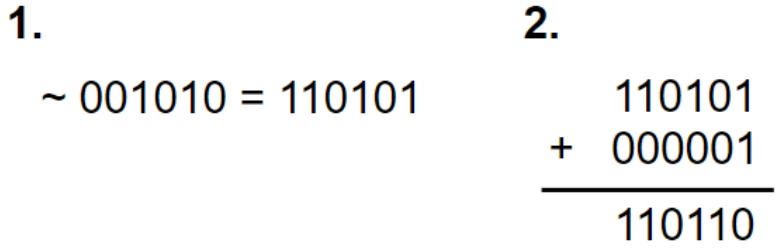
$$\begin{aligned} INT_C &= FP_C \cdot 2^m \\ INT_C &= (INT_A + INT_B) \cdot 2^{-m} \cdot 2^m \\ INT_C &= INT_A + INT_B \end{aligned}$$

  Primer: Sabiranje brojeva -1,25 i 3,25 (Q3.3 format).

  Prvi korak podrazumeva konverziju brojeva -1,25 i 3,25 u fixed point format Q3.3. Broj 1,25 odgovara binarnom zapisu 0b001.010:

$$\begin{aligned} 1,25 &= 1 + 0,25 \\ &= 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} \end{aligned}$$

  Za predstavu broja -1,25 neophodno naći komplement dvojke 0b001.010 - svi bitovi se negiraju, a zatim se dobijeni binarni broj 0b110.101 sabere sa 1 (slika 7). Dakle, broj -1,25 odgovara binarnom zapisu 0b110.110.



Slika 7: Komplement dvojke broja 0b001010

Broj 3,25 odgovara binarnom zapisu 0b011.010:

$$\begin{aligned} 3,25 &= 2 + 1 + 0,25 \\ &= 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} \end{aligned}$$

Kada se brojevi 0b110110 i 0b011010 saberu dobija se binarni broj 0b1010000, odnosno, kako rezultat u Q3.3 formatu zahteva predstavu celog dela sa 3 bita i frakcionog dela sa 3 bita, rezultat je 0b010.000, odnosno:

$$\begin{aligned} 010.000_2 &= 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 0 \cdot 2^{-3} \\ &= 2_{10} \end{aligned}$$

- **Oduzimanje**
  Rezultat oduzimanja dva fixed point broja, u fixed point formatu:

$$\begin{aligned} FP_C &= FP_A - FP_B \\ FP_C &= INT_A \cdot 2^{-m} - INT_B \cdot 2^{-m} \\ FP_C &= (INT_A - INT_B) \cdot 2^{-m} \end{aligned}$$

  Integer broj rezultata se dalje može dobiti kao:

$$\begin{aligned} INT_C &= FP_C \cdot 2^m \\ INT_C &= (INT_A - INT_B) \cdot 2^{-m} \cdot 2^m \\ INT_C &= INT_A - INT_B \end{aligned}$$

- **Množenje**
  Rezultat množenja dva fixed point broja, u fixed point formatu:

$$\begin{aligned} FP_C &= FP_A \cdot FP_B \\ FP_C &= INT_A \cdot 2^{-m} \cdot INT_B \cdot 2^{-m} \\ FP_C &= (INT_A \cdot INT_B) \cdot 2^{-2m} \end{aligned}$$

  Integer broj rezultata se dalje može dobiti kao:

$$\begin{aligned} INT_C &= FP_C \cdot 2^m \\ INT_C &= (INT_A \cdot INT_B) \cdot 2^{-2m} \cdot 2^m \\ INT_C &= (INT_A \cdot INT_B) \cdot 2^{-m} \end{aligned}$$

Primer: Množenje brojeva 5,125 i 4,25 (format Q3.3).

Broj 5,125 odgovara binarnom zapisu 0b101.001:

$$\begin{aligned} 5,125 &= 4 + 1 + 0,125 \\ &= 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} \end{aligned}$$

Odnosno:

$$\begin{aligned} 5,125_{10} &= 101.001_2 \\ &= 101001_2 \cdot 2^{-3} \end{aligned}$$

Broj 4,25 odgovara binarnom zapisu 0b100.010:

$$\begin{aligned} 4,25 &= 4 + 0,25 \\ &= 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} \end{aligned}$$

Odnosno:

$$\begin{aligned} 4,25_{10} &= 100.010_2 \\ &= 100010_2 \cdot 2^{-3} \end{aligned}$$

Množenjem ova dva broja dobija se:

$$\begin{aligned} FP_C &= 101.001 \cdot 100.010 \\ &= 101001_2 \cdot 2^{-3} \cdot 100010_2 \cdot 2^{-3} \\ &= (101001_2 \cdot 100010_2) \cdot 2^{-6} \\ &= (41_{10} \cdot 34_{10}) \cdot 2^{-6} \\ &= 21,78125 \end{aligned}$$

- **Deljenje**
  Rezultat deljenja dva fixed point broja, u fixed point formatu:

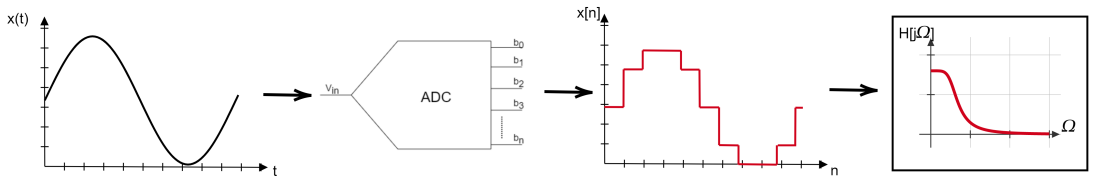
$$\begin{aligned} FP_C &= \frac{FP_A}{FP_B} \\ FP_C &= \frac{INT_A \cdot 2^{-m}}{INT_B \cdot 2^{-m}} \\ FP_C &= \frac{INT_A}{INT_B} \end{aligned}$$

  Integer broj rezultata se dalje može dobiti kao:

$$\begin{aligned} INT_C &= FP_C \cdot 2^m \\ INT_C &= \frac{INT_A \cdot 2^m}{INT_B} \end{aligned}$$

