SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1047

RAZVOJ PROGRAMSKE POTPORE UGRADBENOG RAČUNALNOG SUSTAVA POMOĆU MATLAB EMBEDDED CODER PAKETA

Deni Petak

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 11. ožujka 2014.

DIPLOMSKI ZADATAK br. 1047

Pristupnik: **Deni Petak (0036452847)**
Studij: Elektrotehnika i informacijska tehnologija
Profil: Elektroničko i računalno inženjerstvo

Zadatak: **Razvoj programske potpore ugradbenog računalnog sustava pomoću Matlab Embedded Coder paketa**

Opis zadatka:

U okviru diplomskog rada potrebno je istražiti mogućnost implementacije programske potpore za ugradbeno računalo STM32F4 Discovery korištenjem paketa Matlab Embedded Coder. Demonstrirati mogućnost upravljanja ulaznih i izlaznih jedinica ugradbenog računala korištenjem Simulink blokova. Istražiti mogućnost izvedbe algoritama digitalne obrade signala u Simulink okruženju i automatsko prevođenje modela u izvorni kod ugradbenog računala. Analizirati učinkovitost realiziranog programskog koda sa stanovišta brzine izvođenja i zauzeća memorije. Usporediti tako dobiven programski kod s ručno napisanim kodom u programskom jeziku C za nerekurzivne i rekurzivne digitalne filtre ili blokovske transformacije kao što su DFT ili DCT, za podatke s cjelobrojnim zapisom. Za detaljnije informacije obratiti se mentoru.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 30. lipnja 2014.

Mentor:
Prof.dr.sc. Davor Petrinović

Djelovođa:
Izv.prof.dr.sc. Dražen Jurišić

Predsjednik odbora za diplomski rad profila:
Prof.dr.sc. Mladen Vučić

Sadržaj

Uvod

MATLAB (***mat****rix* ***lab****oratory*) je programski alat visoke razine i interaktivna okolina za numeričko računanje, vizualizaciju i programiranje. Koristeći MATLAB moguće je analizirati podatke, razvijati algoritme i kreirati modele i aplikacije. Ima široki spektar raznovrsnih aplikacija koje se mogu koristiti kod obrade signala, slike i videa, sustava kontrole, testiranja i mjerenja, no nama je zanimljiv zbog MATLAB Embedded Coder paketa koji podržava automatsko prevođenje Simulink modela u izvorni kod ugradbenog računala STM32F4 Discovery.

Pomoću Simulink biblioteke *Target Support Package – STM32F4xx Adapter* moguće je jednostavno razviti aplikaciju za STM32F4 Discovery mikrokontroler. Biblioteka ima pet blokova, prilagođenih za STM32F4 mikrokontroler: ADC, GPIO, INTERRUPTS, TIMERS i USART.

U ovom radu bit će implementirane dvije verzije DFT transformacije i nerekurzivnih (FIR) filtara. Jedna verzija razvijena je u MATLAB programskom alatu pomoću Simulink modela i Embedded Coder paketa, dok je druga ručno napisana u programskom jeziku C/C++. Obje verzije koriste 16-bitne podatke u cjelobrojnom zapisu. Analizirat ćemo dvije verzije DFT transformacije i FIR filtara sa stanovišta brzine izvođenja, broja ciklusa procesora i zauzeća memorije mikrokontrolera.

1. Diskretna Fourierova transformacija

U matematici, Diskretna Fourierova transformacija (DFT) je specifična vrsta Fourierove transformacije koja se koristi kod Fourierove analize. Pomoću Fourierove transformacije moguće je transformirati funkciju iz vremenske domene u frekvencijsku domenu i obrnuto. Zbog kontinuiranih i diskretnih te periodičnih i aperiodičnih funkcija imamo četiri vrste Fourierovih transformacija. Redom to su vremenski kontinuirana Fourierova transformacija (CTFT), vremenski kontinuiran Fourierov red (CTFS), vremenski diskretna Fourierova transformacija (DTFT) i vremenski diskretan Fourierov red (DTFS). Diskretna Fourierova transformacija (DFT) ima mnogo sličnih svojstava s vremenski diskretnim Fourierovim redom koji se koristi za prikaz diskretnih periodičnih funkcija. Kao ulaz u DFT funkciju dovodi se diskretan signal koji je limitiran u trajanju, ima konačan niz diskretnih brojeva. Takav ulaz, niz diskretnih brojeva, često je dobiven otipkavanjem iz kontinuirane funkcije ili signala (zvuk, slika i slično). Da bi DFT pravilno izračunao spektar zadanog signala potrebno je imati barem jednu periodu ulaznog signala. Iz istog razloga, inverzna DFT ne može reproducirati cijelu vremensku domenu, osim ako je ulaz periodičan.

Ulaz u DFT je konačna sekvenca realnih i kompleksnih brojeva što DFT čini idealnim za procesiranje informacija spremljenih u računalu ili mikrokontroleru. Ima veliku primjenu u digitalnoj obradi signala u područjima analiziranja frekvencijskog spektra uzrokovanog signala, pri rješavanju parcijalnih diferencijalnih jednadžbi i ostalih operacija poput konvolucije i množenja velikih cijelih brojeva. Ključni čimbenik za izračunavanje DFT u realnom vremenu je efikasan algoritam brza Fourierova transformacija (FFT).

FFT algoritam se toliko često koristi za izračun DFT-a da se pojam FFT koristi kao naziv za DFT u praktičnim primjenama. Formalno postoji jasna razlika, DFT se odnosi na matematičku transformaciju ili funkciju, bez obzira kako se izračunava, dok se FFT odnosi na specijalnu obitelj algoritma za izračun DFT-a.

1.1. Povijest Fourierove transformacije

Jean Baptiste Joseph Fourier (21.03.1768. – 16.05.1830.) bio je francuski matematičar i fizičar rođen u Auxerre i poznat po pokretanju istrage o Fourierovom redu i njegove primjene kod problema s prijenosom topline i vibracija. Fourierova transformacija nazvana je njemu u čast. Fourier je zaslužan i za otkriće efekta staklenika.

Vrlo brzo, u dobi od trinaest godina, Fourier pokazuje veliki interes za matematiku u školi École Royale Militaire u Auxerre. Godine 1783. dobiva prvu nagradu za svoj doprinos o Bossutovoj Mécanique en général. Od 1790. godine postaje profesor na Benediktinskom fakultetu, École Royale Militaire u Auxerre. Ideali francuske revolucije upleli su ga u politiku zbog čega je nekoliko puta bio u opasnosti za svoj život. Fourier je 1795. godine postao profesor u Collège de France i započeo dodatno matematičko istraživanje. Imao je odlične odnose s ostalim umovima tog vremena poput Lagrangea, Laplacea i Mongea. Napoleon je 1798. u pohodu na Egipat odlučio Fourieru ponuditi posao znanstvenog savjetnika. Dio te ekspedicije bili su i Monge i Malus.

U Grenobleu 1804. Fourier je započeo važan matematički rad o teoriji topline kojeg je 21. prosinca 1807. godine u Pariškom institutu objavio pod nazivom Širenje topline u čvrstim tijelima (*On the Propagation of Heat in Solid Bodies*). U komisijskom odboru bili su Lagrange, Laplace, Monge i Lacroix, iako je Fourierov rad i danas veoma cijenjen u to vrijeme je izazvao dosta kontroverzi. Komisijski odbor nije bio zadovoljan radom. Prvu dopunu donose Lagrange i Laplace 1808., proširuju Fourierovu funkciju kao trigonometrijski razvoj funkcije, ono što danas zovemo Fourierov red. Dok je drugi prigovor bio od strane Biota protiv Fourierove derivacijske jednadžbe prijenosa topline. Fourier nije napravio osvrt na Biotov rad iz 1804. na temu o prijenosu topline, no kasnije se pokazalo da su u Biotovom rad neke tvrdnje bile pogrešne. Laplace i kasnije Poisson imali su slične primjedbe. Institut je ipak 1811. godine odlučio dati Fourieru nagradu za doprinos u matematici. Iako izvješće nije u potpunosti bilo favorizirano, nije bilo izlika da se Fourierov rad ne objavi u Parizu. Najviše su zamjerili što Fourier nije imao matematičku strogost, koju su radu kasnije dali Dirichlet i Riemann.

1.2. Fourierov red

U ovom poglavlju (Krnić, 2010) razlagat ćemo o trigonometrijskom razvoju funkcije $f(x)$ definirane na nekom intervalu, kao na primjer $-\pi \leq x \leq \pi$. Trigonometrijski razvoj funkcije odnosno Fourierov red je

$$(x) = a_0 + \sum_{k=1}^{\infty} (a_k \cos kx + b_k \sin kx), \tag{1.1}$$

pri čemu ta suma može biti konačna i beskonačna.

1.2.1. Računanje Fourierovog reda

Kako bi izračunali Fourierove koeficijente iskorištavamo relacije ortogonalnosti i tako općeniti izraz, relacija (1.1), pomnožimo s obje strane funkcijom $\cos\frac{nx}{\pi}$ i integriramo da bi našli koeficijent a_n :

$$\frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\, dx = \frac{1}{\pi}\int_{-\pi}^{\pi}\left[a_0 + \sum_{k=1}^{\infty}(a_k \cos \mathrm{kx} + b_k \sin \mathrm{kx})\right]\cos nx\ \ dx,$$

korištenjem relacija ortogonalnosti dobivamo da je

$$\frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\, dx = a_n,\ \ n \geq 1.$$

Kako bi dobili koeficijent b_n množimo relaciju (1.1) sa $\sin\frac{nx}{\pi}$ i integriramo te dobivamo

$$\frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx\, dx = b_n,\ \ n \geq 1,$$

preostali koeficijent a_0 dobivamo tako što integriramo obje strane relacije

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)dx = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[a_0 + \sum_{k=1}^{\infty}(a_k \cos \mathrm{kx} + b_k \sin \mathrm{kx})\right]\ dx,$$

dobimo da je koeficijent a_0

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x) = \frac{1}{2\pi}\int_{-\pi}^{\pi} a_0\, dx = a_0.$$

Koeficijenti a_n i b_n su koeficijenti Fourierove funkcije $f(x)$.

Kao što je poznato funkcija $f(x)$ je periodična i tada se može razviti u Fourierov red, no što ako je funkcija neperiodična? Tada tu neperiodičnu funkciju pretvorimo u periodičnu koja se ponavlja beskonačno puta i tada interval postaje period te funkcije.

1.2.2. Fourierov red parnih i neparnih funkcija

Važno je spomenuti svojstva parnih i neparnih funkcija te tako dobivamo svojstva umnoška dviju funkcija s obzirom na parnost:

$$parna \cdot parna = parna$$

$$parna \cdot neparna = neparna$$

$$neparna \cdot neparna = parna.$$

Neka je a nenegativan realan broj tada vrijede ove jednadžbe

$$\int_{-a}^{a} f(x)dx = 2\int_{0}^{a} f(x)dx \ , gdje\ je\ f(x)\ parna\ funkcija,$$

$$\int_{-a}^{a} f(x)dx = 0 \ , gdje\ je\ f(x)\ neparna\ funkcija.$$

Za isti a na intervalu od $[-a, a]$, gdje je $f(x)$ parna funkcija , onda njezin Fourierov red sadrži samo kosinus funkcije

$$f(x) = a_0 + \sum_{k=1}^{\infty} a_k cos\frac{k\pi x}{a},$$

gdje su koeficijenti

$$a_0 = \frac{1}{a}\int_{0}^{a} f(x)dx \quad i \quad a_k = \frac{2}{a}\int_{0}^{a} f(x)\cos\frac{k\pi x}{a}dx.$$

Fourierov red neparne funkcije $f(x)$, za isti a na intervalu od $[-a, a]$, sadrži samo sinus funkcije

$$f(x) = \sum_{k=1}^{\infty} b_k sin\frac{k\pi x}{a},$$

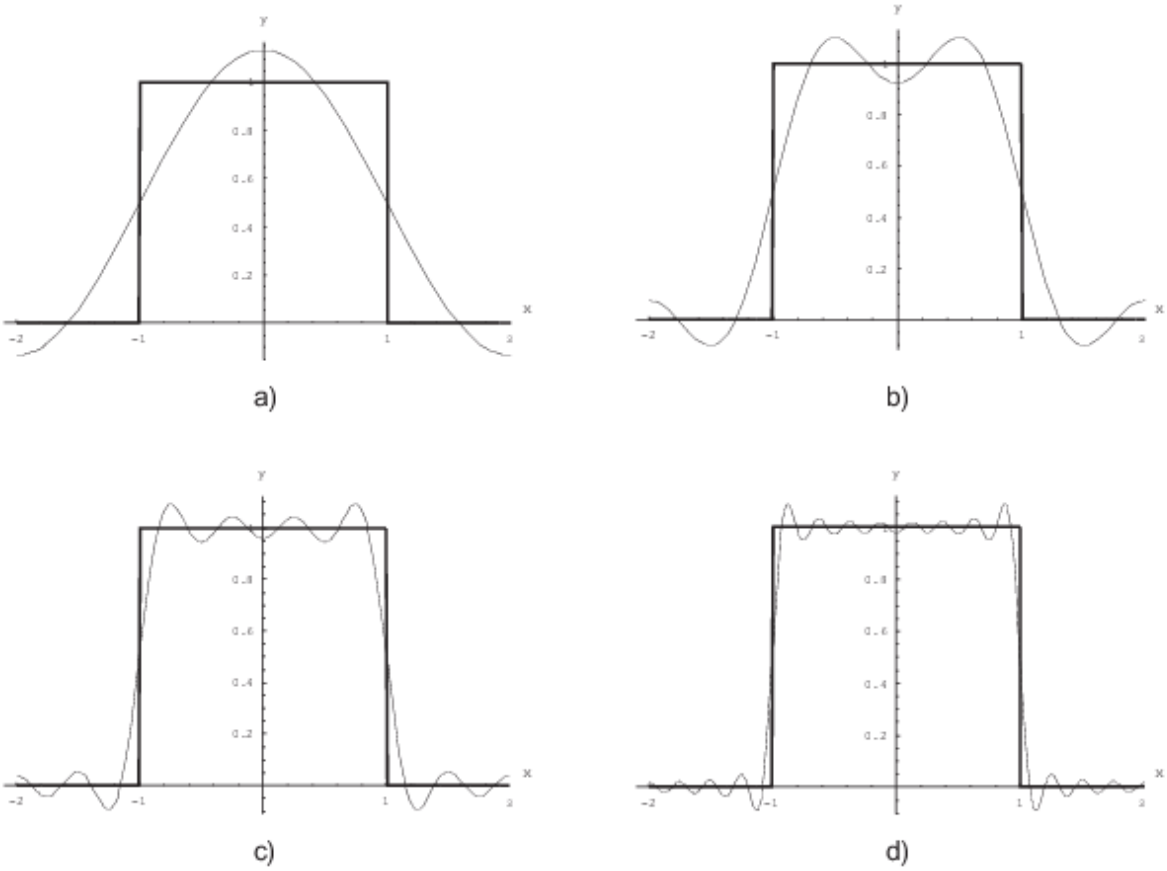
gdje je koeficijent

$$b_k = \frac{2}{a}\int_0^a f(x)\sin\frac{k\pi x}{a}\,dx.$$

Uz pretpostavku da funkcija definirana na intervalu $[0, a]$ moguće je parno i neparno proširenje funkcije i tako si olakšati računanje Fourierovog reda.

Slika 1.1 prikazuje tzv. *rectangle* funkciju pomoću grafa parcijalnih suma različitih brojeva članova Fourierovog reda.

$$f(x) = \begin{cases} 1, & x \in (-1,1), \\ 0, & inače. \end{cases}$$



Slika 1.1 Prikaz aproksimacije tzv. *rectangle* funkcije na intervalu [-2, 2] s pomoću Fourierovog reda za: a) n=2, b) n=4, c) n=8, d) n=16

1.2.3. Kompleksni oblik Fourierovog reda

Često se Fourierov red funkcije zbog zgodnijeg izraza zapisuje pomoću kompleksnih eksponencijalnih funkcija e^{inx}, $n \in \boldsymbol{Z}$.

Prisjetimo se, kompleksna eksponencijalna funkcija definirana je izrazom $e^{it} = \cos t + i \sin t$.

Ako je f realna funkcija, onda se oblik njezinog Fourierovog reda može izvesti iz kompleksnog Fourierovog reda i obrnuto. U kompleksom Fourierovom redu pozitivne i negativne članove možemo razdvojiti

$$f(t) = \sum_{n=-\infty}^{-1} \alpha_n e^{int} + \alpha_0 + \sum_{n=1}^{\infty} \alpha_n e^{int}, \tag{1.2}$$

pri tome je

$$\alpha_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-int} dt.$$

Ako je f realna funkcija, onda je

$$\alpha_{-n} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{int} dt = \overline{\frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-\imath nt} dt} = \overline{\alpha_n},$$

prema tome, relacija (1.2) je oblika

$$f(t) = \alpha_0 + \left(\sum_{n=1}^{\infty} \alpha_n e^{int} \right) + \left(\overline{\sum_{n=1}^{\infty} \alpha_n e^{\imath nt}} \right),$$

odnosno,

$$f(t) = \alpha_0 + \mathrm{Re} \left(\sum_{n=1}^{\infty} \alpha_n e^{int} \right),$$

zato jer je $z + \bar{z} = 2 \,\mathrm{Re}\, z$,

veza između kompleksnih koeficijenata α_n i realnih koeficijenata a_n , b_n, očito je

$$\alpha_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) dt = a_0.$$

Iz čega slijedi, ako je $n \geq 1$, onda je

$$\alpha_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) e^{-int} dt = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) (dosn\, nt - i \sin nt) dt = \frac{1}{2} (a_n - i b_n).$$

Iz jednakosti dobivamo

$$f(t) = \alpha_0 + 2\mathrm{Re} \left(\overline{\sum_{n=1}^{\infty} \alpha_n e^{\imath nt}} \right)$$

$$= \alpha_0 + \mathrm{Re}\left(\sum_{n=1}^{\infty}(a_n - ib_n)(\cos nt + i\sin nt)\right)$$

$$= \alpha_0 + \mathrm{Re}\left(\sum_{n=1}^{\infty}(a_n \cos nt + b_n \sin nt)\right).$$

Što predstavlja realni oblik Fourierovog reda funkcije f.

1.3. Fourierova transformacija

Fourierova transformacija se može shvatiti kao neprekinuti oblik Fourierovog reda. Promatrali smo kako razviti periodičnu funkciju perioda $2a$ definiranu na intervalu $[-a, a]$ u Fourierov red. S druge strane (Krnić, 2010), Fourierova transformacija razlaže signal, definiran na beskonačnom vremenskom intervalu, na komponente s frekvencijama λ, gdje λ može biti bilo koji realni ili kompleksni broj.

Fourierovu transformaciju možemo dobiti ako pustimo da broj l teži u beskonačnost, a da je Fourierov red funkcije definiran na intervalu $-l \leq x \leq l$. Pa tako imamo oblik

$$f(x) = \lim_{l\to\infty}\left[\sum_{n=-\infty}^{\infty}\frac{1}{2l}\int_{-l}^{l} f(t)e^{in\pi(x-t)/l}dt\right],$$

Na desnoj strani jednakosti nalazi se integralna suma odgovarajućeg integrala, zamjenjujemo $\lambda_n = \frac{n\pi}{l}$ i $\Delta\lambda = \lambda_{n+1} - \lambda_n = \frac{\pi}{l}$ i dobivamo

$$f(x) = \lim_{l\to\infty}\sum_{n=-\infty}^{\infty}\left[\frac{1}{2\pi}\int_{-l}^{l} f(t)e^{\lambda_n i(x-t)}dt\right]\Delta\lambda,$$

ako je

$$F_l(\lambda) = \frac{1}{2\pi}\int_{-l}^{l} f(t)e^{\lambda i(x-t)}dt,$$

tada prethodna jednakost poprima oblik

$$f(x) = \lim_{l\to\infty}\int_{-\infty}^{\infty} F_l(\lambda)d\lambda.$$

Kada $l \to \infty$, $F_l(\lambda)$ postaje integral $\frac{1}{2\pi}\int_{-\infty}^{\infty} f(t)e^{i\lambda(x-t)}dt$ zbog toga je

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left(\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\lambda t} dt \right) e^{i\lambda x} d\lambda,$$

i tako dobivao Fourierovu transformaciju funkcije f u kompleksnom obliku

$$\hat{f}(\lambda) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\lambda t} dt,$$

dok je Fourierova integralna formula ili inverz Fourierove transformacije

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\lambda) e^{-i\lambda x} d\lambda.$$

1.3.1. Primjena Fourierove reda i transformacije

Primjena Fourierove transformacije i reda koristi se kod digitalne obrade signala. Imamo četiri moguće kombinacije za vremenski kontinuirane ili diskretne te periodične ili neperiodične signale (Jeren, 2012). Svaki periodičan signal nastao je linearnom kombinacijom vremenski kontinuiranih sinusoida $A_k \cos(\omega_k t + \theta_k)$ što sugerira da se može razložiti (dekomponirati) na sinusoidne komponente koje ga sačinjavaju. Prema tome zaključujemo kako periodični signal može biti potpuno definiran frekvencijama ω_k, amplitudama A_k i fazama θ_k sinusoidnih komponenti koje ga sačinjavaju. Prepravljanjem i prilagođavanjem Fourierove transformacije i reda moguće je od takvih signala prikazati amplitudni i fazni spektar. Amplitudni spektar prikazuje amplitude sinusoidnih komponenata signala kao funkciju frekvencije ω. Fazni spektar predstavlja prikaz faze θ_k , u radijanima, kao funkciju frekvencije ω.