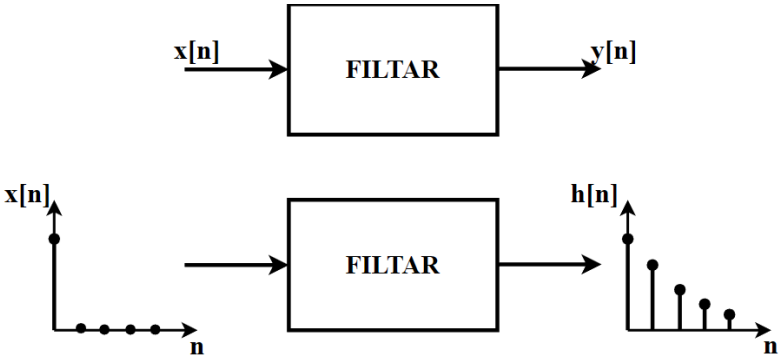
3 Digitalni filtri

Digitalni filtri predstavljaju značajne elemente digitalne obrade signala. Digitalni filtri su programabilni, te se njihovi parametri mogu veoma jednostavno podešavati i menjati, takođe, digitalne filtre je jednostavno implementirati i testirati na računarima. Još jedna od prednosti digitalnih filtara u odnosu na analogne jeste otpornost na parametre sredine, gde su analogni filtri podložni promenama usled temperature, parazitnih efekata komponenata itd. Digitalni filtri imaju ulogu otklanjanja neželjenih komponenata iz digitalnih signala. Digitalni filtri u sprezi sa analognim sistemima koriste A/D konvertor, gde se analogni signali x(t) sa različitih modula i senzora dovode na A/D konvertor na čijem izlazu se dobijaju odbirci analognih signala x[n], koji se zatim prosleđuju digitalnom filtru. Na osnovu odbiraka signala i određenih parametara, digitalni filtar preračunava vrednosti izlaza. Blok dijagram ovakvog sistema ilustrovan je na slici 9.



Slika 8: Filtriranje analognih signala digitalnim filtrom

Filtri se karakterizuju svojim impulsnim odzivom h[n], odnosno, funkcijom prenosa H(z). Kada se na ulaz filtra dovede impulsna pobuda, izlaz koji se dobija naziva se impulsnim odzivom. Za poznavanje izlaza filtra y[n] neophodno je znati vrednost ulaza x[n] i impulsni odziv h[n].



Slika 9: Ulaz, izlaz i impulsni odziv digitalnog filtra

U odnosu na odziv, digitalni filtri dele se na:

- Filtre sa konačnim odzivom (FIR - *Finite Impulse Response*)

$$y[n] = \sum_{k=0}^{M} a_k x[n-k]$$

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{M} a_k z^{-k}$$

- Filtre sa beskonačnim odzivom (IIR - *Infinite Impulse Response*)

$$y[n] = \sum_{k=0}^{M} a_k x[n-k] - \sum_{k=1}^{N} b_k y[n-k]$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} a_k z^{-k}}{1 + \sum_{k=1}^{N} b_k z^{-k}}$$

Na ovim vežbama biće implementiran FIR NF filtar na PIC18F87K22 mikrokontroleru, te će se dalja analiza digitalnih filtara zasnivati isključivo na FIR filtrima.

Jedan od načina projektovanja FIR filtara jeste na specificiranog frekvencijskog odziva, gde se impulsni odziv određuje kao inverzna Furijeova transformacija frekvencijskog odziva:

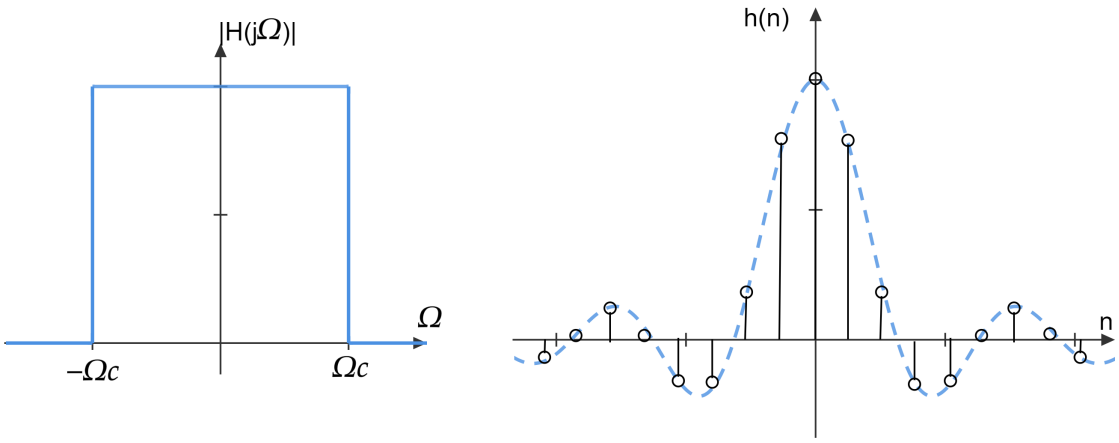
$$h(n) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} H(j\Omega) e^{j\Omega t}$$

Ukoliko se želi projektovati FIR NF filtar idealne frekvencijske karakteristike $H(j\Omega)$ (slika 10):

$$H(j\Omega) = \begin{cases} 1, |\Omega| \leq \Omega_c \\ 0, \text{inače} \end{cases}$$

Inverznom transformacijom ovakve frekvencijske karakteristike dobija se impulsni odziv filtra kao:

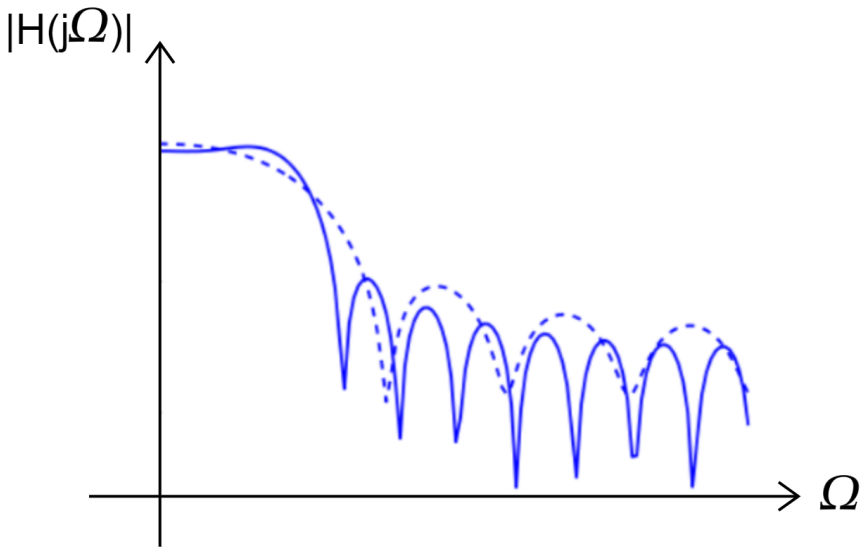
$$h[n]_{NF} = \frac{\Omega_c}{\pi} \cdot \frac{\sin(\Omega_c n)}{\Omega_c n}$$



Slika 10: Frekvencijski i impulsni odziv idealnog FIR NF filtra

Ovakav sistem nije kauzalan i beskonačne je dužine, te zahteva beskonačan broj koeficijenata, stoga nije praktično primenljiv, već je potrebno izvršiti određene modifikacije. Rešenje problema podrazumeva ograničavanje broja odbiraka impulsnog odziva h[n] na N i kašnjenje h[n] za $M = \frac{N-1}{2}$ odbiraka ili upotrebu određene prozorske funkcije u kombinaciji sa h[n].

Primer izgleda realne frekvencijske karakteristike NF FIR filtra prikazan je na slici 11.



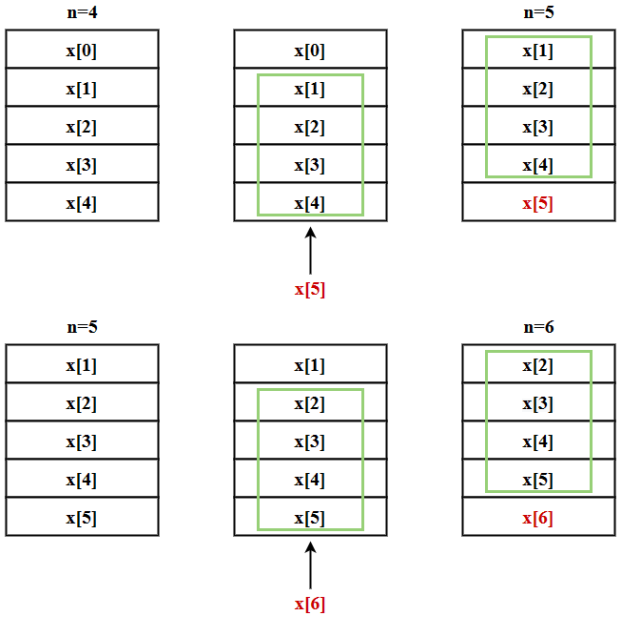
Slika 11: Frekvencijski odziv realnog FIR NF filtra

3.1 Cirkularni bafer

Pri digitalnoj obradi signala često je potrebno pamtiti vrednost signala u datom trenutku i u nekoliko prethodnih trenutaka. Za ove potrebe moguće je koristiti linearni shift registar, čije dimenzije odgovaraju broju vrednosti signala koje je potrebno pamtiti. Ukoliko, na primer, projektujemo FIR filtar čiji se izlaz računa kao:

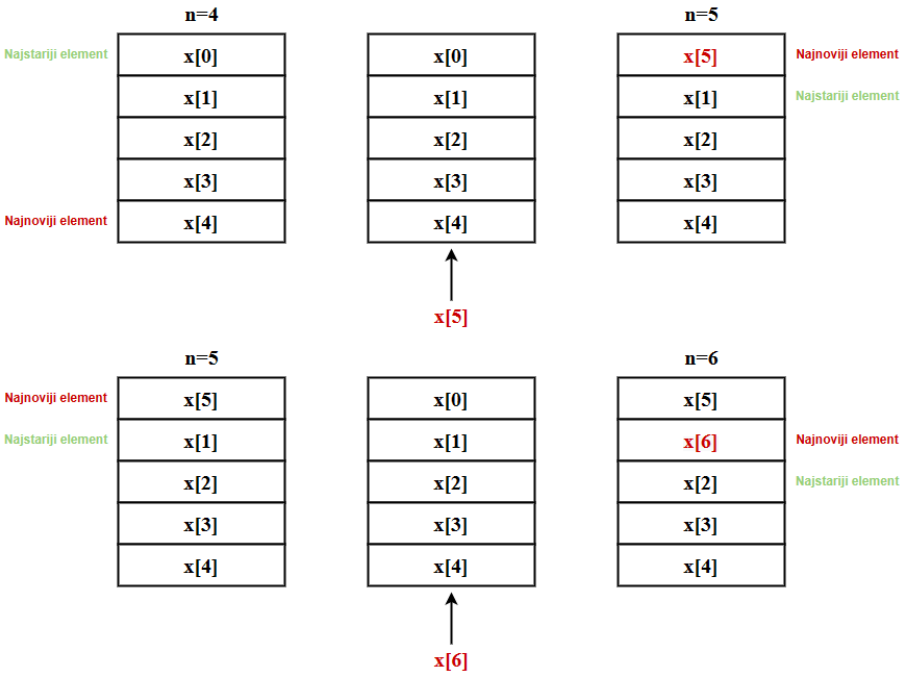
$$\begin{aligned} y[n] &= \sum_{k=0}^{4} a_k x[n-k] \\ &= a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + a_4 x[n-4] \end{aligned}$$

neophodno je pamtiti trenutnu vrednost ulaznog singla x[n], kao i vrednosti ulaznog signala u četiri prethodna trenutka x[n-1], x[n-2], x[n-3] i x[n-4]. Dakle, u n-tom trenutku u linearni shift registar dužine pet biće upisane vrednosti x[n], x[n-1], x[n-2], x[n-3] i x[n-4]. U narednom trenutku potrebno je odbaciti vrednost najstarijeg odbirka i u registar upisati vrednost najnovijeg registra. Na slici 12 ilustrovan je sadržaj linearnog shift registra u trenutku n=4. Kada u trenutku n=5 pristigne vrednost novog odbirka, za računanje izlaza FIR filtra od interesa će biti vrednosti x[1], x[2], x[3], x[4] i x[5], stoga je neophodno izvršiti pomeranje sadržaja registra za jedno mesto i upisati vrednost x[5] na poslednju memorijsku lokaciju registra. U svakom sledećem trenutku sadržaj registra se pomera za jedno mesto, čime se postiže odbacivanje vrednosti najstarijeg odbirka, dok se vrednost najnovijeg odbirka uvek upisuje na poslednju lokaciju u registru. Pri ovom postupku, u svakom koraku n potrebno je izvršiti četiri pomeranja vrednosti. Za filtar dužine N vrši se N-1 pomeranja u svakom trenutku n. Broj read/write instrukcija pri ovom pomeranju, dakle, proporcionalan je dužini filtra, te upotreba linearnih shift registara pri difitalnom filtriranju nije efikasna.



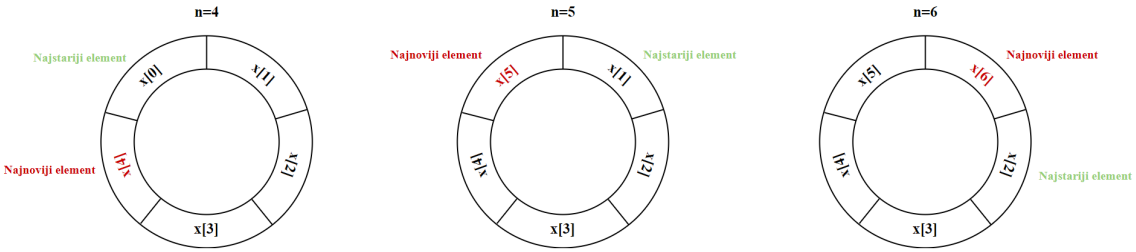
Slika 12: Upis vrednosti u linearni shift registar

U postupcima digitalne obrade signala najčešće se koriste **cirkularni baferi** za skladištenje vrednosti. Sa slike 12 moguće je uočiti da za trenutke n i n+1 postoje četiri zajedničke vrednosti x[n], x[n-1], x[n-2], x[n-3], dakle, efikasniji postupak bi bio zadržavanje vrednosti koje su zajedničke upravo na njihovim lokacijama u baferu i zamena najstarije vrednosti signala sa vrednošću najnovijeg odbirka. Na slici 13 ilustrovan je princip upisa vrednosti u cirkularni bafer.



Slika 13: Upis vrednosti u cirkularni bafer

U trenutku n=4 sadržaj registra čine vrednosti x[0], x[1], x[2], x[3] i x[4]. U sledećem trenutku, n=5, najstariji odbirak x[0] biva zamenjen vrednošću odbirka x[5], dok preostale vrednosti ostaju na mestima kao u trenutku n=4. Kada se sa upisivanjem najnovijeg odbirka stigne do kraja bafera, odnosno, kada je memorijska lokacija na koju se u datom trenutku n upisuje vrednost x[n], poslednja lokacija datog bafera, u narednom trenutku n+1, vrednost najnovijeg odbirka upisuje se na prvu memorijsku lokaciju bafera. Primena cirkularnih bafera zahteva poznavanje memorijske lokacije na koju je potrebno upisati najnoviji odbirak - potreban je pokazivač na lokaciju upisa najnovijeg odbirka. Pokazivač na lokaciju upisa najnovijeg odbirka se inkrementira po svakom upisu, a kada dostigne vrednost poslednje lokacije bafera, resetuje se na vrednost prve lokacije bafera, što omogućava cirkularni upis vrednosti. Dodatno, potreban je i pokazivač na memorijsku lokaciju najstarije nepročitane vrednosti u baferu. Na slici 14 ilustrovan je cirkularni bafer.



Slika 14: Cirkularni bafer

4 Primeri

Primer 1: *Poređenje aritmetike sa pokretnim i nepokretnim zarezom*

U listingu koda 1 data je implementacija aritmetike sa nepokretnim zarezom (*fixed-point arithmetics*) nakon čega je izvršeno poređenje pojedinih aritmetičkih operacija gde su operandi brojevi sa pokretnim (*floating point*) odnosno brojevi sa nepokretnim zarezom.

U ovom slučaju realizovana je aritmetika Q3.12 stoga je definisana konstanta $FRAC$ gde 12 predstavlja broj bita nakon decimalne tačke. Makroom *fp2float* se za parametar x vrši njegova konverzija iz *int* vrednosti (interpretirane kao broja sa nepokretnim zarezom) u broj sa pokretnim zarezom. Kako bi se izvršila ova konverzija neophodno je celobrojnu vrednost podeliti sa 2^{FRAC}. Pošto je ekvivalent 2^{FRAC} pomeranje u levo jedinice $FRAC$ puta, ovo je realizovano i u kodu, jer se sa stanovišta brzine izvršavanja brže izračunava bitsko pomeranje u odnosu na stepenovanje. Takođe, kako bi se *float* vrednost prevela u vrednost sa nepokretnim zarezom koristi se makro *float2fp* koji vrši konverziju množenjem sa 2^{FRAC}. Pošto se dobijena vrednost pretvara u celobrojnu, zbog celobrojnog zaokruživanja može se javiti greška zaokruživanja jer se tom prilikom vrednosti iza decimalne tačke otpisuju. Kako bi se uticaj ove greške eliminisao pre konverzije na celobrojnu vrednost, *float* vrednosti se dodaje 0,5.

Spram opisane aritmetike sa nepokretnim zarezom definisani su makroi i to:

- *fp_sum* koji određuje zbir prosleđenih parametara,
- *fp_sub* koji određuje razliku prosleđenih parametara,
- *fp_mul* koji određuje proizvod prosleđenih parametara,
- *fp_div* koji određuje količnik prosleđenih parametara.