1.3.1.1 Fourierova transformacija vremenski diskretnih aperiodičnih signala (DTFT)

Za aperiodične diskretne signale $x(n), \forall n \in \boldsymbol{Z}$**,** definira se DTFT

$$\forall \omega \in \boldsymbol{R} \quad X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x(n) e^{-j\omega n},$$

i spektar takvog vremenski diskretnog signala je kontinuiran, zbog aperiodičnosti signala u vremenskoj domeni, i periodičan s periodom 2π, zbog diskretnosti signala u vremenskoj domeni. Zbog toga je frekvencijsko područje bilo kojeg diskretnog signala omeđeno na

$(-\pi, \pi)$ ili $(0,2\pi)$. Inverznu Fourierovu transformaciju vremenski diskretnih aperiodičnih signala možemo zapisati u obliku

$$\forall n \in \boldsymbol{Z} \quad x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X\left(e^{j\omega}\right) e^{j\omega n} d\omega.$$

Kako je spektar kompleksna funkcija DTFT-a možemo prikazati kao

$$X\left(e^{j\omega}\right) = \left|X(e^{j\omega})\right| e^{j\angle X(e^{j\omega})},$$

gdje su $\left|X(e^{j\omega})\right|$ amplitudni spektar, a $\angle X(e^{j\omega})$ fazni spektar.

1.3.1.2 Fourierova transformacija vremenski kontinuiranih aperiodičnih signala (CTFT)

Za aperiodične kontinuirane signale $x(t), \forall t \in \boldsymbol{R}$, definira se CTFT

$$\forall \Omega \in \boldsymbol{R} \qquad X(j\Omega) = \int_{-\infty}^{\infty} x(t) e^{-j\Omega t} dt.$$

Spektar vremenski kontinuiranog aperiodičnog signala je kontinuiran, zbor aperiodičnosti signala u vremenskoj domeni, i aperiodičan, zbog kontinuiranosti signala u vremenskoj domeni. Inverznu Fourierovu transformaciju možemo zapisati u obliku

$$\forall t \in \boldsymbol{R} \qquad x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\Omega) e^{j\Omega t}\, d\Omega.$$

Kao rezultat CTFT-a je Fourierov spektar ili jednostavno spektar signala $x(t)$ i kako je on kompleksna funkcija možemo pisati

$$X(j\Omega) = \ |X(j\Omega)| e^{j\angle X(j\Omega)},$$

gdje su $|X(j\Omega)|$ amplitudni spektar, a $\angle X(j\Omega)$ fazni spektar.

1.3.1.3 Fourierov red periodičnih vremenski diskretnih signala (DTFS)

Periodični vremenski diskretni signal $x(n), \forall \in \boldsymbol{Z}$, možemo također prikazati kao koeficijente Fourierovog reda, odnosno DTFS-om

$$\forall k \in \boldsymbol{Z} \quad X_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-jk\frac{2\pi}{N}n}.$$

Spektar diskretnog signala je periodičan, što vrijedi i za X_k koji je periodičan s osnovnim periodom N. Kako koeficijenti Fourierovog reda X_k omogućuju prikaz $x(n)$ u

frekvencijskoj domeni, tako da $X_k = |X_k|e^{j\angle X_k}$ predstavlja amplitudu i fazu vezanu uz frekvencijske komponente $e^{jk\frac{2\pi}{N}n} = e^{j\omega_k n}$, gdje je $\omega_k = k\frac{2\pi}{N}$. Frekvencijski spektar je diskretan i periodičan zbog svoje periodičnosti i diskretnosti signala u vremenskoj domeni. Inverz Fourierovog reda vremenski diskretnih periodičnih signala definiramo kao

$$\forall n \in \boldsymbol{Z} \quad x(n) = \sum_{k=0}^{N-1} X_k e^{jk\frac{2\pi}{N}n}.$$

1.3.1.4 Fourierov red periodičnih vremenski kontinuiranih signala (CTFS)

Periodični vremenski kontinuirani signal $x(t), \forall t \in \boldsymbol{R}$, moguće je prikazati pomoću Fourierovog reda tako što izračunamo koeficijente reda X_k

$$\forall k \in \boldsymbol{Z} \quad X_k = \frac{1}{T_0}\int_{T_0} x(t)e^{-jk\Omega_0 t}\,dt.$$

Spektar periodičnog signala je diskretan i aperiodičan, a frekvencijski razmak između uzoraka spektra je $\Omega_0 = \frac{2\pi}{T_0}$. Inverz Fourierovog reda periodičnih vremenskih kontinuiranih signala definiramo kao

$$\forall t \in \boldsymbol{R} \qquad x(t) = \sum_{k=-\infty}^{\infty} X_k e^{jk\Omega_0 t}.$$

Kao rezultat CTFS-a je Fourierov spektar signala $x(t)$ i kako je on kompleksna funkcija možemo pisati

$$X_k = |X_k|e^{j\angle X_k},$$

gdje su $|X_k|$ amplitudni spektar, a $\angle X_k$ fazni spektar.

1.4. Diskretna Fourierova transformacija (DFT)

Fourierova analiza omogućuje prikaz zastupljenosti signala u pogledu sinusnih valova. Ta zastupljenost omogućuje manipulaciju signala u velikom broju praktičnih primjena u znanosti i inženjerstvu. Iako je Fourierova transformacija bila važan matematički alat kod analize linearno vremenski nepromjenjivih (LTI) sustava, pojavom digitalnih računala i brzih numeričkih algoritama Fourierova transformacija postala je praktički najvažniji alat u

mnogim granama znanosti i inženjerstva. Fourierov prikaz signala je izuzetno koristan kod spektralne analize.

Diskretna Fourierova transformacija koristi se za veliku većinu signala koje nije moguće definirati matematičkim izrazom pa tako nije moguće primijeniti do sada izvedene transformacije. Signal i njegov spektar treba predstaviti njihovim uzorcima, odnosno očitati, što znači da će se očitani signal i njegov spektar periodički produžiti.

Za periodički diskretan signal $x(n)$, duljine L ($x(n) = 0\ za\ n < 0\ i\ n \geq L$) vrijedi par:

- diskretna Fourierova transformacija (DFT)

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad k = 0,1, \dots, N-1,$$

- inverzna diskretna Fourierova transformacija (IDFT)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn}, \quad n = 0,1, \dots, N-1.$$

Zbog jednostavnijeg zapisa često se zamjenjuje $e^{-j\frac{2\pi}{N}kn} = W_N^{nk}$.

Transformacija se uobičajeno koristi za prikaz konačnog niza duljine N. Diskretna Fourierova transformacija aperiodičnog niza interpretiramo kao očitanje njegove DTFT i to je razlog da su gotovo sva svojstva koja vrijede za DTFT primjenjiva i za DFT.

DFT pretvara proizvoljnu vremensku sekvencu od N brojeva u N brojeva frekvencijskih koeficijenata. Ti koeficijenti predstavljaju zastupljenost vremenske domene u frekvencijskoj domeni. Kao rezultat, frekvencijski koeficijenti predstavljaju amplitudu i fazu. Signal je u potpunosti okarakteriziran kao N broj uzoraka u vremenskoj domeni koji odgovaraju N brojeva frekvencijskih koeficijenata. Prikaz signala od DFT koeficijenata omogućuje minimalnu najmanju kvadratnu pogrešku.

Imamo nekoliko posebnih algoritama kod računanja DFT-a. Ovdje ćemo obraditi RE2FFT, pomoću kojeg možemo računati dva realna niza uz pomoć jedne kompleksne DFT, i REDFFT, transformacija realnog niza $x(n)$ duljine 2N pomoću DFT duljine N.

1.4.1. RE2FFT

Od dva nezavisna realna i konačna niza $x(n)$ i $y(n)$ želimo naći DFT transformaciju $X(k)$ i $Y(k)$. Prvo trebamo od dva realna stvoriti novi kompleksni niz $z(n) = x(n) + jy(n)$ i prvi korak je

$$Z(k) = \sum_{n=0}^{N-1} [x(n) + jy(n)] e^{-j\frac{2\pi}{N}kn}.$$

Kako se kompleksna eksponencijalna funkcija može zapisati kao $e^{it} = \cos t + i \sin t$ razlažemo DFT na realni i imaginarni dio

$$Z(k) = \overbrace{\sum_{n=0}^{N-1} x(n) \cos\frac{2\pi}{N}kn}^{Re[X(k)]} + \overbrace{\sum_{n=0}^{N-1} y(n) \sin\frac{2\pi}{N}kn}^{-Im[Y(k)]} +$$

$$+j\left[\overbrace{\sum_{n=0}^{N-1} y(n) \cos\frac{2\pi}{N}kn}^{Re[Y(k)]} - \overbrace{\sum_{n=0}^{N-1} x(n) \sin\frac{2\pi}{N}kn}^{Im[X(k)]}\right],$$

od prethodne relacije dobivamo $Z(k)$ u obliku imaginarnog i realnog dijela

$$Re[Z(k)] = Re[X(k)] - Im[Y(k)]$$

$$Im[Z(k)] = Re[Y(k)] + Im[X(k)]$$

$$Re[Z(N-k)] = Re[X(N-k)] - Im[Y(N-k)]$$

$$Im[Z(N-k)] = Re[Y(N-k)] + Im[X(N-k)].$$

Kombinacijom prethodnih jednadžbi određujemo imaginarni i realni dio $X(k)$ i $Y(k)$

$$Re[X(k)] = \frac{1}{2}\{Re[Z(k)] + Re[Z(N-k)]\}$$

$$Im[X(k)] = \frac{1}{2}\{Im[Z(k)] - Im[Z(N-k)]\}$$

$$Re[Y(k)] = \frac{1}{2}\{Im[Z(k)] + Im[Z(N-k)]\}$$

$$Im[Y(k)] = \frac{1}{2}\{Re[Z(N-k)] - Re[Z(k)]\}.$$

I tako smo pomoću DFT duljine N istovremeno transformirali dva realna niza duljine N.

1.4.2. REDFFT

Od realnog niza duljine 2N želimo izračunati transformaciju DFT-a duljine N, to je moguće ako niz $v(n)$ duljine 2N podijelimo na dva niza definirana kao $x(n) = v(2n)$ i $y(n) = v(2n+1)$ za $0 \leq n \leq N-1$.

Dok je DFT niza $v(n)$, za $0 \leq k \leq 2N-1$, oblika

$$V(k) = DFT_{2N}[v(k)] = \sum_{m=0}^{2N-1} v(m) W_{2N}^{km} =$$

$$= \sum_{m=0,2,4,..} v(m)\, W_{2N}^{km} + \sum_{m=1,3,5,...} v(m)\, W_{2N}^{km}.$$

Uvrštavanjem $x(n) = v(2n)$ i $y(n) = v(2n+1)$ dobivamo

$$V(k) = \sum_{n=0}^{N-1} v(2n) W_{2N}^{2kn} + \sum_{n=0}^{N-1} v(2n+1) W_{2N}^{(2n+1)k}$$

$$V(k) = \underbrace{\sum_{n=0}^{N-1} x(n) W_N^{kn}}_{X(k)} + W_{2N}^{k} \underbrace{\sum_{n=0}^{N-1} y(n) W_N^{nk}}_{Y(k)}.$$

Posebno računamo dva niza $X(k)$ i $Y(k)$ i DFT transformaciju početnog niza $v(n)$ izračunamo iz

$$V(k) = X(\langle k \rangle_N) + W_{2N}^{k} Y(\langle k \rangle_N), \qquad 0 \leq k \leq 2N-1.$$

1.4.3. Dvodimenzionalni (2D) DFT

U teoriji dvodimenzionalni (2D) signal je, najvećim dijelom, jednostavno proširenje jednodimenzionalnog (1D) signala. Tako da se diskretna Fourierova transformacija može primjenjivati i na signalima kao što su slike, odnosno prostorno vremenski (2-D) signala.

2D DFT kao slika $N \times N$

$$X(k_1, k_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) W_N^{n_1 k_1} W_N^{n_2 k_2}, \quad k_1, k_2 = 0,1, \dots, N-1,$$

2D IDFT definiran kao

$$x(n_1, n_2) = \frac{1}{N^2} \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} X(k_1, k_2) W_N^{-n_1 k_1} W_N^{-n_2 k_2}, \quad n_1, n_2 = 0,1, \dots, N-1.$$

1.5. Brza Fourierova transformacija (FFT)

Brza Fourierova transformacija je učinkovita implementacija diskretne Fourierove transformacije. Od svih diskretnih transformacija, DFT se naširoko koristi u digitalnoj obradi signala. Razvoj DFT-a originalno je djelo Cooleya i Tukeya, no njihov algoritam je poboljšan od strane raznih drugih znanstvenika. Mnoge tvrtke nude softversku implementaciju FFT i srodnih aplikacija, kao što su konvolucija, korelacija, filtriranje, spektralna analiza i slično na različitim platformama. DSP čipovi opće namjene mogu biti programirani za izvedbu FFT-a i drugih diskretnih transformacija.

Izravno računanje DFT-a u N točaka zahtjeva gotovo $O(N^2)$ kompleksnih aritmetičkih operacija (N^2 kompleksnih množenja i $N(N-1)$ kompleksnih zbrajanja). Međutim, ta složenost može biti znatno smanjena razvojem učinkovitog algoritma. Ključ smanjenja računske složenosti je da se u $N \times N$ DFT matrici od N^2 elemenata razlikuju samo N elemenata. Takvi algoritmi se nazivaju FFT algoritmi. Nekoliko tehnika je razvijeno za FFT, a ovdje su nekoliko tih algoritama: Cooley-Tukey, decimacija po vremenu, decimacija po frekvenciji, radix-2, radix-3, radix-4 , mixed-radix, split-radix, vector-radix, vector-split radix, itd.

1.5.1. Prednosti

Općenito, brzi algoritmi smanjuju računsku složenost N točaka DFT-a na $N \log_2 N$ složenost računskih operacija. Dodatne prednosti su smanjenje zahtjeva za pohranu i računske pogreške zbog konačne širine aritmetike. Brzi algoritmi pridonijeli su mogućnosti DFT implementacije na DSP čipovima.

1.5.2. Metoda podijeli pa vladaj (Cooley-Tukey algoritam)

Cooley-Tukey FFT algoritam (Jeren, 2012) se računa tako da se DFT u N točaka podijeli u manje DFT-a. Najprije N točaka DFT-a faktoriziramo kao produkt dva cijela broja, $N = ML$ te zatim računamo DFT od M točaka i DFT od L točaka. Od

jednodimenzionalnog niza $x(n), 0 \leq n \leq N-1$, dobijemo dvodimenzionalno polje indeksirano s l i m, $0 \leq l \leq L-1,\ 0 \leq m \leq M-1$, gdje l predstavlja indeks redaka, a m indeks stupaca, to je prikazano na tablici 1.1.

Tablica 1.1 Indeksiranje redaka i stupaca kod Cooley-Tukey FFT algoritma (Jeren, 2012)

$l\backslash m$	0	1	...	$M-1$
0	$x(0,0)$	$x(0,1)$	...	$x(0,M-1)$
1	$x(1,0)$	$x(1,1)$	...	$x(1,M-1)$
...	...	...	...	...
$L-1$	$x(L-1,0)$	$x(L-1,1)$	...	$x(L-1,M-1)$

Jednodimenzionalni niz $x(n)$ možemo razložiti u dvodimenzionalno polje na više načina, tako imamo razlaganje po redcima i stupcima. Razlaganje po redcima računa se po formuli $n = Ml + m$, gdje prvi redak sadrži prvih M elemenata $x(n)$, drugi redak sadrži sljedećih M elemenata, itd. Razlaganje po stupcima računa se po formuli $n = l + mL$, gdje prvi stupac sadrži prvih L elemenata $x(n)$, drugi stupac sadrži sljedećih L elemenata, itd.

1.5.3. Radix-2 FFT

Matematika je pokazala da se DFT u N točaka može još efikasnije izračunati ako se N fakturira kao $N = r_1 \cdot r_2 \cdot r_3 \cdot \cdots \cdot r_j$ gdje su r_j prim brojevi. Posebno je zanimljivo kada su $r_j = r_1 = r_2 = r$ pa je tada $N = r^j$ i DFT poprima pravilnu strukturu, r se tada naziva baza (*radix*) FFT algoritma. Radix-2 FFT algoritam je FFT algoritam po bazi 2 koji razlaže N točaka u $N = 2^j$ i koristi postupak podijeli pa vladaj. Radix-2 algoritam može se računati kao algoritam decimacije u vremenu (DIT FFT) i kao algoritam decimacije u frekvenciji (DIF FFT).

1.5.3.1 Decimacija u vremenu (decimation-in-time algorithm, DIT FFT)

DFT u N točaka podijelimo tako da je $M = N/2$ i $L = 2$. Jednodimenzionalni niz $x(n)$ razlažemo u dva retka, tako da se u prvom retku $f_1(n)$ nalaze samo parni uzorci niza $x(n)$, a u drugom $f_2(n)$ samo neparni uzorci. Tada DFT u N točaka ima oblik

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\dots,N-1$$

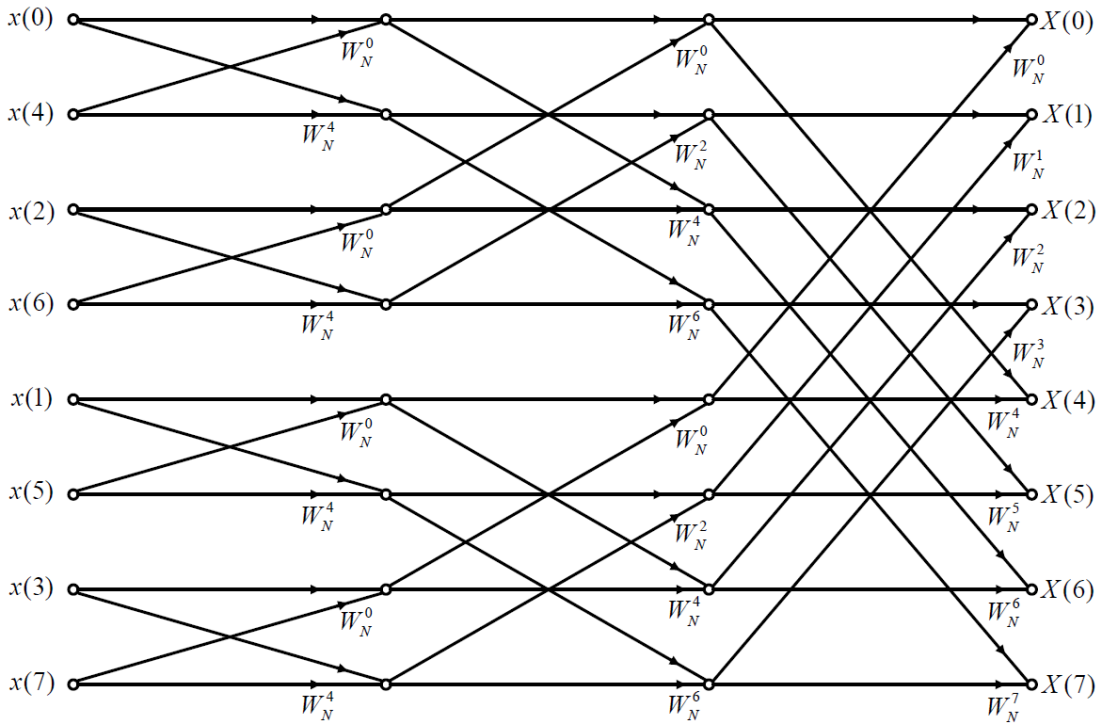
$$= \sum_{n\ je\ paran} x(n) W_N^{kn} + \sum_{n\ je\ neparan} x(n) W_N^{kn} =$$

$$= \sum_{m=0}^{N/2-1} x(2m) W_N^{2km} + \sum_{m=0}^{N/2-1} x(2m+1) W_N^{(2m+1)k} =$$

$$= \underbrace{\sum_{m=0}^{N/2-1} f_1(n) W_{\frac{N}{2}}^{mk}}_{F_1 = DFT_{N/2}[f_1(n)]} + \underbrace{\sum_{m=0}^{N/2-1} f_2(n) W_{\frac{N}{2}}^{mk}}_{F_2 = DFT_{N/2}[f_2(n)]}$$

$$X(k) = F_1(k) + W_N^k F_2(k), \qquad k = 0,1,\dots,N-1.$$

$F_1(k)$ i $F_2(k)$ su izračuni DFT-a u $N/2$ točaka i periodični su s periodom $N/2$. Slika 1.2 prikazuje graf tijeka signala DFT-a u N točaka pri čemu je $N = 8$ i koristi se decimacija u vremenu.