U glavnom programu pozivom funkcije *rand* generiše se pseudo slučajani broj u opsegu između 0 i 32768. Kako bi se vrednost svela na opseg od 0 do 0,999, određen je ostatak pri deljenju (moduo) generisanog broja sa 1000 čime je generisan pseudo slučajni broj u opsegu od 0 do 999, nakon čega se njegova vrednost deli sa 1000 čime se dobija željeni opseg. Na ovaj način generisana su dva pseudo slučajna broja a i b. Dobijeni brojevi su konvertovani u vrednosti *uint16_t fp_a* i *fp_b* koje će se kasnije interpretirati kao Q3.12 brojevi sa nepokretnim zarezom.

Nakon generisanja brojeva kako bi se merilo vreme izvršavanja pojedinih instrukcija koristi se tajmerski registar *TMR1*. Tajmer je podešen u 16-bitni mod, frekvencija takta odgovara frekvenciji oscilatora f_{osc}, pa će se uvećanje tajmerskog registra dešavati na svakih 15,625 ns. U delu koda od linije 32 do linije 38, po resetovanju tajmerskog registra pozivom *TMR1_Reload* funkcije, vrši se merenje vremena potrebno za određivanje zbira, razlike i proizvoda brojeva sa nepokretnim zarezom. Na kraju se vrednost sadržana u tajmerskom registru čuva u promenljivoj *time_fp*. Identična procedura izvršena je za merenje vremena izvršavanja aritmetike sa pokretnim zarezom.

Na kraju se u terminalu prikazuje proteklo vreme, kao i rezultati pojedinih operacija. Akcenat je stavljen na sabiranje, oduzimanje i množenje jer se oni koriste u projektovanju digitalnih filtera.

Listing 1: Poređenje aritmetike sa pokretnim i nepokretnim zarezom

```
1  #include "mcc_generated_files/mcc.h"
2
3  typedef int16_t fix;
4  typedef int32_t fix2;
5
6  #define FRAC 12
7
8  #define fp2float(x) (float)x/((float)(1<<FRAC))
9  #define float2fp(x) (fix)(x*((float)(1<<FRAC))+0.5)
10
11
12 #define fp_sum(x, y)  ((x)+(y))
13 #define fp_sub(x, y)  ((x)-(y))
14 #define fp_mul(x, y)  (fix)(((fix2)(x)*(fix2)(y))>>FRAC)
15 #define fp_div(x, y)  (fix)(((fix2)(x)<<FRAC)/(y))
16
17 void main(void)
18 {
19     SYSTEM_Initialize();
20
21     float f_sum, f_sub, f_mul, f_div;
22     fix sum, sub, mul, div;
23
24     uint16_t time_float = 0, time_fp = 0;
25     while (1)
26     {
27         float a = (float)(rand()%1000)/1000.0;
28         float b = (float)(rand()%1000)/1000.0;
```

```
29          fix fp_a = float2fp(a);
30          fix fp_b = float2fp(b);
31
32          TMR1_Reload();
33          TMR1_StartTimer();
34          sum = fp_sum(fp_a,fp_b);
35          sub = fp_sub(fp_a,fp_b);
36          mul = fp_mul(fp_a,fp_b);
37          TMR1_StopTimer();
38          time_fp = TMR1_ReadTimer();
39
40          TMR1_Reload();
41          TMR1_StartTimer();
42          f_sum = a+b;
43          f_sub = a-b;
44          f_mul = a*b;
45          TMR1_StopTimer();
46          time_float = TMR1_ReadTimer();
47
48          printf("-----------------------------\n");
49          printf("Broj a je %f, broj b je %f\n", a, b);
50          printf("Float proteklo vreme %.2f ns\n", (float)time_float*15.625);
51          printf("Float sum %f\n",f_sum);
52          printf("Float sub %f\n",f_sub);
53          printf("Float mul %f\n",f_mul);
54          printf("-----------------------------\n");
55          printf("Fixed point vreme %.2f ns\n", (float)time_fp*15.625);
56          printf("Broj a je %d, broj b je %d\n", fp_a, fp_b);
57          printf("Fixed point sum %f\n",fp2float(sum));
58          printf("Fixed point sub %f\n",fp2float(sub));
59          printf("Fixed point mul %f\n",fp2float(mul));
60          printf("-----------------------------\n");
61          b+=1.0;
62          a+=1.1;
63          __delay_ms(3000);
64          }
65 }
```

Primer 2: *FIR NF filter - aritmetika sa pokretnim zarezom*

U ovom primeru data je realizacija FIR niskopropusnog filtera podataka očitanih sa akcelerometra MPU9250. Ovaj senzorski modul je implementiran u poglavlju rada sa *I2C* modulom odakle će se koristiti funkcije za inicijalizaciju i očitavanje podataka sa akcelerometra.

U listinzima koda 2 i 3 dat je kod za implementaciju FIR filtera. U heder fajlu definisana je konstanta *FIR_FILTER_LENGTH* koja predstavlja broj odbiraka odnosno broj koeficijenata koji figurišu u proračunu izlaza prilikom filtriranja. Zatim je generisan tip strukture *FIRfilter_t* koji opisuje digitalni FIR filter i sadrži: niz x koji sadrži *FIR_FILTER_LENGTH* najsvežijih odbiraka, *index* promenljivu koja čuva poziciju najsvežijeg (poslednjeg) odbirka u cirkularnom baferu i niz h koji čuva koeficijente filtera. Nakon čega su definisani prototipovi funkcija koje su implementirane u *FIRfilter.c* fajlu. Funkcija *FIRfilter_Init* prima kao parametar pokazivač na filter definisan tipom strukture *FIRfilter_t* kao i pokazivač na koeficijente filtra koji predstavljaju niz float koeficijenata. Ova funkcija popunjava vrednosti svih odbiraka nulama, a zatim promenljivu *index* postavlja na početak cirkularnog bafera i izlaznu vrednost filtera postavlja na nulu. Na kraju se korišćenjem funkcije *memcpy* kopiraju prosleđeni koeficijenti filtera *coeff* u niz h prosleđene strukture *filter*. Funkcija *FIR-*