Slika 1.2 Prikaz graf tijeka signala DFT-a u N=8 točaka, decimacija u vremenu (Jeren, 2012)

Promatranjem grafa tijeka signala uočavamo pravilnu strukturu koju nazivamo leptir (*butterfly*). Kao što vidimo imamo $\log_2 N = 3$ stupnjeva, po svakom stupnju provodi se N kompleksnih množenja pa je ukupni broj kompleksnih množenja $N\log_2 N = 24$.

Prisjetimo se da složenost DFT-a N^2 i imamo 64 kompleksnih množenja pa je faktor smanjenja broja množenja 2.66 što se s povećanjem broja *N* povećava. Važno je primijetiti i poseban poredak uzoraka niza $x(n)$ koji se postiže algoritmom kojeg nazivamo *bit reversed algoritmom*.

1.5.3.2 Decimacija u frekvenciji (decimation-in-frequency algorithm, DIF FFT)

DFT u N točaka podijelimo tako da je $L = N/2$ i $M = 2$. Jednodimenzionalni niz $x(n)$ razlažemo u dva stupca, tako da se u prvom stupcu nalaze samo prvih $L = N/2$ elemenata niza $x(n)$, a u drugom nalazi sljedeći $L = N/2$ elemenata niza. Tada DFT u N točaka ima oblik

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0,1,\dots,N-1$$

$$= \sum_{prvih\ n} x(n) W_N^{kn} + \sum_{drugih\ n} x(n) W_N^{kn} =$$

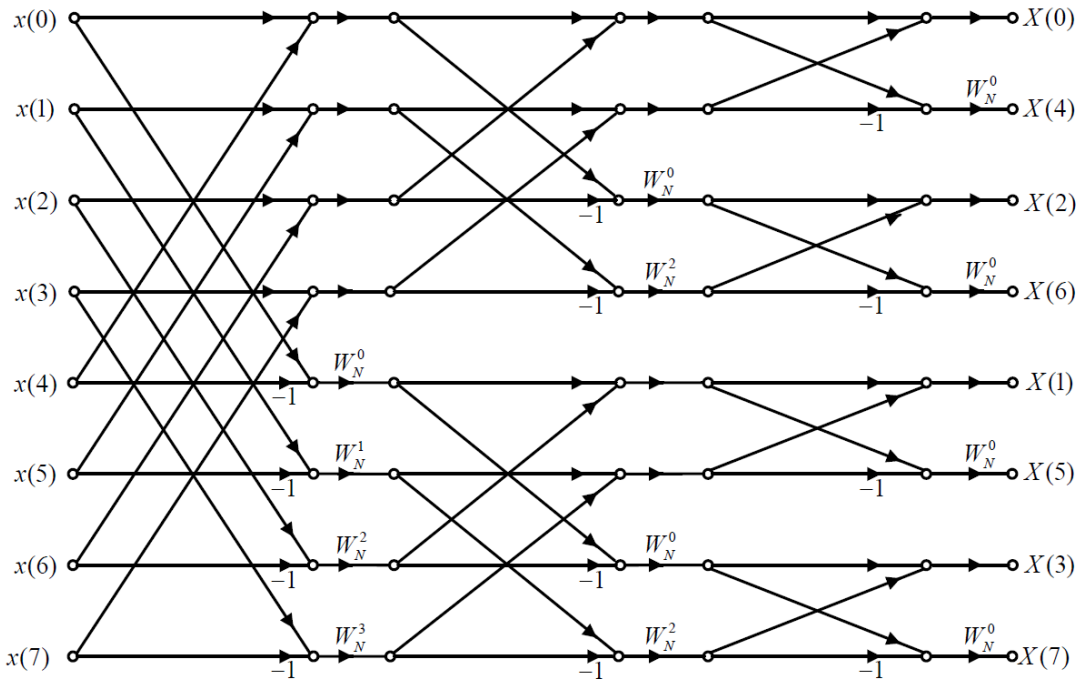
$$= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=N/2}^{N-1} x(n) W_N^{nk} =$$

$$= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + \sum_{n=0}^{N/2-1} x\left(n+\frac{N}{2}\right) W_N^{(n+\frac{N}{2})k} =$$

$$= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + W_N^{\frac{N}{2}k} \sum_{n=0}^{N/2-1} x\left(n+\frac{N}{2}\right) W_N^{nk} =$$

$$= \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + (-1)^k \sum_{n=0}^{N/2-1} x\left(n+\frac{N}{2}\right) W_N^{nk}.$$

Postupak je sličan kao i kod decimacije u vremenu, na slici je prikaz grafa tijeka signala DFT-a u N=8 točaka na kojem se koristi decimacija u frekvenciji.



Slika 1.3 Prikaz DFT-a u N=8 točaka, decimacija u frekvenciji (Jeren, 2012)

Važno je primijetiti da je niz $x(n)$ u prirodnom poretku uzoraka, a izračunati niz uzoraka $X(k)$ je u *bit reversed* poretku. Kao i kod decimacije u vremenu tako i kod decimacije u frekvenciji možemo uočiti pravilnu strukturu grafa tijeka signala koju zovemo leptir.

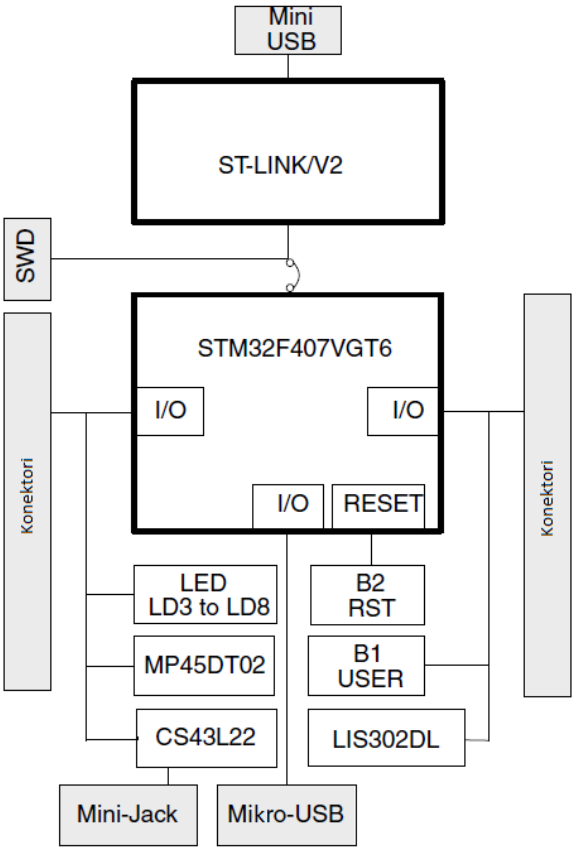
2. STM32F4 Discovery

STM32F4 Discovery pomaže vam da otkrijete STM32F4 značajke visokih performansi i razvoj različitih aplikacija. Temelji se na STM32F407VGT6 čipu i uključuje ST-LINK/V2 ugradbeno sučelje za *debugiranje*, ST MEMS digitalni akcelerometar, ST MEMS digitalni mikrofon, audio DAC s integriranom klasom D, LED-ice, tipkala i USB OTG *micro*-AB priključak.

2.1. Značajke

- STM32F407VGT6 mikrokontroler s 1 MB Flash memorije, 192 KB RAM u LQFP100 kućištu
- Napajanje: preko USB sabirnice ili vanjsko napajanje od 3 V ili 5 V
- LIS302DL, ST MEMS senzor pokreta, 3-osni digitalni izlaz akcelerometra
- MP45DT02, ST MEMS audio senzor, digitalni mikrofon
- CS43L22, audio DAC s integriranom klasom D
- LED:
  - USB komunikacija, LD1 (crvena/zelena)
  - Napajanje, LD2 (crvena)
  - 4 opće namjene, LD3 (narančasta), LD4 (zelena), LD5 (crvena) i LD6 (plava)
  - 2 USB OTG, LD7 (zelena) i LD8 (crvena)
- 2 tipke (korisnička i reset)
- USB OTG s micro-AB priključak

Na slici 2.1 je prikazan blok dijagram sklopovlja



Slika 2.1 Blok dijagram sklopovlja (STMicroelectronics, 2013)

2.2. Opis

STM32F407xx familija mikrokontrolera je bazirana na ARM Cortex™-M4 32-bitnoj RISC jezgri visokih performansi koja radi na frekvencijama i do 168 MHz. Cortex-M4 jezgra ima floating-point jedinicu (FPU) jednostruke preciznosti koja podržava sve ARM instrukcije i tipove podataka. Ima implementirani cijeli set DSP instrukcija i zaštitu memorije (Memory protection unit-MPU) koja povećava sigurnost aplikacija.

Familija STM32F407xx uključuje ugrađenu memoriju velike brzine (Flash memorija do 1 MB i 192 KB SRAM-a), bogati asortiman ulazno/izlaznih perifernih priključaka spojenih na dvije APB sabirnice, tri AHB sabirnice i 32-bitnu multi-AHB sabirnicu.

Na raspolaganju su tri 12-bitna ADC-a, dva DAC-a, *low-power* RTC, dvanaest 16-bitnih brojila opće namjene uključujući i dva PWM brojila za kontrolu motora, dva 32-bitna brojila opće namjene, generator slučajnih brojila (RNG). Tu su također standardna i

napredna komunikacijska sučelja kao što su I^2C, SPI, I^2S, USART, UART, USB OTG, CAN, SDIO/MMC sučelje, te sučelje za ethernet i kameru.

STM32F407xx familija radi na temperaturnom području od -40 do +105 °C ako je napajanje između 1.8 i 3.6 V. Napajanje može biti i 1.7 V samo ako uređaj radi na temperaturnom području od 0 do 70 °C.

Ove značajke čine STM32F407xx familiju mikrokontrolera pogodnu za široki raspon aplikacija:

- Kontrola vrtnje motora
- Medicinska oprema
- U industriji: PLC, pretvarači, prekidači
- Printeri i skeneri
- Alarmni sustav, video interfon
- Kućni audio uređaj

2.3. STM32F407VGT6 mikrokontroler

Mikrokontroler STM32F407VGT6 ima 32-bitnu ARM Cortex-M4 jezgru. ARM Cortex arhitektura dijeli se na tri podvrste: A, R i M. A profil (*Application profile*) ARM Cortex arhitekture zamišljena je kao jezgra na kojoj će raditi aplikacije visokih performansi (pametni telefoni, tableti, itd.). R profil (*Real-Time profile*) arhitekture zamišljen je za ugradbene aplikacije visokih performansi s posebnim zahtjevima za rad u stvarnom vremenu, dok je M profil (*Microcontroller profile*) arhitekture osmišljen za mikrokontrolerske aplikacije kod kojih su bitne kako performanse, tako i cijena, potrošnja, latencija prekida, jednostavnost itd.

ARM Cortex-M4F (s FPU) procesor je najnovija generacija ARM procesora u ugradbenim računalnim sustavima. Razvijena je da omogućuje platformu niske cijene koja zadovoljava MCU implementaciju uz smanjeni broj priključaka i nisku potrošnju energije, a u drugom pogledu da ima dovoljno brze računske performanse i napredni odgovor na prekide.

Procesor podržava DSP instrukcije koje omogućuju efikasnu obradu signala i izvođenje kompleksnih algoritama.

ART akcelerator je memorijski akcelerator koji je optimiziran za STM32 industrijski standard za ARM Cortex-M4 procesore. On uravnotežuje inherentne prednosti u

performansama procesora ARM Cortex-M4 nad Flash memorijom, koja obično zahtjeva da je procesor čeka na višim frekvencijama.

Zaštita memorije (MPU) se koristi za upravljanje CPU pristupa memoriji kako ne bi neki zadatak slučajno zauzeo memoriju ili resurse koje koristi aktivni zadatak. Ta memorija je organizirana u osam zaštićenih područja koje mogu biti podijeljene na osam potpodručja. Veličina zaštićenog područja je između 32 B i cijele 1 GB adresne memorije.

MPU je posebno korisna za aplikacije koje imaju kritične sekcije koje moraju biti zaštićene od ostalih zadataka koji mogu pristupati istim podacima. To se često dešava kod RTOS-a (*real-time operating system*).

STM32F40x uređaj ima ugrađenu Flash memoriju od 1 MB raspoloživu za pohranu programa i podataka. RAM memorija je organizirana u obliku SRAM od 192 kB uključujući i 64 kB CCM (*core coupled memory)* te može pristupati (čitati/pisati) taktu CPU-a bez čekanja. Ima 4 kB rezerve kojemu može pristupati samo CPU. Njegov sadržaj je zaštićen od mogućeg neželjenog pisanja u memoriju.

2.4. Standardno sinkrono/asinkrono serijsko sučelje (USART)

Ovaj mikrokontroler ima četiri standardna sinkrona/asinkrona serijska sučelja (USART1, USART2, USART3 i USART6) i dva standardna asinkrona serijska sučelja (UART4 i UART5). Omogućava fleksibilnu dvosmjernu (*full-duplex*) razmjenu podataka između mikrokontrolera i vanjskih uređaja. USART nudi veoma široki raspon brzine prijenosa podataka.

Podržava sinkronu jednosmjernu komunikaciju i poludvosmjernu komunikaciju (*half-duplex*), te lokalnu povezanu mrežu (*local interconnection network*-LIN), *Smartcard* protokol i infracrveni prijenos podataka (IrDA) i modem komunikaciju (CTS/RTS).

Serijska sučelja USART1 i USART6 mogu postići brzinu komunikacije od 10.5 Mbit/s dok druga raspoloživa sučelja mogu postići brzinu komunikacije do 5.25 Mbit/s.

2.4.1. Glavne značajke

- Dvosmjerni prijenos (*full-duplex*), asinkrona komunikacija
- NRZ (*Non return to zero*) standard
- Proizvoljna širina podatkovne riječi (8 ili 9 bita)

- Stop bit – podržava 1 ili 2 stop bita
- Odašiljački takt za sinkroni prijenos
- Frakcijski generator brzine prijenosa
    - Zajednička brzina prijenosa odašiljača i prijemnika
- Sposobnost LIN *mastera* za sinkronu pauzu i sposobnost LIN *slavea* da detektira pauzu
- IrDA SIR enkoder i dekoder
- Mogućnost *Smartcard* emulacije
- Jednožična poludvosmjerna komunikacija (*single-wire half-duplex*)
- Podesivi spremnik za komunikaciju koji koristi DMA (*direct memory access*)
    - Primanje sadržaja primljenih/poslanih bajtova u rezerviranom SRAM-u koji koristi centralizirani DMA
- Zastavice za detekciju prijenosa:
    - Prijemni spremnik je pun
    - Odašiljački spremnik je prazan
    - Zastavica za kraj prijenosa
- Kontrola pariteta
    - Paritet odašiljačke poruke
    - Provjera pariteta dobivene poruke
- Zastavice za otkrivanje pogrešaka
    - Detekcija šuma
    - Pogreška pariteta
    - Pogreška broja okvira
- Prekidi sa zastavicama
    - CTS promjene
    - Prazni podatkovni odašiljači registar
    - Prijenos je dovršen
    - Puni podatkovni prijemni registar
    - Pogreška šuma
    - Pogreška pariteta
- Multiprocesorska komunikacija
- Buđenje iz *mute* način rada procesora

2.4.2. Funkcijski opis USART-a

Sučelje je vanjski spojeno s nekim drugim uređajem pomoću tri priključka. Bilo koji USART za dvosmjernu komunikaciju zahtjeva minimalno dva priključka: prijemni priključak (RX) i odašiljački priključak (TX).

Prijemni priključak (RX) je serijski ulaz podataka. Tehnologijom podotipkavanja možemo dobiti korisne podatke iz signal koji je mješavina podataka i šuma.

Odašiljački priključak (TX) može biti onemogućen kada se ništa ne odašilje i tada se može koristiti kao obični ulazno/izlazni priključak. Kada je omogućeni prijenos podataka, ali se ništa ne prenosi TX-u je pridružena visoka razina. U jednožičnom i *Smartcard* načinu rada TX se koristi kao prijemni i odašiljački priključak.

Kroz ta dva priključka podaci se serijski odašilju i primaju u normalnom USART načinu rada. Okviri se sastoje od:

- Start bita
- Podatkovne riječi (8 ili 9 bita), LSB se šalje prvi
- 0.5, 1, 1.5 ili 2 Stop bita
- Sučelje koristi frakcijski generator brzine prijenosa – 12 bitna mantisa i 4 bitna frakcija
- Statusni registar
- Podatkovni registar
- Registar brzine prijenosa

Sljedeći priključak za sinkroni način rada sučelja je SCLK, odnosno odašiljački izlazni takt. Ovaj priključak je izlazni odašiljački podatkovni takt za sinkroni prijenos koji odgovara SPI *master* načinu rada (nema takta na start i stop bit, programska mogućnost za slanje takta na zadnji bit podatka). Paralelno se može primiti podatak na sinkronizirani RX priključak. To se može koristiti za periferiju koja koristi posmačne registre (npr. LCD pokaznik).

2.5. Brojila opće namjene

Brojila opće namjene mogu biti 16-bitna ili 32-bitna, pogonjena programiranim preskalarom. Mogu biti korištena u različite svrhe, uključujući mjerenje dužine pulsa

ulaznog signala (ulazno hvatanje) ili generiranje izlaznih valnih oblika (PWM, usporedba izlaza).

Širina pulsa i period valnih oblika moguće je modulirati kroz nekoliko mikrosekundi do nekoliko milisekundi koristeći preskalar brojila i preskalar RCC takta. Brojila su u potpunosti neovisna te ne dijele zajedničke resurse.

2.5.1. Glavne značajke

Brojila opće namjene uključuju:

- 16-bitna (TIM3 i TIM4) ili 32-bitna (TIM2 i TIM5) brojila prema gore, prema dolje, prema gore/dolje
- 16-bitni programabilni preskalar za dijeljenje frekvencije takta brojila, može biti bilo koji faktor između 1 i 65536
- 4 nezavisna kanala:
    - Hvatanje ulaza
    - Usporedba izlaza
    - PWM generator
    - Izlazni jednoimpulsni način rada
- Sinkronizirani krug za kontrolu rada brojila s vanjskim signalom i za povezivanje nekoliko brojila
- Mogućnost okidanja na događaj (start, stop, inicijalizacija ili brojanje na unutarnji/vanjski okidač)
- Podržava inkrementalni enkoder i *Hallov* senzor koji služi za pozicioniranje

2.5.2. Funkcijski opis brojila

2.5.2.1 Jedinica vremenske baze

Glavni blok programabilnog brojila je 16-bitni/32-bitni brojač sa samopunjivim (*auto-reload*) registrom. Brojač može brojati prema gore i takt može biti dijeljen s preskalarom.

U brojač, samopunjivi registar i preskalar se može pisati i iz njih čitati pomoću programa.

Pomoću programa možemo čitati i pisati u brojač, samopunjivi registar i preskalar.

Jedinica vremenske baze uključuje:

- Registar brojača
- Registar preskalara
- Samopunjivi registar

2.5.2.2 Načini rada brojila

2.5.2.2.1 Brojanje prema gore

U ovom načinu rada, brojanje prema gore, brojač broji od 0 do samopunjive vrijednosti koja je zapisana u samopunjivom registru, tada se resetira brojač na 0 i generira se događaj koji označava preljev brojača.

2.5.2.2.2 Brojanje prema dolje

U ovom načinu rada, brojanje prema dolje, brojač broji od samopunjive vrijednosti koja je zapisana u samopunjivom registru do 0, tada se resetira brojač na samopunjivu vrijednost i generira se događaj koji označava da je brojač na 0.

2.5.2.2.3 Brojanje prema gore/dolje (centralno poravnat način rada)

Centralno poravnat način rada broji od 0 do samopunjive vrijednosti, koja je zapisana u samopunjivom registru, -1, generira se događaj koji označava preljev brojača, tada se od samopunjive vrijednosti broji prema 1 i generira se događaj koji označava da je brojač na 0. Nakon toga se brojač resetira na početno stanje 0.

2.6. Analogno-digitalni pretvornik (ADC)

Analogno digitalni pretvornik sa sukcesivnom aproksimacijom ima 12-bitnu rezoluciju. Ima 19 multipleksiranih kanala koji omogućuju mjerenje signala sa 16 različitih vanjskih izvora, dva interna izvora i napon U_{BAT}. AD pretvorba pojedinog kanala može biti u pojedinačnom, kontinuiranom, skenirajućem ili isprekidanom načinu rada. Rezultat pretvorbe ADC-a je spremljen, lijevo ili desno poravnat, u 16-bitni podatkovni registar.

Analogni *watchdog* omogućuje da se detektira da li je ulazni napon izvan granica koje je definirao korisnik.

2.6.1. Glavne značajke ADC-a

- Rezolucija: 12-bita, 10-bita, 8-bita ili 6-bita
- Generira prekid: na kraju pretvorbe, na kraju injektirane pretvorbe te u slučaju aktiviranja analognog *watchdoga* ili prekoračenja granica
- Pojedinačni i kontinuirani način pretvorbe
- Skenirajući način pretvorbe za automatsku pretvorbu kanala
- Poravnavanje podataka
- Programabilno vrijeme uzrokovanje signala
- Isprekidani način pretvorbe
- Napajanje ADC-a: 2.4 V do 3.6 V za brži način rada i do 1.8 V za sporiji način rada
- Ulazni raspon ADC-a: $0 \leq U_{IN} \leq U_{REF+}$

2.6.2. Funkcijski opis ADC-a

2.6.2.1 ADC on-off

Uključivanje ADC-a se postiže postavljanjem bita ADON u ADC_CR2 registru. Pretvorba započinje kada se ili SWSTART ili JSWSTART bit postavi na jedinicu.

Brisanjem bita ADON isključujemo ADC i tada zaustavljamo pretvorbu. U takvom načinu rada ADC gotovo ne troši struju (samo nekoliko μA).

2.6.2.2 ADC takt

- Takt za analogni krug: ADCCLK, zajednički za sve ADC-ove,
  Analogni takt je generiran pomoću APB2 takta koji je podijeljen programabilnim preskalarom koji omogućuje ADC-u da radi na f_{PCLK2}/2, /4, /6 ili /8.
- Takt za digitalno sučelje koji je jednak APB2 taktu i može biti omogućen/onemogućen posebno za svaki ADC

2.6.2.3 Odabir kanala ADC-a

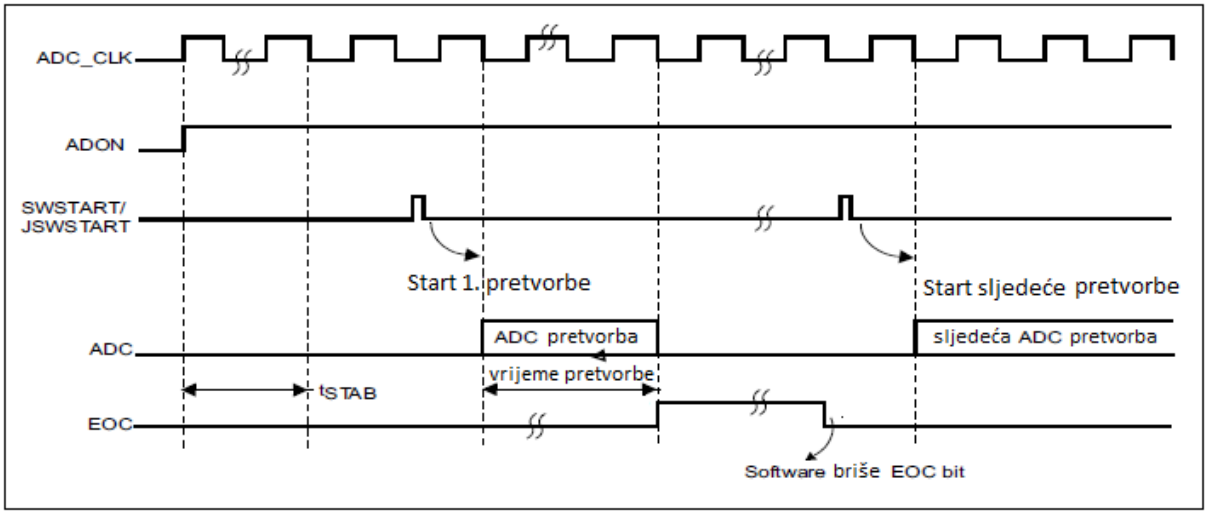
Na odabiru su 16 multipleksiranih kanala koje je moguće organizirati u dvije grupe: regularnu i injektiranu. Grupa u kojoj se nalaze regularni kanali sastoji se od 16 pretvorba. Regularni kanali i njihov redoslijed pretvorbe treba biti upisan u ADC_SQRx registrima.

Konačni broj pretvorbi mora biti zapisan u 4 najmanje značajna bita [3:0] u ADC_SQR1 registru.

Grupa u kojoj se nalazi injektirani kanali sastoji se od 4 pretvorbe. Ti injektirani kanali i njihov redoslijed pretvorbe treba biti upisan u ADC_JSQR registar. Konačni broj pretvorbi mora biti zapisan u 2 najmanje značajna bita [1:0] u ADC_JSQR registru.

2.6.2.4 Dijagram vremena

ADC treba neko vrijeme t_{STAB} da bi se stabilizirao prije nego započne s pretvorbom. Nakon početka ADC pretvorbe i nakon 15 ciklusa takta (vrijeme potrebno za najbržu pretvorbu), EOC zastavica se postavlja i 16-bitni ADC podatkovni registar poprima vrijednost pretvorbe. Na slici 2.2 je prikazan dijagram vremena.



Slika 2.2 Dijagram vremena (STMicroelectronics, 2013)

2.6.2.5 Pojedinačni način pretvorbe (single mode)

U pojedinačnom načinu pretvorbe, ADC izvede samo jednu pretvorbu. Ova pretvorba započinje postavljanjem bita SWSTART u ADC_CR2 registru za regularni kanal, postavljanjem bita JSWSTART za injektirani kanal i vanjskim okidačem za regularni ili injektirani kanal.

Kada je pretvorba završila :

- Pretvoreni podatak je spremljen u 16-bitni ADC_DR registar
- Postavlja se EOC (kraj pretvorbe) zastavica
- Servira se prekid ako je postavljen EOCIE bit

2.6.2.6 Kontinuirani način pretvorbe (continuous conversion mode)

U kontinuiranom načinu pretvorbe, ADC započinje novu pretvorbu čim završi s prethodnom. Pretvorba započinje kada se postavi CONT bit ili pomoću vanjskog okidača ili postavljanjem SWSTRT bita ADC_CR2 registra.

Nakon svake pretvorbe:

- Podaci se spremaju u 16-bitni ADC_DR registar
- Postavlja se EOC (kraj pretvorbe) zastavica
- Servira se prekid ako je postavljen EOCIE bit

2.6.2.7 Skenirajući način pretvorbe (scan mode)

Ovaj način pretvorbe se koristi kada se skenira grupa analognih kanala. Skenirajući način pretvorbe selektira se postavljanjem SCAN bita u ADC_CR1 registru. Kada je taj bit postavljen, ADC skenira sve kanale koji su selektirani u ADC_SQRx registrima. Pojedinačna pretvorba se izvodi za svaki kanal u grupi. Nakon kraja svake pretvorbe, sljedeći kanal automatski započinje pretvorbu. Ako je CONT bit selektiran, kada zadnji kanal obavi pretvorbu, tada se automatski kreće od prvog kanala i tako se sve vrti u krug.

Ako je DMA bit selektiran, tada DMA pretvoreni podatak odmah transportira iz kanala u SRAM.

2.6.2.8 Isprekidani način pretvorbe (discontinuous mode)

Ovaj način pretvorbe omogućuje se postavljanjem DISCEN bita u ADC_CR1 registru. Može biti upotrebljen za pretvorbu kratkih sekvenci on N pretvorbi ($N \leq 8$) kanala koji su selektirani u ADC_SQRx registrima. Vrijednost broja N se upisuje u DISCNUM[2:0] bitove ADC_CR1 registra.

Kada vanjski okidač okine, počinje sljedećih N pretvorbi selektiranih u ADC_SQRx registrima sve dok svi kanali nisu završili pretvorbu. Duljina sekvenci je definirana u četiri najmanje značajna bita [3:0] u ADC_SQR1 registru.

3. MATLAB

MATLAB (***mat**rix **lab**oratory*) je programski jezik visoke razine i interaktivna okolina za numeričko računanje, vizualizaciju i programiranje. Koristeći MATLAB moguće je analizirati podatke, razvijati algoritme i kreirati modele i aplikacije. Programski jezik, alati i izrađene matematičke funkcije omogućuju vam da istražite više različitih pristupa. Uz pomoć MATLAB-a do rješenja se dolazi brže nego s proračunskim tablicama ili tradicionalnim programskim jezicima poput C/C++ ili Jave.

Može se koristiti za veliki niz aplikacija, uključujući obradu signala i komunikacije, obrada slike i videa, sustav kontrole, testiranja i mjerenja, računanje financija itd. Više od milijun inženjera i znanstvenika u industriji i akademskoj zajednici koriste MATLAB, jezik tehničkog računarstva.

Prvu verziju MATLAB-a počeo je razvijati u kasnim 1970-tim Cleve Moler, predsjednik odjela za informatiku u Sveučilištu u Novom Meksiku (*University of New Mexcio*). On je osmislio kako da studentima omogući pristup LINPACK i EISPACK biblioteci bez da ih nauči kako radi Fortran. Ubrzo se taj pristup proširio i na druga sveučilišta te pronašao publiku u zajednici za primijenjenu matematiku. Jack Little, inženjer, bio je oduševljen Molerovim posjetom Sveučilištu Stanford (*Stanford University*) 1983. Prepoznao je komercijalni potencijal i pridružio se Moleru i Steveu Bangertu. Kasnije su ponovno napisali MATLAB u C-u i osnovali MathWorks 1984. i nastavili razvijati programski alat. Te ponovno napisane biblioteke poznate su kao JACKPAC. MATLAB je 2000. ponovo napisan i koristi noviji set biblioteka LAPACK za matričnu manipulaciju.

3.1. Ključne značajke MATLABA-a

- Programski jezik visoke razine za numeričko računanje, vizualizaciju i razvoj aplikacija
- Interaktivno okruženje za iterativno istraživanje, projektiranje i rješavanje problema
- Matematičke funkcije za linearnu algebru, statistiku, Fourierovu analizu, filtriranje, optimizaciju, numeričku integraciju i rješavanje diferencijalnih jednadžbi

- Alati za izradu grafova za vizualizaciju podataka
- Razvojni alati za poboljšanje kvalitete koda i povećanje performansi
- Alati za izradu aplikacija s prilagođenim grafičkim sučeljem
- Funkcije za integraciju MATLAB-a bazirane su na algoritmima vanjskih aplikacija i programskih jezika poput C-a, Jave, .NET-a i Microsoft Excel-a

3.1.1. Numeričko računanje

MATLAB pruža široki raspon numeričkih računanja za analizu podataka, razvoj algoritama i izradu modela. Uključuje matematičke funkcije koje podržavaju zajedničke inženjerske i znanstvene operacije. Temeljne matematičke funkcije koriste procesorsko optimiziranje biblioteka kako bi pružali brzo izvođenje vektorskih i matričnih proračuna.

3.1.2. Analiza i vizualizacija podataka

MATLAB nudi alate za analizu i vizualizaciju podataka što omogućuje uvid u dobivene podatke, a za to je potrebno kratko vrijeme u odnosu da se koriste proračunske tablice ili tradicionalni programski jezici. Također je moguće dokumentirati i podijeliti svoje rezultate kao grafove funkcija ili u obliku izvješća.

MATLAB omogućuje pristup podacima s različitih aplikacija, baza podataka i vanjskih uređaja. Podaci se mogu čitati u popularnim alatima poput Microsoft Excel-a, čitanje tekstualnih i binarnih datoteka, datoteka slike, zvuka i videa.

3.1.3. Programiranje i razvoj algoritma

MATLAB je programski jezik visoke razine i nudi razvojne alate s pomoću kojih je moguće brzo razviti i analizirali algoritam i aplikaciju.

S MATLAB programskim jezikom, moguće je napisati program ili razviti algoritam brže nego kod tradicionalnih programskih jezika zato što ne treba izvoditi administrativne procese niske razine kao što su deklaracija varijabli, specifikacija vrste podataka i alokacija memorije. Kod mnogo slučaja, vektorske i matrične operacije trebaju *for* petlju za izračun, kod MATLAB to je samo jedna linija koda, dok kod C-a i C++ trebamo koristi nekoliko linija koda. MATLAB nudi značajke tradicionalnih programskih jezika, poput rukovanja greškama, tijek kontrole i objektno orijentirano programiranje. Mogu se koristiti

fundamentalni tipovi podataka ili napredne strukture podataka, mogu se koristiti i prilagođeni tipovi podataka.

3.1.3.1 Razvojni alati

U MATLAB-u su uključeni razni alati za razvoj učinkovitih algoritama:

- Komandni prozor (*Command Window*) - omogućuje interaktivni unos podataka, izvršne naredbe, programe i prikaz rezultata
- MATLAB urednik (*Editor*) – pruža uređivačke i ispravljačke značajke
- Analizator koda (*Code Analyzer*) – automatski provjerava greške u kodu i predlaže izmjene kako bi se povećala učinkovitost performansi
- MATLAB *Profiler* – mjeri performanse MATLAB koda

3.1.3.2 Integracija s drugim programskim jezicima i aplikacijama

MATLAB aplikacije moguće je integrirati s aplikacijama napisanim u drugim programskim jezicima. Iz MATLAB-a je moguće izravno prebaciti kod u C, C++, Javu i .NET. Pomoću MATLAB *engine* biblioteke je moguće konvertirati kod iz C-a, C++ ili Fortrana u MATLAB kod.

3.1.3.3 Performanse

MATLAB koristi biblioteke, koje su optimizirane u odnosu na procesor, za brzo izvođenje matričnih i vektorskih računanja. Za računanje skalara opće namjene, MATLAB koristi *just-in-time* (JIT) kompilacijsku tehnologiju kako bi brzina izvođenja programa bila u usporedbi s tradicionalnim programskim jezicima. Kako bi iskoristio višejezgrene procesore, pruža mogućnost izvođenja paralelnih funkcija linearne algebre. Te funkcije se automatski izvode na više računskih dretvi u jednoj MATLAB operaciji, što će im omogućiti da rade brže na višejezgrenim procesorima.

3.1.4. Razvoj i razvijanje aplikacija

MATLAB alati i aplikacije za dodavanje pružaju niz mogućnosti za razvoj i razvijanje aplikacija. Moguće je dijeliti individualni algoritam i aplikaciju s drugim MATLAB korisnicima ili ih razvijati za one koji ne koriste MATLAB.

3.1.4.1 Projektiranje grafičkog sučelja

Koristeći GUIDE (*Graphical User Interface Development Environment*), moguće je dizajnirati i uređivati prilagođeno grafičko sučelje. Zajedničkim kontrolama moguće je dodavanje popisa, padajućih izbornika i upravljačkih tipaka. Grafičko sučelje može biti kreirano programski pomoću MATLAB funkcije.

3.1.4.2 Generiranje C koda

Moguće je koristiti MATLAB *Coder* kako bi se generirao samostalni C kod iz MATLAB koda. MATLAB *Coder* podržava veliki broj MATLAB programskih jezika, koje obično koriste razni inženjeri za razvoj algoritama kao komponente, koje su dio većeg sustava. Taj kod je moguće koristiti kao samostalni kod, a i kao integraciju za druge aplikacije ili ugradbene primjene.

3.2. STM32 ugradbeni alat za MATLAB i Simulink (STM32-MAT/TARGET)

3.2.1. Značajke

- Konfiguracijske Simulink aplikacije za STM32 mikrokontrolere
- Dostupni programi za programiranje ugradbenih sustava:
    - Atollic: TrueSTUDIO
    - IAR: EWARM
    - Keil: µVision4
- Automatsko generiranje C koda za STM32 mikrokontrolere
- Procesor u petlji (*Processor In the Loop*-PIL) s mogućnošću praćenja rezultata pomoću USART komunikacije
- Izvješće:
    - Generiranje koda
    - Izvršenja koda
- Simulink STM32 periferijska biblioteka

3.2.2. Opis

STM32 ugradbeni alat omogućuje brzi razvoj aplikacijskih modela izrađenih u MATLAB-u i Simulinku te njihovu implementaciju na STM32 mikrokontrolerima. Pruža mogućnost pokretanja Simulink aplikacije na STM32F4 koristeći konfiguraciju zvanu PIL i komunikaciju pomoću USART priključaka.

Generiranje C koda i implementacija na STM32F4 mikrokontroleru je u potpunosti automatizirana. Kod je generiran i prilagođen za jednog od tri moguća programska alata za programiranje ugradbenih sustava (Atollic, IAR i Keil). Izvješće generiranog koda se automatski generira.

STM32 ugradbeni alat osigurava Simulink biblioteku koja ima nekoliko periferijskih blokova za STM32F4 mikrokontrolere, tim blokovima moguće je promijeniti parametre i oni tada generiraju periferijsku inicijalizaciju u C kodu. Generirani kod moguće je integrirati u postojeću aplikaciju i na mikrokontroler.

Ove mogućnosti i funkcionalnosti lako je koristiti i lako su dostupne ako se STM32F4xx mape dodaju u MATLAB.

3.2.3. Simulink biblioteka Target Support Package – STM32F4xx Adapter

Simulink biblioteka *Target Support Package – STM32F4xx Adapter* sastoji se od pet osnovnih grupa blokova, a to su redom: ADC, GPIO, INTERRUPTS, TIMERS, USART. Svaka od postojećih grupa blokova sastoji se od inicijalizacijskog bloka i radnih blokova.

3.2.3.1 ADC

Sastoji se od 3 bloka:

- ADC zajednička inicijalizacija – koja služi za inicijalizaciju svih ADC kanala
- ADC inicijalizacija pojedinog kanala
- Čitanje podataka s pojedinog ADC kanala

3.2.3.2 GPIO

Sastoji se od 3 bloka:

- GPIO blok za inicijalizaciju pojedinog priključka

- GPIO blok za čitanje priključka
- GPIO blok za pisanje u priključak

3.2.3.3 INTERRUPTS

Grupa se sastoji od jednog konfiguracijskog bloka na kojem se odabire vrsta i prioritet prekida te na kojoj se periferiji izvodi prekid.

3.2.3.4 TIMERS

Grupa se također sastoji od jednog bloka na kojem se konfigurira brojilo mikrokontrolera.

3.2.3.5 USART

Sastoji se od 3 bloka:

- USART inicijalizacija
- USART blok za slanje podataka
- USART blok za primanje podataka

4. Implementacija diskretne Fourierove transformacije na mikrokontroleru STM32F4

U ovom radu bit će opisane dvije implementacije diskretne Fourierove transformacije na mikrokontroleru STM32F4. Prva implementacija je izvedena pomoću blokova u Simulinku koji je dio programskog alata MATLAB, dok je druga izvedena kao programska funkcija diskretne Fourierove transformacije u C/C++ programskom jeziku napisana u programu za razvoj mikrokontrolera Keil µVision4.

Kako bi rezultati diskretne Fourierove transformacije bili usporedivi, obje verzije baziraju se na korištenju cjelobrojne aritmetike (*fixed-point*). Cjelobrojna aritmetika odabrana je zbog toga što se očekuje da će mikrokontroler najbrže raditi upravo s *fixed-point* operacijama. U ovim implementacijama koristimo 16-bitne podatke tipa *fixed-point* (*short*), odnosno q15_t format koji je tip Q15.0 notacije.

4.1. Cjelobrojna aritmetika (fixed-point)

Cjelobrojni zapis je u stvari podatak tipa *integer* pomnožen s nekim faktorom za skaliranje. Moguće je zapisati brojeve s decimalnom točkom (na primjer 4.568, to je u stvari prikaz broja 4568 koji je skaliran s faktorom 0.001), dok to nije moguće kod korištenja *integera*. Riječ „*fixed*“ znači da je decimalna točka fiksirana i da se uvijek nalazi na istom mjestu.

Tablica 4.1 Usporedba različitih 8-bitnih tipova podataka

	Signed int.	Fixed-point (Q4.3)	Fixed-point (Q0.7)	Floating point
Min.	-128	-16	-1	-240
Maks.	127	15.5875	0.992188	240
Maks. – 1	126	15.75	0.984375	224
Preciznost	1	0.125	0.0078125	16

Za početak usporedit ćemo nekoliko različitih tipova zapisa brojeva. Kada pogledamo raspon različitih formata vidimo da je kod *signed integer* i *fixed-point* zapisa jednolika razdioba brojeva dok je kod *floating-point* zapisa nejednolika. U tablici 4.1 možemo očitati raspon brojeva, maksimalnu i minimalnu vrijednost zapisa te preciznost.

Imamo različite notacije cjelobrojnih zapisa, jedna od tih notacija je i Qx.y notacija. Ta notacija je zapisana u zapisu s dvojnim komplementom. Broj bitova je zbroj $x + y + 1$ (bit za predznak), x je broj bitova ispred decimalne točke (*integer*), a y je dio iza decimalne točke (frakcionalni dio, ujedno povezan i s faktorom skaliranja). Radi lakšeg razumjevnja napisano je nekoliko primjera Qx.y notacije u tablici 4.2.

Kako bi se lakše snalazili u tablici opisat ćemo Q4.3 notaciju. Sastoji se od 8 bitova koji su zapisani u dvojnom komplementu te zbog toga ima jedan bit rezerviran za predznak, on je najznačajniji bit (MSB) kod ovog zapisa i svih ostalih cjelobrojnih zapisa s dvojnim komplementom. Broj 4 označava 4 bita koja označuju cjelobrojni zapis i broj 3 označava 3 bita frakcionalnog dijela zapisa.