Filter_FilterSignal prima kao parametar pokazivač na filter kao i vrednost odbirka x. Na početku se izlaz filtera postavlja na nulu jer će se uz pomoć ove funkcije izračunati vrednost na izlazu filtera. Zatim se prosleđeni odbirak smešta u cirkularni bafer x na mesto *index*, a kako bi se izvršila konvolucija neophodno je poznavati trenutnu poziciju najsvežijeh odbirka. Zbog toga se pozicija najsvežijeg odbirka smešta u promenljivu j. Nakon toga se promenljiva *index* uvećava za jedan kako bi u sledećem pozivu funkcije bio prepisan najstariji odbirak. Pošto je cirkularni bafer ograničen na dužinu *FIR_FILTER_LENGTH* neophodno je da ako *index* dosegne vrednost *FIR_FILTER_LENGTH* vratiti ga na prvi element. Na kraju *for* petlja prolazi kroz sve elemente koefcijenata i odbiraka smeštenih u filtru, međusobno ih množi i akumulira u vrednosti izlaza y. Važno je istaći da se vrednosti koeficijenata množe sa vrednostima obiraka u invertovanom redosledu (počevši od najsvežijeg ka najstarijem odbirku), zbog čega se ako promenljiva j dosegne vrednost nula ona vraća na kraj niza (*FIR_FILTER_LENGTH*-1). Po završetku izvršavanja petlje promenljiva y u strukturi *filter* sadrži vrednost izlaza filtracije ulaznih podataka.

Listing 2: FIRfilter.h fajl

```
#ifndef FIR_FILTER_H
#define FIR_FILTER_H

#include <xc.h>
#include <stdint.h>


#define FIR_FILTER_LENGTH 25

typedef struct
{
    float x[FIR_FILTER_LENGTH];
    uint8_t index;
    float y;
    float h[FIR_FILTER_LENGTH];
}FIRfilter_t;

void FIRfilter_Init(FIRfilter_t *filter, const float *coeff);
void FIRFilter_FilterSignal(FIRfilter_t *filter, float x);

#endif
```

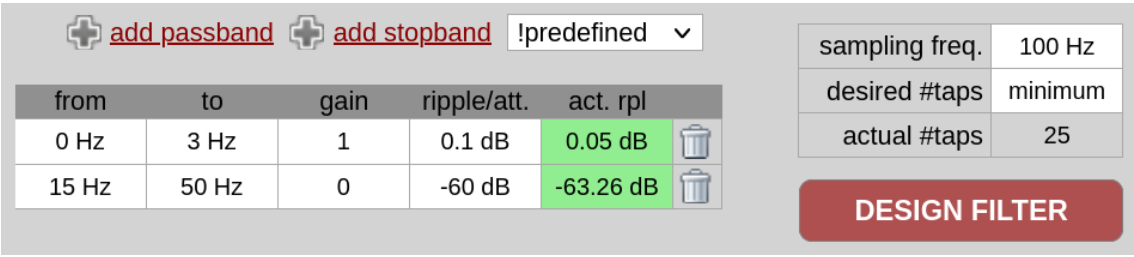
Listing 3: FIRfilter.c fajl

```
#include <xc.h>
#include <string.h>
#include "FIRfilter.h"

void FIRfilter_Init(FIRfilter_t *filter, const float *coeff)
{
    for(uint8_t i=0; i<FIR_FILTER_LENGTH; i++)
    {
        filter->x[i] = 0;
    }
        filter->index = 0;
        filter->y = 0;

        memcpy(filter->h, coeff, FIR_FILTER_LENGTH*sizeof(float));
}

void FIRFilter_FilterSignal(FIRfilter_t *filter, float x)
{
    filter->y = 0;

```

```
21      filter->x[filter->index] = x;
22
23      uint8_t j = filter->index;
24
25      filter->index++;
26      if(filter->index == FIR_FILTER_LENGTH)
27      {
28          filter->index = 0;
29      }
30
31      for(uint8_t i=0; i<FIR_FILTER_LENGTH; i++)
32      {
33          if(j>0)
34          {
35              j--;
36          }
37          else
38          {
39              j = FIR_FILTER_LENGTH-1;
40          }
41
42          filter->y += filter->h[i]*filter->x[j];
43      }
44  }
```

U glavnom programu datog listingom koda 4, uključene su opisane biblioteke za korišćenje FIR filtera, kao i bilbioteka za rad sa *MPU9250* senzorom (u poglavlju koje se bavi radom sa *I2C* modulom). Kako bi se signal sa akcelerometra odabirao ekvidistantno, realizovan je prekid tajmerskog modula na svakih 10 ms koji promenljivu *flag* postavlja na *true*. Tako je generisana frekvencija odabiranja od 100 Hz.

Kako bi se generisali koeficijenti FIR filtera korišćen je *T filter* alat koji se može pronaći na linku `http://t-filter.engineerjs.com/`. U donjem levom uglu alata nalaze se podešavanja filtra gde se pomoću definisanja propusnog opsega *passband* i nepropusnog opsega *stopband* može dobiti željena prenosna karakteristika filtera. Na slici 15 prikazan su podešavanja filtera za ovaj primer.