Tablica 4.2 Primjer Qx.y notacija

Q notacija	Signed int. (Q31.0)	Q4.3	Q0.7 Q7	Q0.15 Q15	Q16.15	Q0.31 Q31
Broj bitova	32	8	8	16	32	32
bit predznaka	1	1	1	1	1	1
Frakcionalni dio	0	3	7	15	15	31
Faktor skaliranja	$1/2^0$	$1/2^3$	$1/2^7$	$1/2^{15}$	$1/2^{15}$	$1/2^{31}$

Neke prednosti *fixed-point* aritmetike:

- *Fixed-point* aritmetika ne zahtijeva dodatno specifično sklopovlje
  - Koristi se uobičajene binarne operacije
- Jednaka preciznost na cijelom rasponu brojeva

Nedostaci *fixed-point* aritmetike:

- Netrivijalna implementacija
  - Razlika između *integer* i *fixed-point* aritmetike

    - Treba paziti na: skaliranje prije i poslije binarnih operacija, skaliranje može rezultirati preljevom ili gubitkom preciznosti
- Konačan raspon
    - 32-bitni *fixed-point* ima 2^{32} različitih vrijednosti
    - Raspon ovisi o faktoru skaliranja

Navest ćemo neke prednosti i nedostatke *floating-point* aritmetike radi bolje usporedbe ta dva tipa podataka.

Prednosti *floating-point* aritmetike:

- Raspon vrijednosti je puno veći od raspona *fixed-point* aritmetike
    - Ima jednak broj bitova kao i *fixed-point* aritmetika samo što su vrijednosti raspršene po puno većem rasponu
- Lakša je za implementaciju
    - *Floating-point* ima neke specijalne operacije i brojeve
    - Potrebno kraće vrijeme razvijanja u odnosu na *fixed-point*

Nedostaci *floating-point* aritmetike:

- *Floating-point* aritmetika možda zatreba dodatno specifično sklopovlje
    - Aritmetičko logička jedinica ne može direktno koristiti *floating-point* operacije
- Preciznost nije jednaka na cijelom rasponu
    - Bazirana je na nejednolikoj razdiobi vrijednosti
    - Moguća pojava pogreške kod zaokruživanja

4.2. Implementacija DFT-a u C/C++ programskom jeziku

Kao što je spomenuto na početku ovog poglavlja implementacija DFT je razvijena u programu za razvoj mikrokontrolera Keil µVision4. Zbog brzine izvođenja koristimo jedan od mnogobrojnih FFT algoritama. Kod ove implementacije odabrali smo koristiti Radix-2 DIT (decimacija u vremenu) algoritam. Njegove specifikacije naveli smo u prijašnjem poglavlju (vidi 1.5.3. Radix-FFT) ali da se prisjetimo glavnih obilježja. Radix-2 FFT algoritam je FFT algoritam koji koristi bazu 2 kao razlaganje broja *N* (broj točaka za izračun DFT-a). Pravilnu strukturu grafa tijeka signala nazivamo leptir (*butterfly*) te koristimo poseban *bit reversed* algoritam da promijenimo poredak ulaznih točaka.

4.2.1. Bit reversed algoritam

Kao *bit reversed* algoritam koristimo Yongov algoritam. 1991. godine Yong je izmislio algoritam koji je niz od N točaka uspio preokrenuti u $N/4$ prolaza kroz petlju. *Bit reversed* algoritam služi kako bi zamijenio centralno simetrične bitove i prilagodio ulazni niz točaka za izračun DFT-a. U tablici 4.3 prikazan je originalan i *bit reversed* niz od 8 točaka.

Tablica 4.3 Prikaz originalnog i *bit reversed* niza

Originalni niz		Bit reversed niz	
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

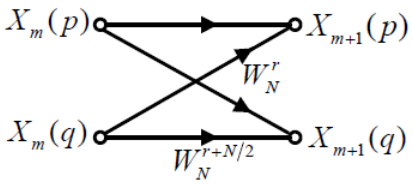
Yongov algoritam sastoji s od petlje koja se izvršava do $N/2$ indeksa, ali s korakom 2. Zatim se u slučaju da je indeks i, koji se povećava za broj 2 svakim prolaskom petlje, manji od indeksa j zamjenjuju (i, j) i $(i + N/2 + 1, j + N/2 + 1)$ parovi elemenata polja. Elementi $(i + 1, j + N/2)$ polja zamjenjuju se pri svakom prolasku petlje. Indeks j skalira se pomoću broja $N/4$.

4.2.2. Implementacija FFT Radix-2 DIT algoritma

Nakon zamjena poretka N elemenata ulaznog polja sve je spremno za računanje DFT-a. Ova *fixed-point* implementacija sastoji se od tri *for* petlje koje računaju DFT i dodatne petlje za računanje IDFT-a. Zbog cjelobrojnog množenja koristimo *lookup* tablicu. To je tablica koja sadržava sve diskretne vrijednosti sinusne funkcije u rasponu od -32768 do

32767, koristimo q15_t format cjelobrojnog zapisa u Q15.0 formatu koji ima raspon od -2^{15} do 2^{15}-1. Zbog *fixed-point* aritmetike kod množenja diskretnih vrijednosti iz *lookup* tablice i elemenata polja često dolazi do preljeva (*overflow*) pa trebamo voditi računa o skaliranju poslije množenja.

Prva *for* petlja određuje koliko stupnjeva ima DFT transformacija ($\log_2 N$) dok ostale dvije služe za unakrsno (*butterfly*) množenje elemenata polja. Na slici 4.1 je prikaz unakrsnog množenja dva elementa polja.



Slika 4.1 Unakrsno množenje dva elementa polja (Jeren, 2012)

Unakrsni množitelji $W_N^{N/2} = e^{-j\frac{2\pi}{N}\cdot\frac{N}{2}} = \cos(\pi) - jsin(\pi) = -1$ i tad su jednadžbe za izračun sljedeće

$$X_{m+1}(p) = X_m(p) + W_N^r X_m(q)$$

$$X_{m+1}(q) = X_m(p) + W_N^{r+\frac{N}{2}} X_m(q) = X_m(p) + W_N^r X_m(q).$$

Kao što vidimo rezultat DFT je uglavnom u kompleksnom obliku pa funkcija za izračun prima i realni i imaginarni niz ulaznih podataka, te broj točaka *N* DFT-a i jedinicu ili nulu koja označava da li se izračunava DFT ili IDFT. Zbog cjelobrojnog množenja $\sin(\pi)$ nije reprezentacija jedinice nego najveći broj u *fixed-point* rasponu, a to je 32767 pa zbog toga imamo pogrešku kod množenja. Ona se javlja zbog skaliranja odnosno dijeljenja pomnoženih brojeva s 2^{15}. Skaliranje je potrebno da se ne prekorači raspon *fixed-point* brojeva. Pojava ovakve pogreške je logična jer skaliramo broj 2^{15}-1 s 2^{15} pa izgubimo jedan bit. Kod razlaganja DFT-a na stupnjeve, prva dva stupnja su reprezentacija sinusa i kosinusa koji su 1, -1 i 0 pa zbog toga u petljama provjeravamo da li se nalazimo unutar ta dva stupnja i ne izvršavamo množenje nego samo proslijedimo odgovarajuće elemente polja. Kod množenja sa sinusom i kosinusom pomak kod DFT je u negativnom smjeru, a pomak kod IDFT-a je u pozitivnom. Algoritam koji brine da bi se dobro kretali po *lookup* tablici sinusa izračunava njegov korak tako da ukupni broj elemenata *lookup* tablice pomiče u desno za $\log_2 M$ gdje je M broj koji je potencija broja 2, odnosno svakim

sljedećim stupnjem DFT-a se broj elemenata tablice dijeli s sljedećom potencijom broja 2. Izgled *lookup* sinus tablice od $N = 8$ elemenata dan je u izvornom kodu.

```
#define N 8
q15_t sine[N]= {      0,  23170,  32767,  23170,
                      0, -23170, -32768, -23170 };
```

Potrebne *lookup* tablice generirali smo pomoću MATLAB programskog alata.

U tablici 4.4 je prikazana usporedba vrijednosti trigonometrijskih funkcija i reprezentacija sinusa i kosinusa u *lookup* tablici, također sa $N = 8$ točaka.

Tablica 4.4 Usporedba vrijednosti trigonometrijskih funkcija i reprezentacija sinusa i kosinusa u *lookup* tablici

kut α	sin(α)	cos(α)	korak k	sin[k]	cos[k]=sin[k+N/4]
0°	0	1	0	0	32767
45°	$\sqrt{2}/2$	$\sqrt{2}/2$	1	23170	23170
90°	1	0	2	32767	0
135°	$\sqrt{2}/2$	$-\sqrt{2}/2$	3	23170	-23170
180°	0	-1	4	0	-32768
225°	$-\sqrt{2}/2$	$-\sqrt{2}/2$	5	-23170	-23170
270°	-1	0	6	-32768	0
315°	$-\sqrt{2}/2$	$\sqrt{2}/2$	7	-23170	23170

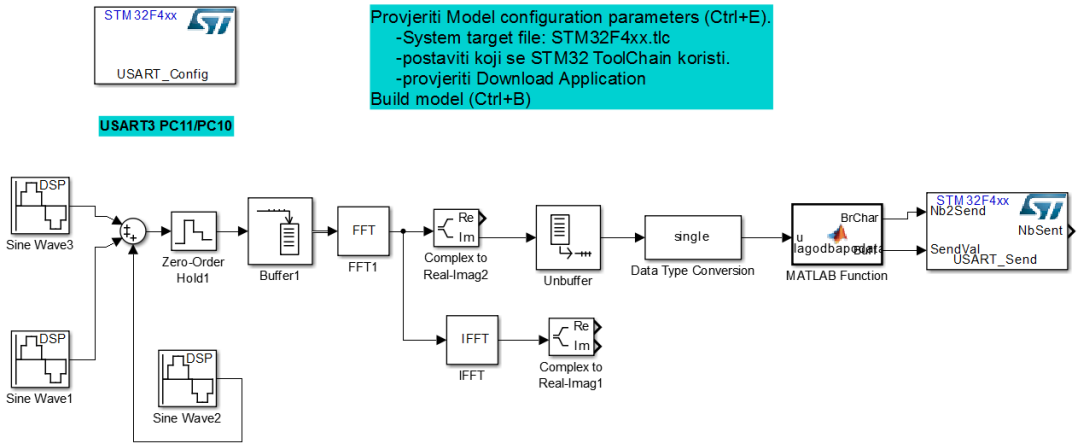
Kod računanja IDFT-a algoritam je jednak kao i za DFT samo je dodana jedna *for* petlja koja se izvršava do N indeksa i ona skalira sve elemente realnog i imaginarnog polja s N. U privitku se nalazi C/C++ funkcija za računanje DFT-a u N točaka.

Dobivene rezultate DFT-a i IDFT-a šaljemo na računalo za daljnju obradu i prikaz putem USART komunikacije. Koristimo USART3 s brzinom od 115200 bita po sekundi. Priključci za slanje (TX) i primanje (RX) podataka su pinovi PC10 i PC11.

4.3. Implementacija DFT-a u MATLAB programskom alatu na STM32F4 mikrokontroleru

U MATLAB programskom alatu razvili smo implementaciju DFT-a i IDFT-a pomoću Simulinka i MATLAB *embedded coder paketa*. Koristili smo MATLAB R2013a. U Simulinku smo koristili biblioteku *Target Support Package – STM32F4xx Adapter* koja prilagođava Simulink kod u Keil µVision4 kod koji je spreman za implementaciju na STM32F4 mikrokontroler.

Kao i u prethodnoj verziji koristili smo *fixed-point* aritmetiku odnosno cjelobrojne tipove podataka sa 16 bitova. Simulink shema dana je na slici 4.2.



Slika 4.2 Prikaz Simulnik sheme DFT-a na STM32F4 mikrokontroleru

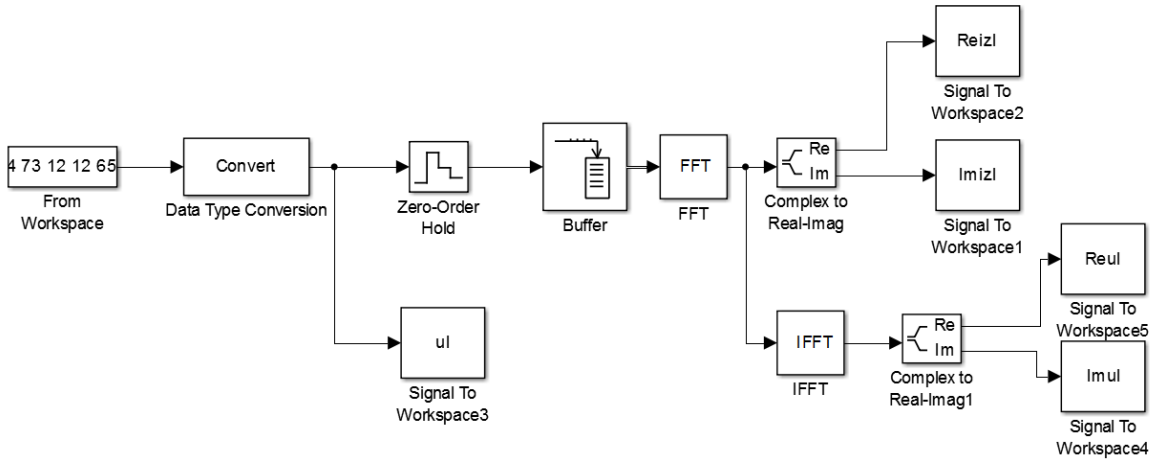
Na ovoj slici kao izvor za provjeru ispravnosti DFT-a koristimo zbroj tri sinusa različitih frekvencija. Kako DFT računa samo diskretne nizove točaka, sinuse različitih frekvencija ili bilo koje druge funkcije trebamo otipkati pomoću Zero-Order Hold bloka. Nakon otipkavanja podatke spremamo u spremnik (*buffer*) koji dalje šalje podatke u FFT blok koji računa DFT. FFT blok smo postavili da koristi radix-2 FFT algoritam. Kako je rezultat FFT-a diskretno kompleksno polje, razdvajamo realni i imaginarni dio tog polja pomoću Complex to Real-Imag bloka. Poslije FFT signal se grana u dvije grane. Jedna grana je Complex to Real-Imag blok, a druga na IFFT blok gdje se računa IDFT. Iz Simulink sheme vidljivo je da su tri od četiri priključka Complex to Real-Imag blokova u „zraku“. To je zbog toga što se Simulink shema izvodi paralelno i ne bi znali koji podaci su od kojeg signala. To bi možda moglo biti riješeno tako da se dodaju nekoliko bitova za

raspoznavanje signala, ali zbog brzine komunikacije to nije praktično rješenje pa smo odlučili da je bolje gledati signal po signal.

Brzina komunikacije je također kao i kod C/C++ verzije DFT implementacije 115200 bita po sekundi. Nakon Complex to Real-Imag blokova spajamo *unbuffer* koji taj niz podataka šalje van istom frekvencijom kojom je taj signal ušao u *buffer*. Ova ostala tri bloka služe za prilagodbu i slanje signala pomoću USART komunikacije.

5. Analiza DFT-a

U ovom poglavlju usporedit ćemo dvije verzije DFT implementacije na STM32F4 mikrokontroleru s MATLAB Simulink verzijom. Sve tri verzije izvode se u *fixed-point* aritmetici. Ulazni niz za ulaz u DFT generirali smo u MATLAB-u pomoću funkcije *randi*. Funkcija *randi* generira nasumičan niz od osam brojeva u rasponu od 1 do 100. Kod C/C++ implementacije upisujemo brojeve u obično polje od 8 znakova, dok kod Simulink verzije koristimo dodatni blok From Workspace u koji smo upisali generirane brojeve. Simulink verzija koja se izvodi u MATLAB programskom alatu je prikazana na slici 5.1.



Slika 5.1 Simulink shema DFT-a

U tablici ispod vidimo ulazni niz i rješenja DFT implementacije u Simulinku.

Tablica 5.1 Ulazni niz i rješenja DFT implementacije u Simulinku

Ulazni niz	69	14	73	12	12	65	33	66
Izl. realni niz	344	59	-25	55	30	55	-25	59
Izl. imaginarni niz	0	34	-1	114	0	-114	1	-34
Realni ul. niz	69	14	73	12	12	65	33	65
Imaginarni ul. niz	0	-1	0	-1	0	0	0	0

Ulazni niz sastoji se samo od realnih brojeva. Izlazni realni i imaginarni nizovi su rješenja DFT funkcije izvedene u Simulinku, dok su realni i imaginarni ulazni nizovi u stvari IDFT funkcija izlaznih realnih i imaginarnih nizova. Vidljivo je da je kod ove implementacije maksimalna apsolutna pogreška jedan bit.

Sljedeća implementacija je DFT implementacija pomoću Simulinka na STM32F4 mikrokontrolera. Rezultati će također biti prikazani pomoću tablice 5.2.

Tablica 5.2 Ulazni niz i rješenja DFT implementacije pomoću Simulinka na STM32F4 mikrokontroleru

Ulazni niz	69	14	73	12	12	65	33	66
Izl. realni niz	344	59	-25	54	30	54	-25	59
Izl. imaginarni niz	0	34	-1	114	0	-114	1	-34
Realni ul. niz	69	14	73	12	12	65	33	66
Imaginarni ul. niz	0	0	0	0	0	0	0	0

Kao što vidimo realni i imaginarni niz poslije DFT i IDFT-a je jednak kao i ulazni niz. Razlika između ove dvije verzije je najviše jedan bit.

Rezultati C/C++ verzije DFT implementacije su vidljivi u tablici 5.3.

Tablica 5.3 C/C++ verzija DFT implementacije

Ulazni niz	69	14	73	12	12	65	33	66
Izl. realni niz	341	58	-24	56	31	56	-24	58
Izl. imaginarni niz	0	33	-2	113	0	-113	2	-33
Realni ul. niz	68	13	72	12	12	64	32	64
Imaginarni ul. niz	0	0	0	0	0	0	0	0

Uspoređujući dobivene rezultate vidimo da je apsolutna maksimalna razlika između brojeva ove tri verzije jednaka 3 što je gledano po bitovima razlika u 2 bita.

Usporedba brzine izvođenja verzija i zauzetost memorije mikrokontrolera prikazani su na tablici 5.4.

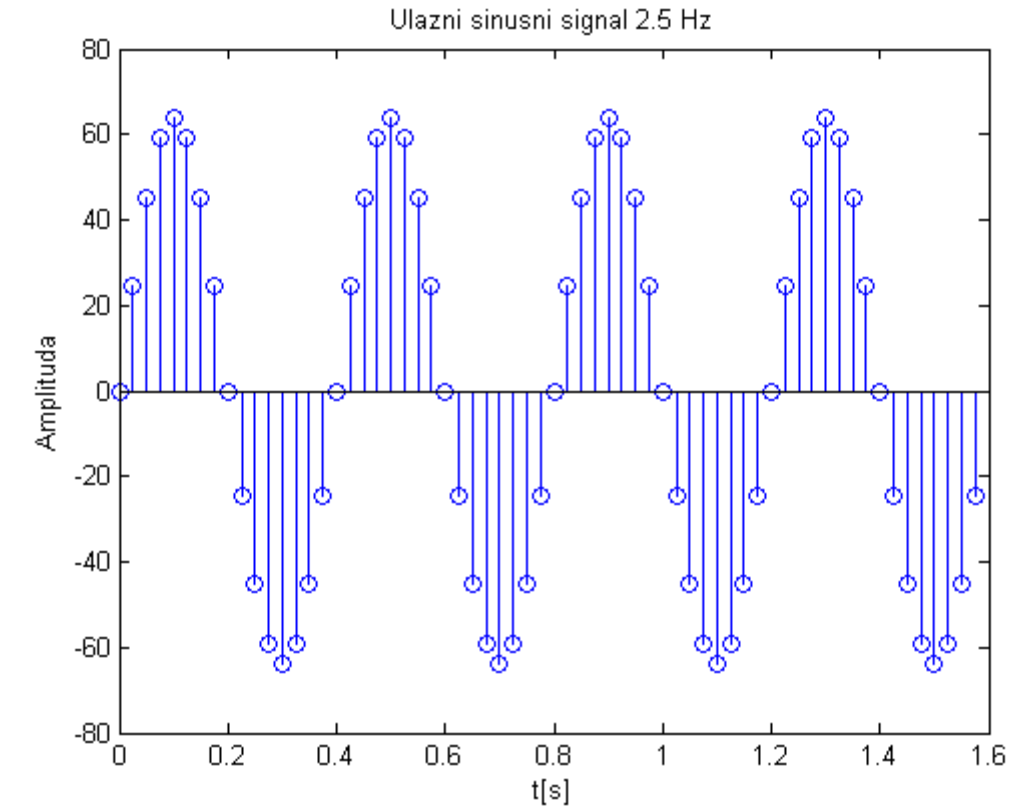
Tablica 5.4 Brzine izvođenja i zauzetost memorije

Brzina izvođenja [μs]	DFT	IDFT
C/C++ verzija	1210.30	1211.90
MATLAB verzija	435.20	643.80
Zauzeće memorije[B]		
C/C++ verzija	7640	
MATLAB verzija	13888	

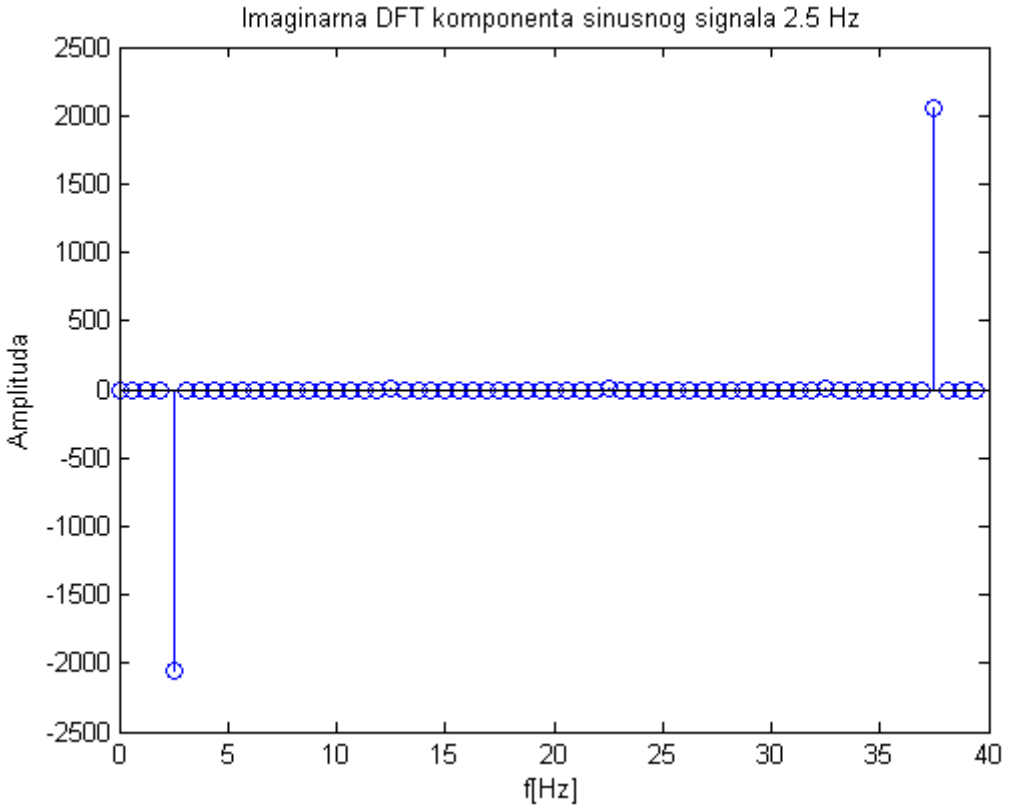
Vidimo da je implementacija u C/C++ programskom jeziku sporija i da zauzima manje memorije. MATLAB verzija je jednostavnija što se tiče programiranja jer se ne treba paziti na inicijalizaciju sklopovlja već MATLAB i biblioteka *Target Support Package – STM32F4xx Adapter* to rade sami.

5.1. Analiza računanja DFT-a nad sinusnim signalima

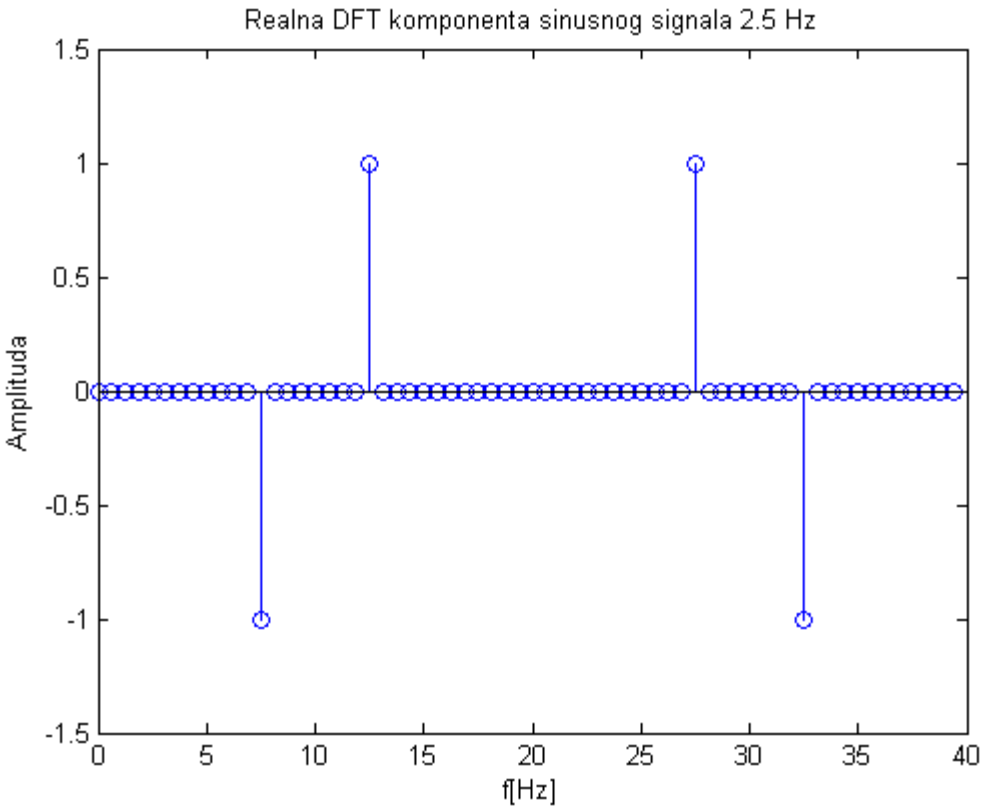
U ovom poglavlju usporedit ćemo ponašanja DFT implementacija nad različitim sinusnim signalima. Za početak uzet ćemo sinusni signal frekvencije 2.5 Hz otipkan frekvencijom od 40 Hz. Važno je spomenuti da signali moraju biti otipkani barem duplo većom frekvencijom od frekvencije samog signala, a to nam određuje Nyquistov teorem otipkavanja. Shema je jednaka kao i prethodne samo je umjesto bloka From Workspace kao izvor stavljen blok Sine Wave. Na slikama ispod su prikazani rezultati dobivene DFT transformacije u 64 točke.



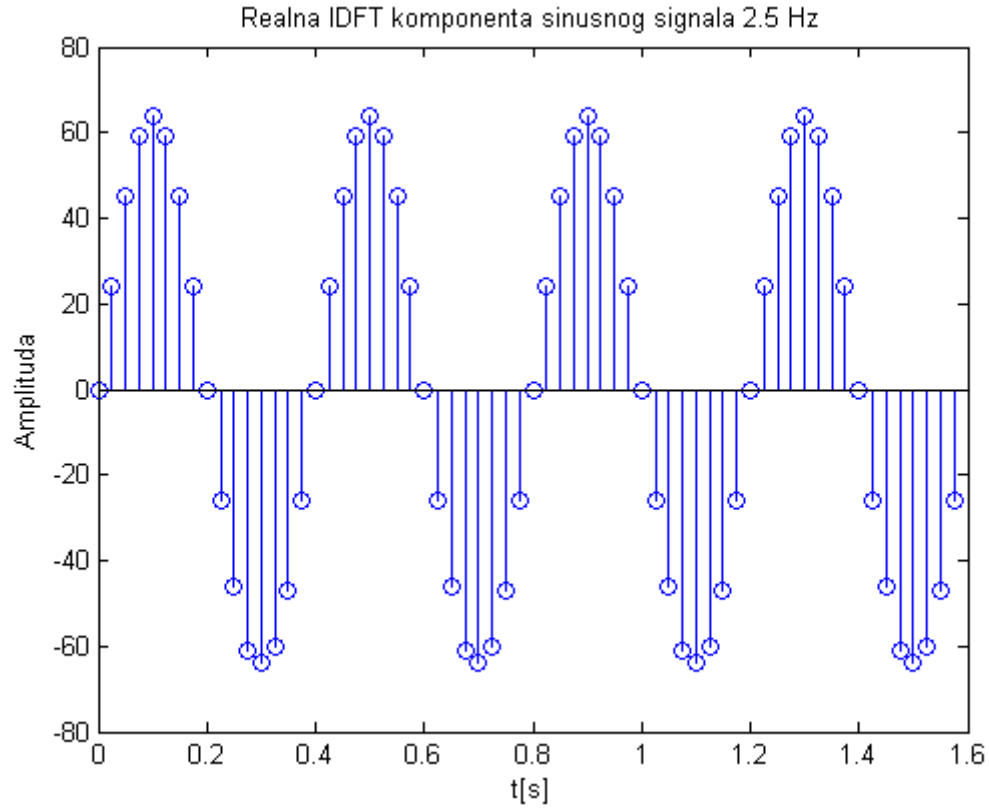
Slika 5.2 Ul. sinusni signal, f=2.5Hz, A=64



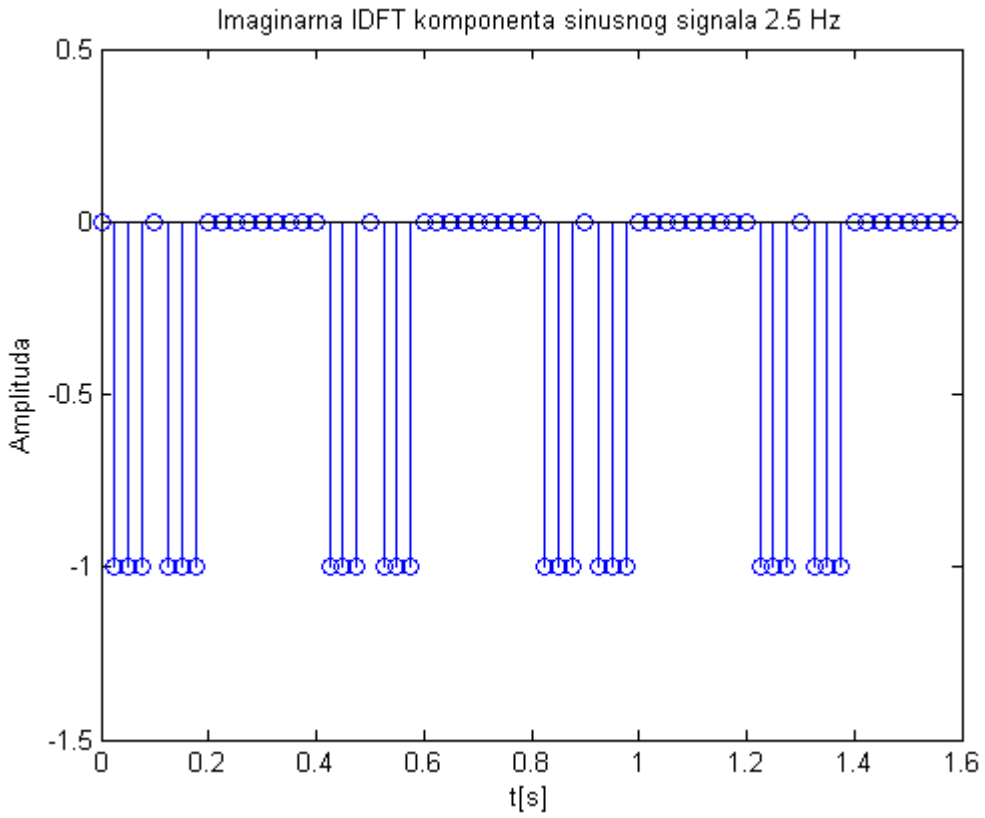
Slika 5.3 Imaginarna DFT komponenta sinusnog signala, f=2.5Hz, A=64



Slika 5.4 Realna DFT komponenta sinusnog signala, f=2.5 Hz, A=64



Slika 5.5 Realna IDFT komponenta sinusnog signala, f=2.5 Hz, A=64



Slika 5.6 Imaginarna IDFT komponenta sinusnog signala, f=2.5 Hz, A=64

Verzija koja se računa u MATLAB programskom alatu i MATLAB verziju koja je implementirana na STM32F4 mikrokontorleru razlikuju se najviše u jednom bitu i to kod računanja IDFT-a.

Maksimalna razlika C/C++ verzije i MATLAB verzije iznosi u brojevima 2 i to je također pogreška u 2 bita.

U Tablici 5.5, 5.6 i 5.7 usporedit ćemo brzinu izvođenja, zauzeće memorije i prosječan broj ciklusa procesora potrebnih za izračun DFT-a za različite duljine DFT transformacije. Procesor radi na frekvenciji 10 MHz. Iz tablice 5.5 možemo očitati da s povećanjem broja duljine DFT transformacije povećava vrijeme izvođenja same transformacije. MATLAB verzija normalno radi do duljine $N = 128$. Ne radi na većim duljina vjerojatno zbog toga što se sve izvodi paralelno. C/C++ verzija normalno radi i na većim duljinama (testirano do $N = 1024$). Možemo primijetiti da je izvođenje MATLAB verzije kraće od C/C++ verzije implementacije DFT-a.

Tablica 5.5 Vrijeme izvođenja DFT transformacije

Vrijeme izvođenja				
	C/C++ verzija		MATLAB verzija	
	DFT	IDFT	DFT	IDFT
N=8	1.214 ms	1.224 ms	159.66 µs	188.57 µs
N=16	2.490 ms	2.508 ms	283.21 µs	375.04 µs
N=32	5.407 ms	5.055 ms	561.23 µs	824.30 µs
N=64	10.248 ms	10.236 ms	1.174 ms	1.823 ms
N=128	20.563 ms	20.582 ms	2.585 ms	4.238 ms
N=256	41.865 ms	42.100 ms	-	-
N=512	84.386 ms	85.058 ms	-	-
N=1024	172.598 ms	172.403 ms	-	-

S povećanjem broja N točaka DFT povećava se zauzeće memorije. MATLAB verzija zauzima više memorije.

Tablica 5.6 Zauzeće memorije mikrokontrolera

Zauzeće memorije [B]		
	C/C++ verzija	MATLAB verzija
N=8	7896	12112
N=16	7896	12132
N=32	7896	12172
N=64	7896	12240

N=128	10100	12384
N=256	10220	-
N=512	10732	-
N=1024	11668	-

S povećanjem broja N točaka DFT povećava se broj ciklusa procesora. C/C++ verzija ima veći broj ciklusa procesora.

Tablica 5.7 Broj ciklusa procesora

Broj ciklusa procesora		
	C/C++ verzija	MATLAB verzija
N=8	12141	1596
N=16	24904	2832
N=32	54070	5612
N=64	102482	11740
N=128	205631	25850
N=256	418653	-
N=512	843862	-
N=1024	1725983	-

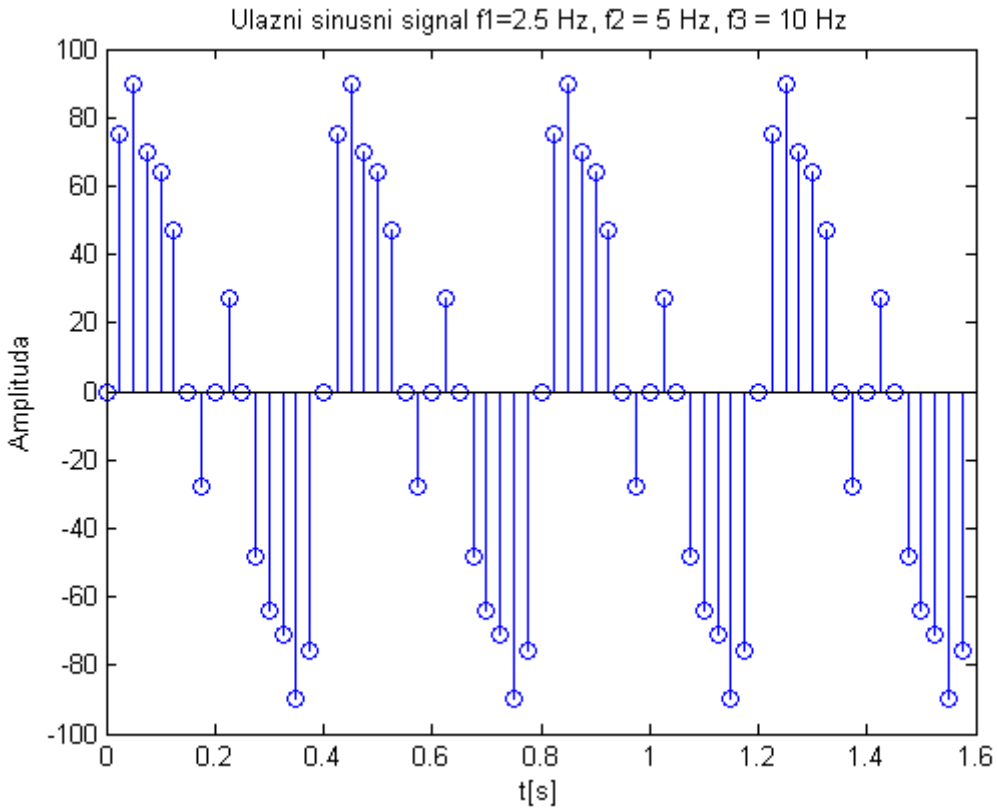
STM32F4 mikrokontroler ima prilagođeno sklopovlje za obradu signala te podržava DSP instrukcije. U Tablici 5.8 prikazani su službeni podaci vremena izvođenja i broja ciklusa procesora FFT implementacije (frekvencija rada jezgre procesora je 168 MHz, vrijeme

izvođenja ovih naših implementacija dijeli se s 16.8) koja se nalazi u STM32F4 DSP biblioteci. Vidimo da su rezultati puno bolji od ovih dviju implementacija.

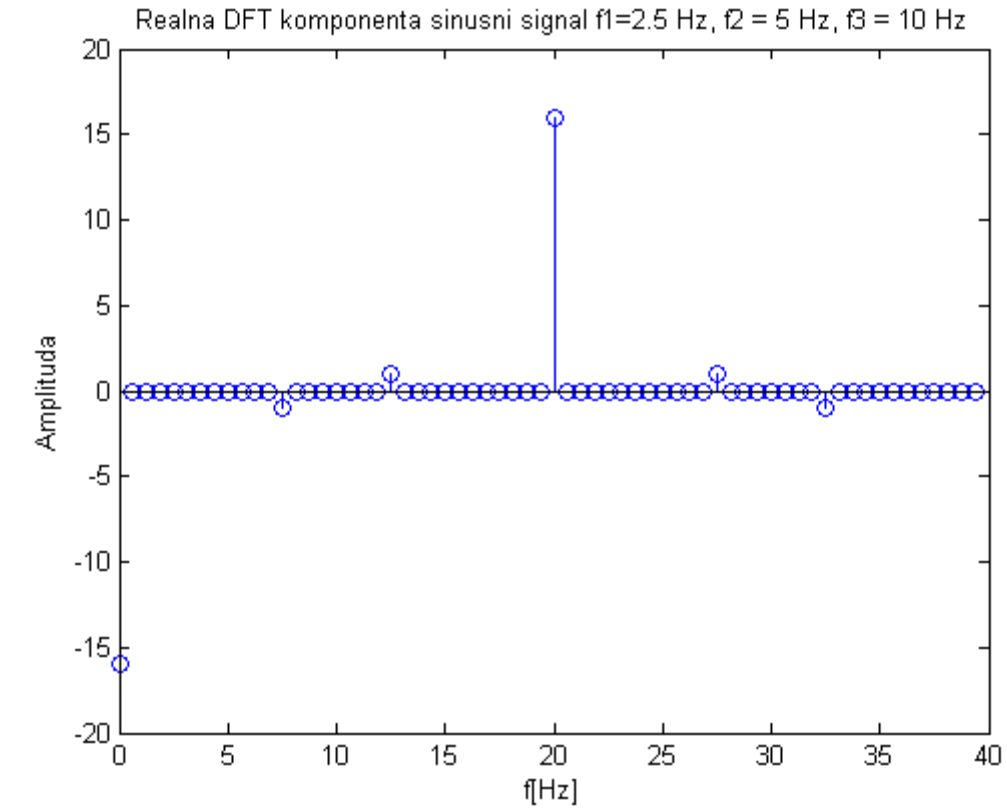
Tablica 5.8 Vrijeme izvođenja i broj ciklusa implementacije FFT-a pomoću DSP biblioteke

	Broj ciklusa	Vrijeme izvođenja[μs]
N=64	3537	22.101
N=1024	82174	496.952

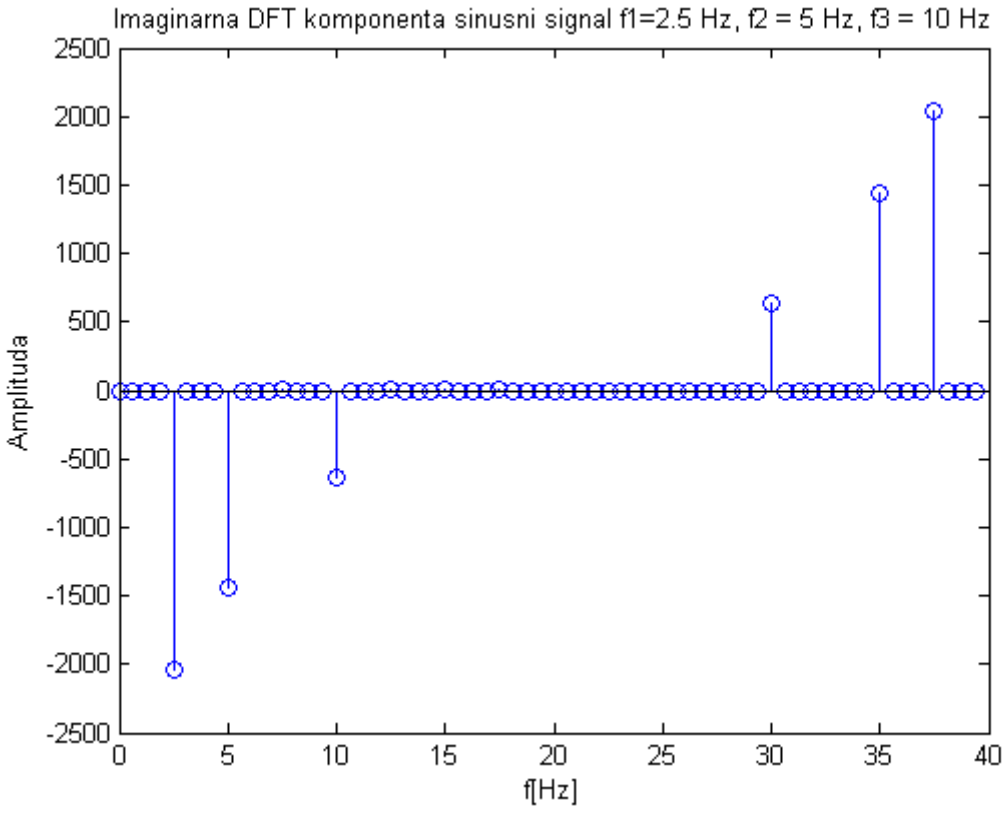
Drugi signal kojeg ćemo iskoristiti za testiranje implementacije DFT-a na STM32F4 mikrokontroleru je zbroj tri sinusa različitih frekvencija, $f_1 = 2.5\ Hz$ i $A_1 = 64$, $f_2 = 5\ Hz$ i $A_2 = 45$ te $f_3 = 10\ Hz$ i $A_3 = 20$. Kao i u prethodnom testiranju koristimo DFT u 64 točaka. Na slikama (5.7, 5.8, 5.9) ispod prikazani su rezultati.