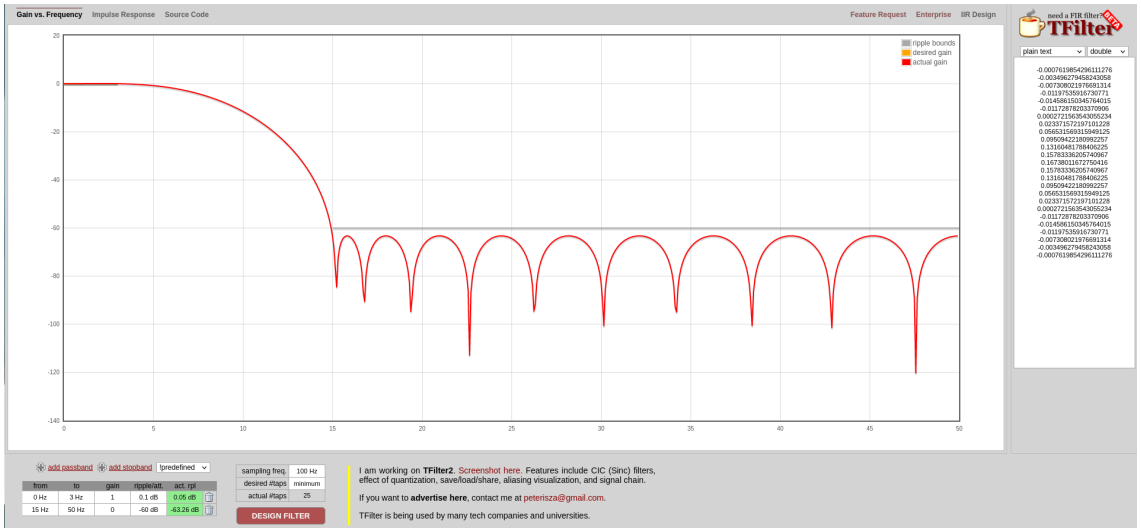


Slika 15: Tfilter alat - podešavanje koeficijenata

U opsegu od 0 Hz do 3 Hz definisan je propusni opseg sa pojačanjem 1 (0 dB) i definisana je varijacija pojačanja u propusnom opsegu do maksimalno 0,1 dB. Takođe, definisan je nepropusni opseg od 15 Hz do 50 Hz podešen je nepropusni opseg bez pojačanja gde se na željenom opsegu očekuje slabljenje od barem -60 dB. U polju *sampling freq.* definisana je frekvencija odabiranja (100 Hz) i u polju *desired #taps* postavljeno je da filter ima minimalno koeficijenata kako bi se ispunili definisani uslovi. Nakon pritiska na polje *design filter*, projektovan je filter gde se u poljima *act.rpl.* dobija varijacija pojaćanja u propusnom opsegu od 0,05 dB, a slabljenje

u nepropusnom opsegu od -63,26 dB. Minimalan broj koeficijenata dat je u polju *actual #taps* i iznosi 25.

Na slici 16 prikazan je bodeov diagram projektovanog FIR filtra, za definisane uslove. U desnom delu dati su koeficijenti FIR filtra. Podrazumevano koeficijenti su dati u formatu sa pokretnom tačkom, ali se pomoću padajućeg menija može umesto *double* odabrati *int*, čime se bira format sa nepokretnom tačkom. U slučaju korišćenja *fixed-point* formata pojaviće se i polje za unos preciznosti odnosno za definisanje broja bita koji se koriste za prikaz frakcionog dela.



Slika 16: Tfilter alat - projektovani filter

Po završetku projektovanja filtera podaci se smeštaju u promenljivu *coeff* koja će se kao parametar proslediti funkciji *FIRfilter_Init* kako bi se inicijalizovao filter. Pre inicijalizacije filtera izvršena je inicijalizacija senzorskog modula MPU9250, pozivom funkcije *MPU9250_init*.

Po očitavanju podataka sa akcelerometra pozivom funkcije *MPU9250_read_accel*, poziva se funkcija *FIRFilter_FilterSignal* kojoj se prosleđuje podatak ubrzanja po x osi. Nakon filtracije podataka korišćenjem *memcpy* funkcije *float* brojevi se mapiraju na odgovarajući četvrobajtni niz kako bi se prosledili u formatu pogodnom za vizualizaciju u *Data Visualizer* alatu. Prvo se šalju nefiltrirani, a zatim se šalju filtrirani podaci. Na kraju čeka se na prekid kako bi se uzeo novi odbirak sa akcelerometra.

Listing 4: Glavni program

```
1  #include "mcc_generated_files/mcc.h"
2  #include <stdio.h>
3  #include <string.h>
4  #include "FIRfilter.h"
5  #include "MPU9250.h"
6
7  volatile bool flag;
8
9  void main(void)
10 {
11     SYSTEM_Initialize();
12     INTERRUPT_GlobalInterruptEnable();
```

```
13      const float coeff[] = {
14          -0.0007619854296111843,
15          -0.0034962794582430994,
16          -0.007308021976691337,
17          -0.011975359167307702,
18          -0.014586150345763979,
19          -0.011728782033709,
20          0.0002721563543055868,
21          0.023371572197101283,
22          0.05653156931594916,
23          0.09509422180992257,
24          0.13160481788406225,
25          0.1578333620574096,
26          0.16738011672750414,
27          0.1578333620574096,
28          0.13160481788406225,
29          0.09509422180992257,
30          0.05653156931594916,
31          0.023371572197101283,
32          0.0002721563543055868,
33          -0.011728782033709,
34          -0.014586150345763979,
35          -0.011975359167307702,
36          -0.007308021976691337,
37          -0.0034962794582430994,
38          -0.0007619854296111843
39      };
40
41      MPU9250_t akcelerometar;
42      FIRfilter_t fir_NF_filter;
43
44      uint8_t accelX[4];
45      uint8_t accelX_filt[4];
46
47      MPU9250_init();
48      FIRfilter_Init(&fir_NF_filter, coeff);
49
50      while (1)
51      {
52          MPU9250_read_accel(&akcelerometar);
53          FIRFilter_FilterSignal(&fir_NF_filter, akcelerometar.ax);
54
55          memcpy(accelX, &akcelerometar.ax, 4);
56          memcpy(accelX_filt, &fir_NF_filter.y, 4);
57
58          EUSART1_Write(0x03);
59          EUSART1_Write(accelX[0]);
60          EUSART1_Write(accelX[1]);
61          EUSART1_Write(accelX[2]);
62          EUSART1_Write(accelX[3]);
63
64          EUSART1_Write(accelX_filt[0]);
65          EUSART1_Write(accelX_filt[1]);
66          EUSART1_Write(accelX_filt[2]);
67          EUSART1_Write(accelX_filt[3]);
68          EUSART1_Write(0xFC);
69          while(!flag);
70          flag = false;
71      }
72  }
```

Primer 3: *FIR NF filter - aritmetika sa nepokretnim zarezom*

Ovaj primer je identičan prethodnom primeru sa razlikom da umesto aritmetike sa pokretnim koristi aritmetiku sa nepokretnim zarezom.