Slika 5.7 Ulazni sinusni signal, $f_1 = 2.5\ Hz$, $A_1 = 64$, $f_2 = 5\ Hz$, $A_2 = 45$,$f_3 = 10\ Hz$, $A_3 = 20$

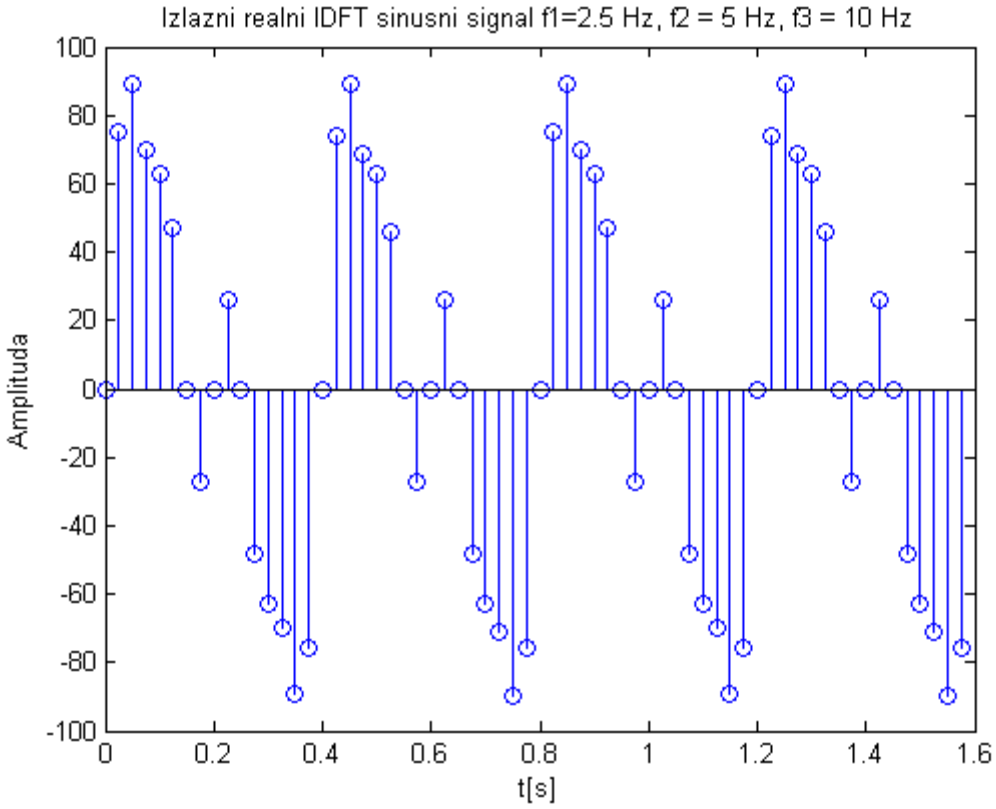


Slika 5.8 Realna DFT komponenta sinusnog signala, $f_1 = 2.5\ Hz$, $f_2 = 5\ Hz$, $f_3 = 10\ Hz$

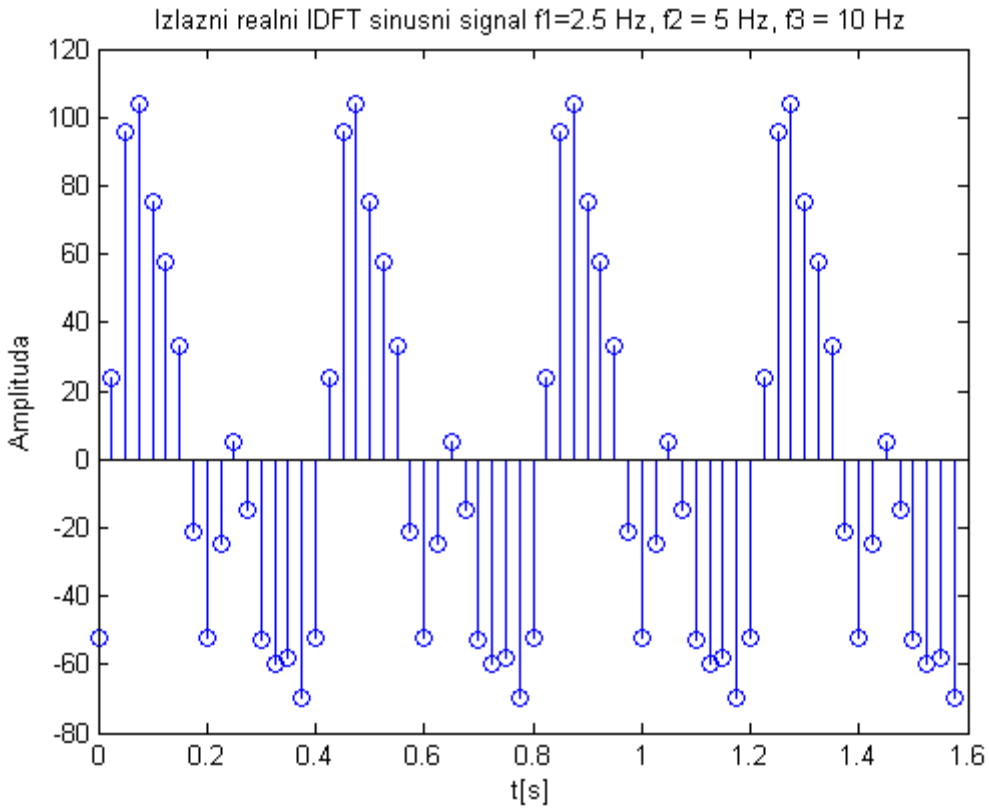


Slika 5.9 Imaginarna DFT komponenta sinusnog signala, $f_1 = 2.5\ Hz$, $f_2 = 5\ Hz$, $f_3 = 10\ Hz$

Apsolutna maksimalna razlika rezultata DFT-a je 7 što u bitovnom svijetu znači pogreška u tri najmanje značajna bita. Slijede slike (5.10 i 5.11) na kojima je vidljiv sinusni signal dobiven od IDFT gore dobivenih transformacija. Imaginarni dio IDFT nećemo prikazivati je on jednak nuli. Kod MATLAB verzije javljaju se male oscilacije (+/- 1) imaginarnog dijela IDFT-a.



Slika 5.10 Izlazni realni IDFT sinusni signal, C/C++ verzija



Slika 5.11 Izlazni realni IDFT sinusni signal, MATLAB verzija

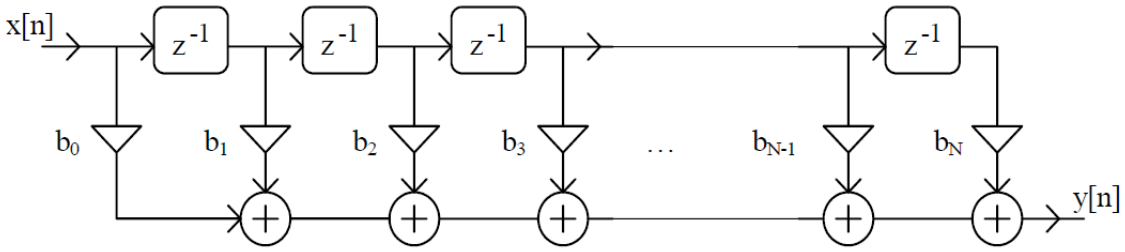
Iz ne objašnjivih razloga pojavila se velika pogreška kod MATLAB verzije implementirane na STM32F4 mikrokontroleru. Maksimalna pogreška dolazi čak i do broja 52, što proračunato u bitove iznosi pogrešku od 6 bitova. Smanjenjem broja N točaka IDFT transformacije na $N = 32$ dobili smo izlazni realni niz jednak kao i ulazni niz.

6. Nerekurzivni digitalni filtri

Nerekurzivni (FIR - *finite impulse response*) digitalni filtri su diskretni LTI (linearni, vremenski nepromjenjivi) sustavi s konačnom duljinom impulsnog odziva koji služe za usrednjavanje slučajnih varijacija signala (Jeren, 2012). Odziv FIR filtra u nekom trenutku n dobivamo konvolucijom njegovog impulsnog odziva i prošlih ulaznih uzoraka filtra. S povećanjem reda filtra smanjuje se prijelazno područje filtra. Izlaz iz FIR filtra računamo kao

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k]$$

pri ćemo je L duljina njegovog impulsnog odziva koji izračunavao kao $L = N + 1$, gdje je N red filtra. Na slici 6.1 je prikazana direktna realizacija filtra.



Slika 6.1 Direktna realizacija FIR filtra (Petrinović, 2003)

Mogućnost projektiranja ovakvih filtera je moguće pomoću metode s vremenskim otvorom (Petrinović, 2003). Prvo je potrebno odrediti željenu idealnu frekvencijsku karakteristiku filtra $H_{id}(e^{j\omega})$ kako bi dobili njegov idealni impulsni odziv, a to je moguće izračunati pomoću inverzne Fourierove transformacije koja je oblika

$$h_{id}[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_{id}(e^{j\omega})e^{jn\omega}d\omega, \qquad , -\infty < n < \infty.$$

S obzirom na to da je idealni impulsni odziv beskonačan, izvedba idealnog filtra nije moguća pa zbog toga impulsni odziv množimo s vremenskim otvorom

$$h[n] = h_{id}[n] \cdot w[n].$$

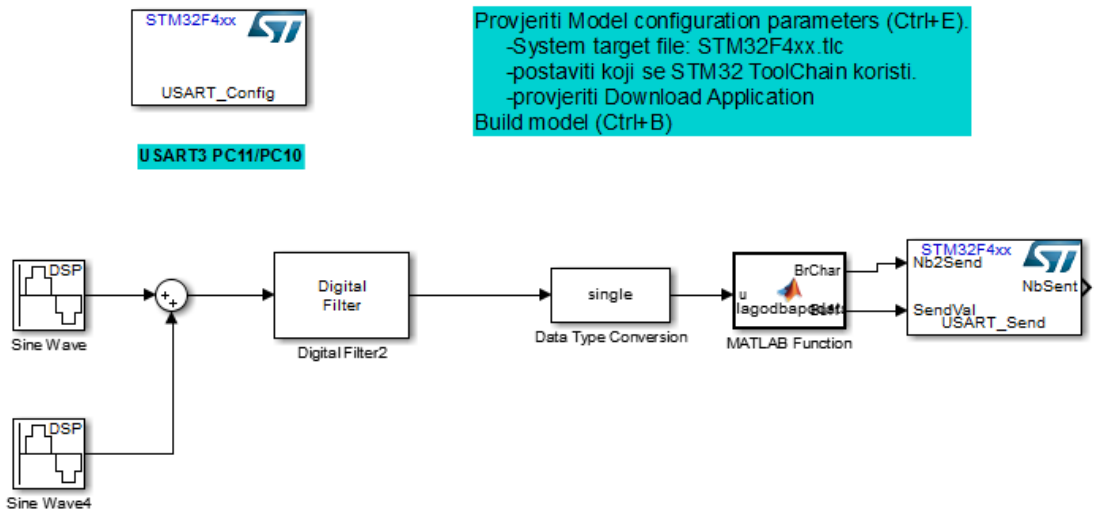
No taj ograničeni impulsni odziv je i dalje nekauzalan pa ga je potrebno pomaknuti u desno za pola uzoraka.

Imamo različite tipove filtara: niski propust, visoki propust, pojasni propust i pojasna brana. Za svaki tip filtra imamo različitu formulu za izračun impulsnog odziva.

Kako ima formula za izračun impulsnih odziva filtra, tako ima i formula za izračun vremenskih otvora. Imamo nekoliko tipova vremenskih otrova: pravokutni, Bartlettov (trokutni), Hannov, Hammingov, Blackmanov, Dolph-Chebyshevljev i Kaiserov. Ovi tipovi vremenskih otvora su fiksni osim Dolph-Chebyshevljevog i Kaiserovog koji su promjenjivi vremenski otvori.

6.1. Implementacija FIR filtra

Razvili smo dvije verzije implementacije FIR filtara, jedna napisana u C/C++ programskom jeziku i jedna razvijena u MATLAB Simulink programskom alatu. Implementirali i testirali smo samo dvije verzije FIR filtara, niskopropusni i visokopropusni filtar. Zbog jednostavnosti provjere rezultata, a i same implementacije odabrali smo red filtra $N = 15$ za niski propust (NP) i $N = 16$ za visoki propust (VP). Simulink verzija sastoji se od ulaznog izvora, sinusnih signala, diskretnog FIR filtra i blokova za prilagodbu podataka za prijenos podataka preko USART komunikacijskog sučelja. Na slici 6.2 je prikazan Simulink model implementacije FIR filtra.



Slika 6.2 Simulnik model FIR filtra

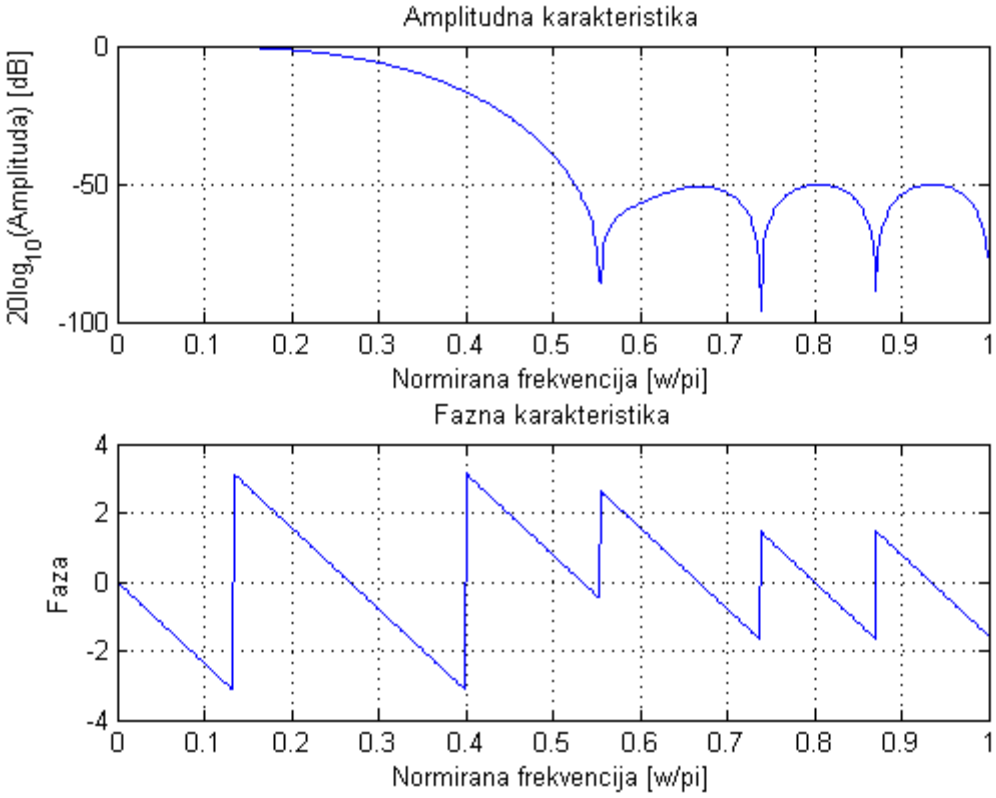
Koeficijente FIR filtra izračunali smo pomoću MATLAB funkcije *fir1*, vremenski otvor kojeg koristimo je Hammingov otvor. Diskretni filtar razvijen je u direktnoj realizaciji filtra. Kao i kod DFT implementacije koristimo cjelobrojnu aritmetiku u Q15.0 notaciji.

Implementacija FIR filtra u C/C++ programskom jeziku također koristi koeficijente filtra izračunate u MATLAB programskom alatu koje smo skalirani s 2^{15} kako bi obuhvatili cijeli raspon cjelobrojnog formata. Funkcija pomoću koje je implementirani FIR filtar kao argument prima ulazno polje, izlazno polje i broj elemenata ulaznog polja. Direktna realizacija realizirana je pomoću jedne *for* petlje i cirkularnog pomaka prošlih uzoraka polja. U privitku se nalazi C/C++ funkcija za računanje visokopropusnog FIR filtra reda $N = 16$.

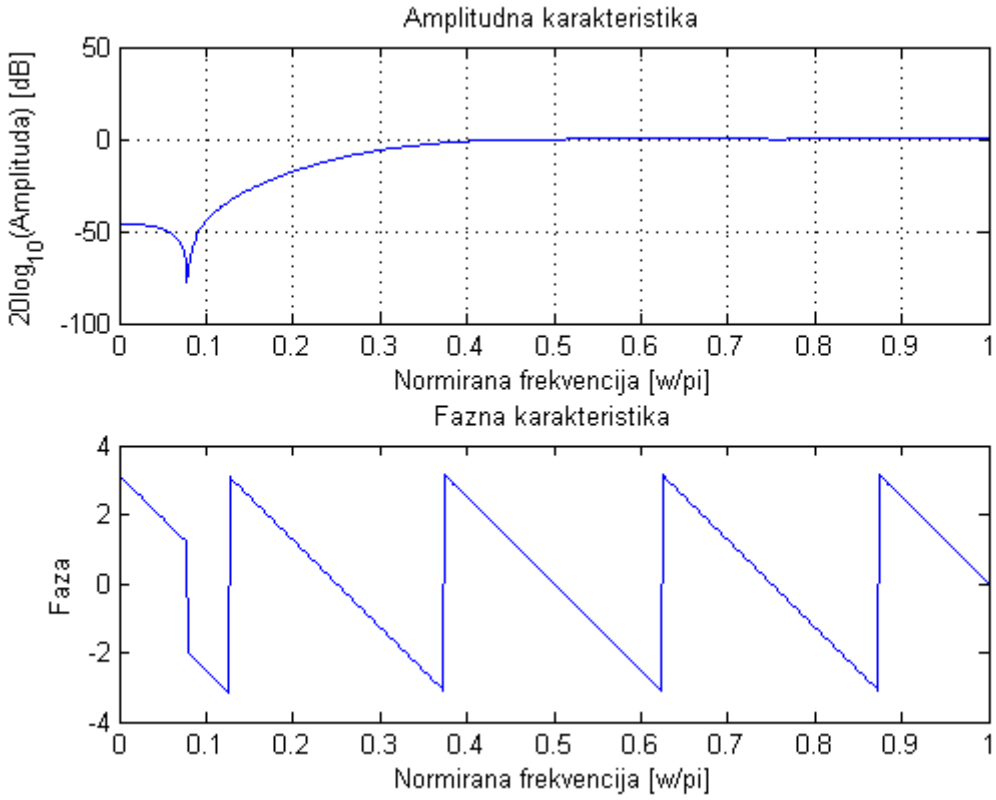
Kao i kod DFT verzije podaci se na računalo šalju pomoću USART komunikacijskog sučelja, te se kasnije podaci prikazuju kao grafovi pomoću MATLAB funkcije *stem*.

6.2. Analiza FIR filtara

Kao što je spomenuto razvili smo dvije verzije FIR filtara, niskopropusni i visokopropusni FIR filtar. Red filtra je 15 za NP odnosno 16 za VP, i broj koeficijenata filtra 16 odnosno 17. Implementirali smo FIR filtar pomoću Hammingovog vremenskog otvora. Amplitudne i fazne karakteristike niskopropusnog i visokopropusnog FIR filtra prikazane su na slici 6.3 i 6.4. Granična frekvencija za oba filtra je $\omega_c = 0.3 \cdot \pi$.

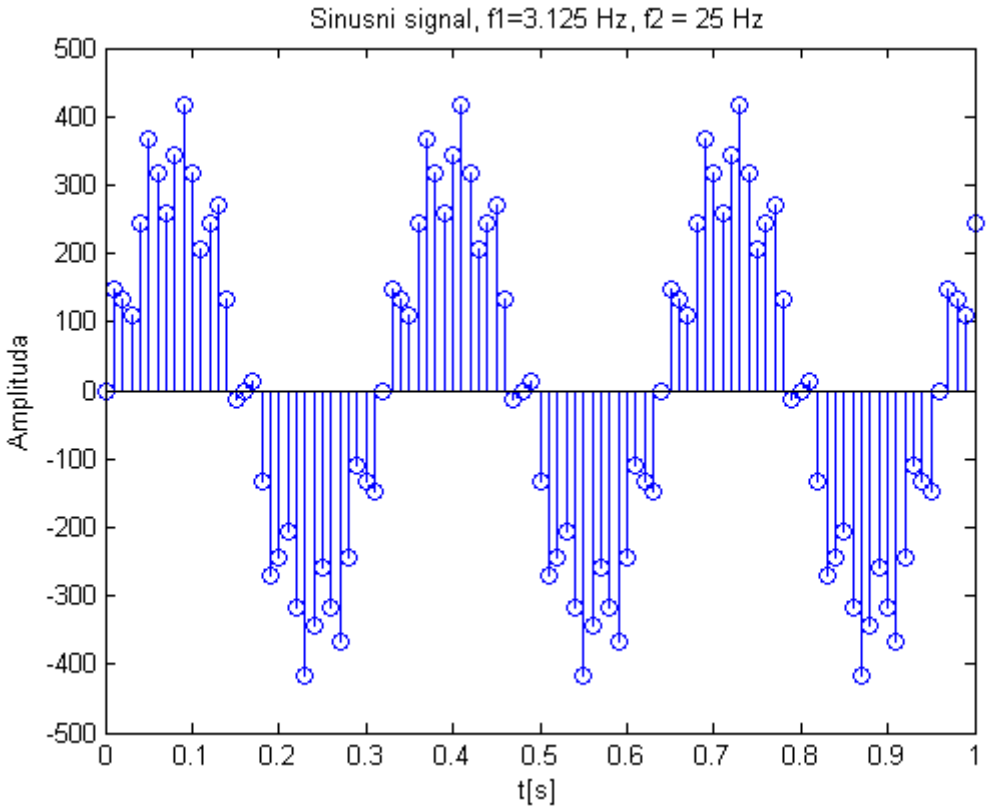


Slika 6.3 Karakteristika niskopropusnog FIR filtra, red $N = 15$

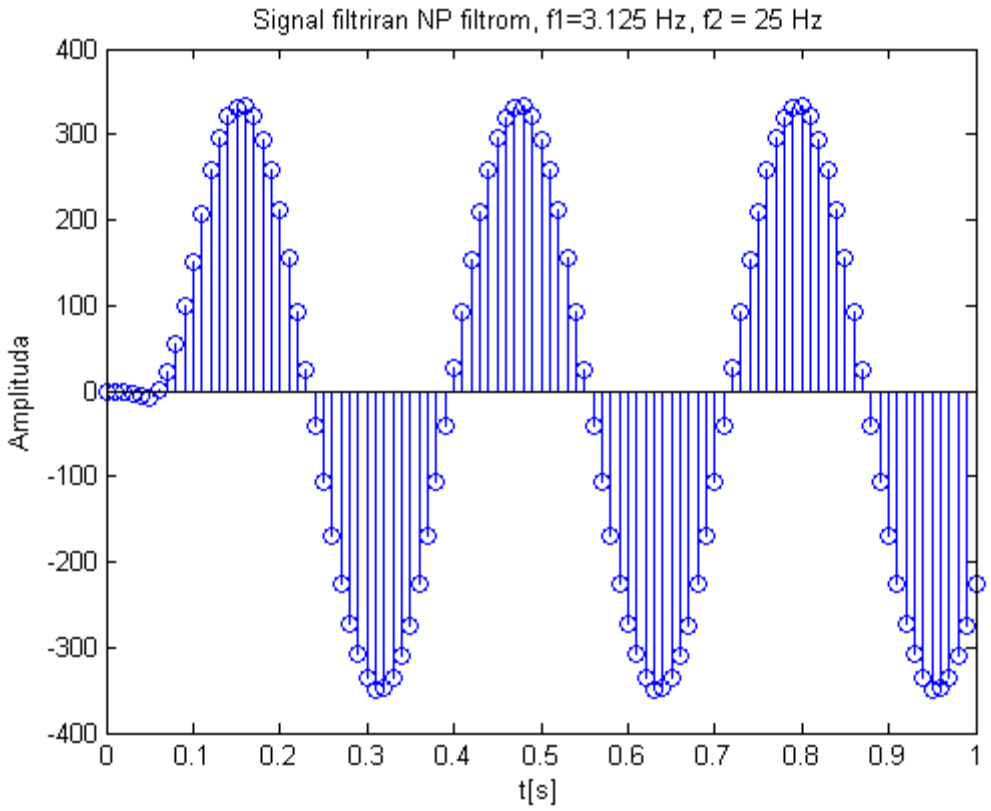


Slika 6.4 Karakteristika visokopropusnog FIR filtra, red $N = 16$

Kao testni primjerak izvora uzeli smo zbroj sinusa frekvencije 3.125 Hz, amplitude 345 i sinus frekvencije 25 Hz, amplitude 80 otipkanih s frekvencijom od 100 Hz. Kako smo uzeli da je granična frekvencija $0.3 \cdot \pi$ odnosno $0.3 \cdot f_s/2$ za očekivati je da niskopropusni filtar propušta samo frekvenciju 3.125 Hz, dok visokopropusni filtar propušta samo frekvenciju 25 Hz. Izgled ulaznog sinusnog signala prikazan je na slici 6.5, a izlaz iz niskopropusnog FIR filtra prikazan je na slici 6.6. Procesor radi na frekvenciji 10 MHz.

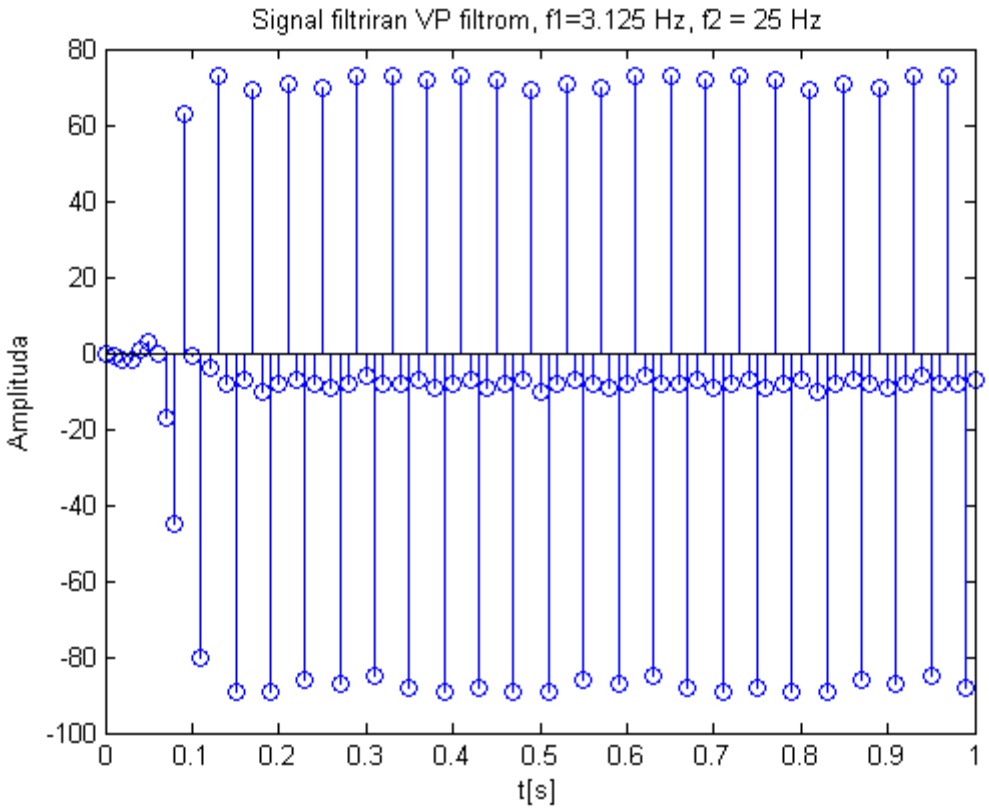


Slika 6.5 Sinusni signal, $f_1 = 3.125\ Hz$, $A_1 = 345$, $f_2 = 25\ Hz$, $A_2 = 80$



Slika 6.6 Izlaz NP filtra sinusnog signala, $f_1 = 3.125\ Hz$, $A_1 = 345$, $f_2 = 25\ Hz$, $A_2 = 80$

Izlaz iz visokopropusnog FIR filtra prikazan je na slici 6.7.



Slika 6.7 Izlaz VP filtra sinusnog signala, $f_1 = 3.125\ Hz$, $A_1 = 345$, $f_2 = 25\ Hz$, $A_2 = 80$

U tablici 6.1 je vidljiva razlika između dvije verzije implementacije NP FIR filtra u odnosu na vrijeme izvođenja, cikluse procesora i zauzeće memorije filtara.

Tablica 6.1 Usporedba NP FIR filtra

	Vrijeme izvođenja [µs]	Ciklusi procesora	Zauzeće memorije [B]
C/C++	30.3	303	3144
MATLAB	37,6	376	9224

Vidimo da je C/C++ verzija implementacije FIR filtra zauzima manje memorije, no broj ciklusa procesora i vrijeme izvođenja je gotovo jednako za obje implementacije. Usporedba dobivenih rezultati ovih implementacija razlikuju se najviše u jednom bitu.

Zaključak

U ovom radu implementirane su dvije verzije DFT transformacije i nerekurzivnih (FIR) filtara. Jedna verzija razvijena je u MATLAB programskom alatu pomoću Simulink modela i Embedded Coder paketa dok je druga napisana u C/C++ programskom jeziku.

DFT transformacija razvijena u MATLAB programskom alatu može maksimalno izračunavati do N=128 točaka, dok implementacija razvijena u C/C++ programskom jeziku normalno radi i na N=1024 točaka. Vrijeme izvođenja i broj ciklusa manji je kod verzije programirane u MATLAB-u, ali je zauzeće memorije manje kod C/C++ verzije.

Nerekurzivni filtar, reda N=15 za niskopropusni, odnosno N=16 za visokopropusni, razvijen u MATLAB programskom alatu ima veće zauzeće memorije od implementacije razvijene u C/C++ programskom jeziku. Vrijeme izvođenja i broj ciklusa procesora gotovo je jednak za obje verzije.

MATLAB verzija je puno jednostavnija za implementaciju, no ne radi s većim brojem točaka DFT i ne radi na većim redovima filtara. Treba uzeti u obzir i cijenu MATLAB programskog alata s obzirom na to da smo u C/C++ programskom alatu radi na besplatnoj inačici Keil µVision 4 koja podržava kodove do 32 kB.

Literatura

[1] MARIO KRNIĆ: "Fourierova analiza", skripta, svibanj 2010.

[2] D. SUNDARARAJAN: "The Discrete Fourier Transform", World Scientific, Singapore, 2001.

[3] K. R. RAO, D. N. KIM, J. J. HWANG: "Fast Fourier Transform", Springer, New York, 2010.

[4] TOR AAMODT, PAUL CHOW: "Numerical Error Minimizing Floating-Point to Fixed-Point ANSI C Compilation," MPDSP-1, pp. 3-12, Haifa Israel, studeni 1999.

[5] PAUL HOLDEN: "Develop FFT apps on low-power microcontrollers," listopad 2005.

[6] ALISTAIR ROBERTSON: "Implementing an FIR Filter on the MPC55xx," Freescale Semiconductor, kolovoz, 2007.

[7] D. PETRINOVIĆ, D. PETRINOVIĆ, R. BREGOVIĆ, H. BABIĆ, B. JEREN: "Digitalna obradba signala upute za laboratorijske vježbe," Zavod za elektroničke sustave i obradbu informacija, FER, Zagreb, 2003.

[8] FENG YU; ZE-KE WANG; RUI-FENG GE: "Novel algorithm for complex bit reversal: employing vector permutation and branch reduction methods," Journal of Zhejiang University SCIENCE A, listopad 2009.

[9] CHONGHUA LI: "Design and Realization of FIR Digital Filters Based on MATLAB," Coll. of Mech. & Electr. Eng., Guizhou Normal Univ., Guiyang, China, srpanj 2010.

[10] STM32F4DISCOVERY STM32F4 high-performance discovery board, STMicroelectronics, siječanj 2012.

[11] STM32F405xx STM32F407xx, ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera, STMicroelectronics, lipanj 2013.

[12] Reference manual, STM32F405xx/07xx, STM32F415xx/17xx, STM32F42xxx and STM32F43xxx advanced ARM-based 32-bit MCUs, STMicroelectronics, rujan 2013.

[13] STM32 Updates for technical development community, STMicroelectronics, 2012.

[14] MATLAB The Language of Technical Computing, MathWorks

[15] DIDIER: Fast Fourier Transform (FFT), s Interneta, http://www.arduinoos.com/2010/10/fast-fourier-transform-fft , 5. listopada 2010.

[16] Discrete Fourier transform, s Interneta, http://en.wikipedia.org/wiki/Discrete_Fourier_transform , 2. lipnja 2014.

[17] DOUGLAS L. JONES: Decimation-in-time (DIT) Radix-2 FFT, s Interneta, http://cnx.org/content/m12016/latest , 15. rujan 2006.

[18] B. JEREN, D. PETRINOVIĆ: "Prezentacije s kolegija Digitalna obradba signal", Zavod za elektroničke sustave i obradbu informacija, FER, Zagreb 2012

Razvoj programske potpore ugradbenog računalnog sustava pomoću MATLAB Embedded Coder paketa

Sažetak

U ovom radu opisana je implementacija diskretne Fourierove transformacije (DFT) i nerekurzivnih (FIR) diskretnih filtara koristeći ugradbeno računalo STM32F4 Discovery. Razvijene su dvije verzije. Jedna u C/C++ programskom jeziku, dok je druga razvijena u MATLAB programskom alatu pomoću Simulink modela i Embedded Coder paketa. Simulink i Embedded Coder paket omogućuju jednostavno prevođenje modela u izvorni kod ugradbenog računala. Kako bi ubrzali rad ugradbenog računala koristimo podatke u cjelobrojnom zapisu. Analizirali smo učinkovitost razvijenih programskih kodova sa stanovišta brzine izvođenja, zauzeća memorije i broja ciklusa procesora.

Ključne riječi: diskretna Fourierova transformacija (DFT), nerekurzini (FIR) filtar, STM32F4 Discovery, MATLAB Embedded Coder paket

Development of embedded system software using Matlab Embedded Coder toolbox

Abstract

This work describes the implementation of the Discrete Fourier Transform (DFT) and finite impulse response (FIR) discrete filters using microcontroller STM32F4 Discovery. Two version were developed. The first version was developed in C/C++ programming language, while the second one, was developed in MATLAB high-level language using the Simulink and Embedded Coder toolbox. Simulink and Embedded Coder toolbox easily transform Simulink model to source code of embedded system. To provide faster execution speed we are using fixed point arithmetic. We analyzed the efficiency of the developed software codes from the point of execution speed, memory consumption and CPU cycles.

Keywords: Discrete Fourier Transform (DFT), finite impulse response (FIR) filter, STM32F4 Discovery, MATLAB Embedded Coder toolbox

Privitak: Programski kod DFT i FIR funkcije

U privitku su dane funkcije napisane u C/C++ programskom jeziku.

Implementacija DFT funkcije

```
#define FFT 0x01
#define N_WAVE 64


/*zamjena elemenata polja*/
void zamjena(q15_t *x, q15_t *y)
{
        q15_t temp = *x;
        *x = *y;
        *y = temp;
}
/*lookup tabblica sinusa*/
q15_t sine[64]= { 0,  3212,  6393,  9512, 12540, 15447, 18205, 20788,
23170,25330,27246, 28899, 30274, 31357, 32138, 32610, 32767, 32610,
32138, 31357, 30274, 28899, 27246,25330, 23170, 20788, 18205, 15447,
12540, 9512, 6393, 3212, 0, -3212, -6393,-9512, -12540, -15447, -18205,
-20788, -23170, -25330, -27246, -28899,-30274, -31357, -32138, -32610,
-32768, -32610, -32138, -31357, -30274,-28899,-27246, -25330, -23170,
-20788, -18205, -15447, -12540, -9512, -6393, -3212 };
/*izracunavanje DFT/IDFT do 64 tocke*/
void fix_FFT(q15_t *Re, q15_t *Im, uint16_t N, uint8_t smjer)
{
          q15_t l2,l1,i1,j=0 ;
          q15_t ur,ui,c,tr,ti,k,x;
          uint16_t NB2,NB4,NBTP1;
          int log2N=log2(N);
          NB2=N/2;
          NB4=N/4;
          NBTP1=NB2+1;
              /* bit reversed algoritam */
             for (uint16_t i = 0; i < NB2; i+=2) {
                 if(i<j){
                    zamjena(&Re[i], &Re[j]);
                    zamjena(&Im[i], &Im[j]);
                    zamjena(&Re[i+NBTP1], &Re[j+NBTP1]);
```

```
                    zamjena(&Im[i+NBTP1], &Im[j+NBTP1]);
                      }
                     zamjena(&Re[i+1], &Re[j+NB2]);
                     zamjena(&Im[i+1], &Im[j+NB2]);
                     k = NB4;
                     while (k <= j) {
                              j -= k;
                              k >>= 1;}
                  j += k;}
/*izracun DFT pomocu FFT algoritma Radix-2:DIT(decimacija u vremenu)*/
          l2 = 1;
      for (uint16_t l = 0; l < log2N; l++) {
         l1 = l2;
         l2 <<= 1;
         ur   = sine[N_WAVE/4 ];  //koristenje lookup tablica
         ui   =  sine[0];
          for (j = 0; j < l1; j++) {
             for (uint32_t i = j; i < N; i += l2) {
                        i1 = i + l1;
                     if(l<2 && smjer==FFT){
                        tr=Re[i1];
                        ti=Im[i1];
                        Re[i1] = Re[i]  - tr;
                        Im[i1] = Im[i]  - ti;
                        Re[i] += tr;
                        Im[i] += ti;
                     }else if(l>0){
tr= (q15_t)((q15_t)(Re[i1]* ur >> 15)-(q15_t)(Im[i1]* ui >> 15));
ti= (q15_t)((q15_t)(Im[i1]* ur >> 15)+(q15_t)(Re[i1]* ui >> 15));
                           Re[i1] = Re[i] - tr;
                           Im[i1] = Im[i] - ti;
                           Re[i] =Re[i] + tr;
                           Im[i] =Im[i] + ti;
                      }else{   tr=Re[i1];
                               ti=Im[i1];
                               Re[i1] = Re[i]  - tr;
                               Im[i1] = Im[i]  - ti;
                               Re[i] += tr;
                               Im[i] += ti;}
            }
      /*racunanje pomaka u lookup tablici*/
      x=(log2(l1)+1);
      c=N_WAVE>>x;
      k =c*(j+1);
      ur = sine[k+N_WAVE/4];
         if (smjer == FFT){
                  ui= -sine[k];
          }else{
                  ui= sine[k];}
```

```
            }
}
/*dio koji racuna IDFT*/
  if (smjer != FFT) {
    for (uint16_t i = 0; i < N; i++) {
      Re[i] /= N;
      Im[i] /= N;}
   }
}
```

Implementacija VP FIR filtra

```
#define L 17  // VP 17, broj uzoraka koeficijenata
q15_t prethodni[L];
/*koeficijenti VP FIR filtra*/
q15_t coeffVP[17]={ -99,  -53,   220,  761,  830,   -771,  -4305,  -8169,
                  23012, -8169, -4305, -771, 830, 761, 220, -53, -99 };
/*VP FIR filtar*/
void FIRVP(q15_t *ul,q15_t *izl, uint16_t N){
          q15_t pom=0;
          q31_t acc;//akumulator
            /*cirkularno polje napuniti nulama*/
            for(uint16_t k=0;k<L;k++){
                  prethodni[k]=0;
                  }
            for(uint16_t i=0;i<N;i++){
                    acc=0;
                    prethodni[pom]=ul[i];
                    pom=(pom+1)%L; // racunanje cirkularnog pomaka
                   for(uint16_t j=0;j<L;j++){
                       acc=acc+(q15_t)(prethodni[pom]* coeffVP[j] >> 15);
                       //akumuliranje umnoska stanja i koeficijenta
                       pom=(pom+1)%L;
                       }
              izl[i]=acc;
              //spremiti akumulirani podatak u izlazno polje FIR filtra
            }
}
```