U listinzima koda 5 i 6 dat su funkcije za inicijalizaciju filtra i filtriranje podataka. Razlika je u definisanoj strukturi čiji su podaci *fix* umesto *float* tipa. Tip *fix* predstavlja celobrojni tip pomoću kog je realizovana aritmetika sa nepokretnim zarezom. Takođe, funkcijama *FIRfilter_Init* i *FIRFilter_FilterSignal* se prosleđuju parametri tipa *fix*. Umesto standardnog množenja i sabiranja u funkciji *FIRFilter_FilterSignal* su korišćene funkcije *fp_sum* i *fp_mul* kako bi se sabirali i množili brojevi sa nepokretnim zarezom.

Listing 5: FIRfilter.h fajl

```
 #ifndef FIR_FILTER_H
#define FIR_FILTER_H

#include <xc.h>
#include <stdint.h>
#include "fixed_point.h"

#define FIR_FILTER_LENGTH 25

typedef struct
{
    fix x[FIR_FILTER_LENGTH];
    uint8_t index;
    fix y;
    fix h[FIR_FILTER_LENGTH];
}FIRfilter_fp_t;

void FIRfilter_Init_fp(FIRfilter_fp_t *filter, fix *coeff);
void FIRFilter_FilterSignal_fp(FIRfilter_fp_t *filter, fix x);


#endif
```

Listing 6: FIRfilter.c fajl

```
#include <xc.h>
#include <string.h>
#include "FIRfilter_fp.h"

void FIRfilter_Init_fp(FIRfilter_fp_t *filter, fix *coeff)
{
    for(uint8_t i=0; i<FIR_FILTER_LENGTH; i++)
    {
        filter->x[i] = 0;
    }
        filter->index = 0;
        filter->y = 0;

        memcpy(filter->h, coeff, FIR_FILTER_LENGTH*sizeof(fix));
}

void FIRFilter_FilterSignal_fp(FIRfilter_fp_t *filter, fix x)
{
    filter->y = 0;

    filter->x[filter->index] = x;

    uint8_t j = filter->index;

    filter->index++;
    if(filter->index == FIR_FILTER_LENGTH)
    {
        filter->index = 0;
    }

```

```
31      for(uint8_t i=0; i<FIR_FILTER_LENGTH; i++)
32      {
33          if(j>0)
34          {
35              j--;
36          }
37          else
38          {
39              j = FIR_FILTER_LENGTH-1;
40          }
41           filter->y = fp_sum(filter->y, fp_mul(filter->h[i],filter->x[j]));
42      }
43  }
```

U listingu koda 7, a na osnovu prvog primera, date su osnovne aritmetičke opeacije brojevima sa nepokretnom tačkom. U ovom slučaju korišćena je Q3.12 aritmetika.

Listing 7: Fajl fixed_point.h

```
1  #ifndef FIXED_POINT_H
2  #define FIXED_POINT_H
3
4  #include <xc.h>
5
6  typedef int16_t fix;
7  typedef int32_t fix2;
8
9  #define FRAC 12
10
11 #define fp2float(x) (float)x/((float)(1<<FRAC))
12 #define float2fp(x) (fix)(x*((float)(1<<FRAC))+0.5)
13
14
15 #define fp_sum(x, y)  ((x)+(y))
16 #define fp_sub(x, y)  ((x)-(y))
17 #define fp_mul(x, y)  (fix)(((fix2)(x)*(fix2)(y))>>FRAC)
18 #define fp_div(x, y)  (fix)(((fix2)(x)<<FRAC)/(y))
19
20 #endif
```

U listingu koda glavnog programa 8 prosleđeni brojevi su sada definisani u aritmetici sa nepokretnim zarezom. Ovi podacise prosleđuju funkciji za inicijalizaciju filtera. Pošto funkcija *MPU9250_read_accel* očitava podatke sa akcelerometra i skladišti ih kao *float* neophodno ih je konvertovati u *fixed-point* pozivom funkcije *float2fp*. Ovaj podatak se dalje prosleđuje filteru. Kako bi se podatak mogao prikazati u *Data Visualizer* programu neophodno je broj u aritmetici sa nepokretnim zarezom konvertovati u *float* i to je izvršeno pozivom funkcije *fp2float*. Na posletku se, isto kao i uprethodnom primeru, podaci šalju u standardnom formatu.

Listing 8: Glavni program

```
1  #include "mcc_generated_files/mcc.h"
2  #include <stdio.h>
3  #include <string.h>
4  #include "FIRfilter_fp.h"
5  #include "MPU9250.h"
6  #include "fixed_point.h"
7
8  volatile bool flag;
9
10 void main(void)
11 {
12      SYSTEM_Initialize();
13      INTERRUPT_GlobalInterruptEnable();
14      fix coeff[] = {
```

```
15          -3,
16          -14,
17          -30,
18          -49,
19          -60,
20          -48,
21          1,
22          96,
23          232,
24          390,
25          539,
26          646,
27          686,
28          646,
29          539,
30          390,
31          232,
32          96,
33          1,
34          -48,
35          -60,
36          -49,
37          -30,
38          -14,
39          -3
40      };
41
42      MPU9250_t akcelerometar;
43      FIRfilter_fp_t fir_NF_filter_fp;
44
45      uint8_t accelX[4];
46      uint8_t accelX_filt[4];
47
48      MPU9250_init();
49      FIRfilter_Init_fp(&fir_NF_filter_fp, coeff);
50
51      while (1)
52      {
53          MPU9250_read_accel(&akcelerometar);
54          FIRFilter_FilterSignal_fp(&fir_NF_filter_fp, float2fp(akcelerometar.ax));
55
56          float tmp = fp2float(fir_NF_filter_fp.y);
57          memcpy(accelX, &akcelerometar.ax, 4);
58          memcpy(accelX_filt, &tmp, 4);
59
60
61          EUSART1_Write(0x03);
62          EUSART1_Write(accelX[0]);
63          EUSART1_Write(accelX[1]);
64          EUSART1_Write(accelX[2]);
65          EUSART1_Write(accelX[3]);
66
67          EUSART1_Write(accelX_filt[0]);
68          EUSART1_Write(accelX_filt[1]);
69          EUSART1_Write(accelX_filt[2]);
70          EUSART1_Write(accelX_filt[3]);
71          EUSART1_Write(0xFC);
72          while(!flag);
73          flag = false;
74
75      }
76  }
```