Univerza v Ljubljani
Fakulteta za elektrotehniko

Marko Blažič

MERILNI SISTEM ZA FREKVENČNO ANALIZO VIBRACIJ

Magistrsko delo

Mentor: izr.prof. dr. Andrej Trost

Ljubljana, 2016

Zahvala

Zahvaljujem se mentorju izr. prof. dr. Andreju Trostu in dr. Martinu Furlanu za vso pomoč in nasvete pri nastajanju magistrskega dela. Zahvaljujem se tudi vsem bližnjim, ki so me podpirali.

Vsebina

Seznam slik

Seznam uporabljenih simbolov

AF Kriterij za določanje odstotka prekrivanja frekvenčnih spektrov (ang. Amplitude Flattness)

ASIC Namensko integrirano vezje (ang. Application Specific Integrrated Circuit)

AXI Vodilom, ki se uporablja za komunikacijo med IP komponentami (ang. Advanced eXtended Interface)

DFT Diskretna Fourierjeva transformacija

DMA Komponenta za direkten dostop do pomnilnika (ang. Direct memory access)

DNL Diferencialna nelinearnost (ang. Differential Non Linearity)

EMIO Razširjen multipleksiran vhod/izhod (ang. Multiplexed Input/Output)

FAT Datotečni sistem (ang. File Allocation Table)

FFT Hitri Fourierjev transform (ang. Fast Fourier Transform)

FPGA Programirljiva logična vrata (ang. Field Programmable Gate Array)

FTP Protokol za prenos datotek (ang. File Transfer Protocol)

GP Splošno namenski port (ang. General Purpose Port)

GPIO Vhodno izhodni port za splosno uporabo (ang. General Purpose Input Output)

GPL Licenca za prosto programje (ang. General Public Licence)

HF Visoko frekvenčno območje (ang. high frequency band)

HP Visoko zmogljiv port (ang. High Performance Port)

HTTP Komunikacijski protokol (ang. Hypertext Transfer Protocol)

I/O Vhod / izhod (ang. Input / Output)

IEPE Zvrst pospeškomerjev (ang. Integrated Electronic Piezoelectric)

INL Integralna nelinearnost (ang. Integral Non Linearity)

IP Intelektualna lastnina (ang. Intelectual Property)

IRQ Prekinitev (ang. interrupt reques)

LF Nizko frekvenčno območje (ang. low frequency band)

MF Srednje frekvenčno območje (ang. medium frequency band)

MIO Multipleksiran vhod/izhod (ang. Multiplexed Input/Output)

OC Korelacija prekrivanja - kriterij za določanje deleža prekrivanja pri oknenju (ang. Overlap Corelation)

P2P Vrednost od vrha do vrha (ang. Peak to Peak)

PF Ploskost moči - kriterij za določanje deleža prekrivanja pri oknenju (ang. Pover Flattness)

PL Programirljiva logika

PS Procesorski sistem

RAM Spomin z naključnim dostopom (ang. Random Access Memory)

RMS Efektivna vrednost (ang. RMS – Root Mean Square)

SDK Razvojno okolje za programsko opremo (ang. Software Development Kit)

SNR Razmerje signal šum (ang. Signal to Noise Ratio)

SOC Sistem na čipu (ang. System On Chip)

SPI Protokol za serijsko komunikacijo (angl. Serial Peripheral Interface Bus)

Povzetek

Meritve in analiza vibracij so pomembne za nadzor kakovosti proizvodnje elektromotorjev. Frekvenčna analiza vibracij nam omogoča natančno ugotavljanje morebitnih neustreznih delov in posledično neustreznih elektromotorjev.

V nalogi je predstavljen koncept merilnega sistema za vibracije z uporabo razvojne plošče Zedboard, na kateri teče operacijski sistem FreeRTOS. Predstavljena je obdelava signala v časovnem in frekvenčnem prostoru. Posebnost Zedboarda je v tem, da temelji na sistemu na integriranem vezju (ang. System On Chip) Xilinx Zynq. Zmogljiv procesor nam omogoča poganjanje operacijskega sistema, programirljiva logika pa omogoča izvajanje časovno potratnih opravil.

Zajem in obdelava signala je izvedena za en kanal. Predstavljena je frekvenčna analiza, uporaba okenskih funkcij z vplivi na rezultate. Prav tako je prikazan postopek povprečenja in prekrivanja frekvenčnih spektrov. Predstavljen je tudi izračun RMS (ang. Root Mean Square) vrednosti vibracij v posameznih frekvenčnih pasovih. Na koncu je primerjava meritev izdelanega merilnega sistema z profesionalnim merilnim sistemom LMS Scadas.

Ključne besede: sistem na čipu, FPGA, Xilinx Zynq, FreeRTOS, FFT, okenske funkcije, vzorčenje

Abstract

Vibration measurment and analysis is important for monitoring the quality of production of electromotors. With frequency analysis of vibrations we are able to to detect bad units and also determine the part of electromotor wich is the source of failure.

In this master thesis I presented the concept of measurment system based on Zedboard development board, with instaled FreeRTOS operating system. Also signal procesing in time and frequency domain is presented. Speciality of Zedboard is that it is based on Xilinx Zynq System On Chip (SoC). Powerfull dual core microprocessor is able to run operating system and at the same time programable logic allow us to run time consuming tasks.

Sampling and signal processing is done for one chanell at the moment. Frequency analysis is presented with windowing functions and their influence on results. Then avaraging and overlapping of frequency spectras is presented. Next RMS (Root Mean Square) calculation is used for calculating power in three separated frequency bands. At the end the comparison between made measuring system and professional measuring system LMS Scadas is done.

Key words: system on chip, FPGA, Xilinx Zynq, FreeRTOS, FFT, windowing functions, sampling

1 Uvod

Ljudje imamo sposobnost zaznavanja vibracij, težava pa nastane, ko želimo ovrednotiti vibracije v smislu frekvence in amplitude. To zahteva uporabo ustreznih senzorjev in inštrumentov. Poslužujemo se digitalne obdelave signala, saj želimo signal ovrednotiti s številnimi računsko zahtevnimi algoritmi.

Meritve vibracij se začnejo z natančnim zajemanjem vhodnega signala, običajno iz pospeškomerja. Nato sledi digitalna obdelava signala in izračun določenih cenilk. Razmisliti moramo o vzorčevalni frekvenci, s katero bomo vzorčili vhodni signal, frekvenčni resoluciji, načinu povprečenja frekvenčnih spektrov, o številu povprečenj ter izbiri okenske funkcije.

Za zajem signala smo uporabili analogno digitalni pretvornik Fresno proizvajalca Maxim. Zanj smo se odločili, ker ustreza zahtevi po 16-bitni pretvorbi, in ker omogoča priklop preko priključka PMOD (ang. Peripheral Module Interface). V primeru, da bi potrebovali več kanalov, bi lahko priključili več analogno digitalnih (AD) pretvornikov ali enega večkanalnega.

Obdelava signala je sestavljena na dva dela. Najprej se ovrednoti časovni signal in se določi vrednost RMS (ang. Root Mean Square), ter vrednost od vrha do vrha Vpp (ang. Peak to Peak). Nato se z uporabo FFT (ang. Fast Fourier Transformation) algoritma ob uporabi oknenja izračuna frekvenčni spekter signala. Pri računanju moči iz spektra moramo biti pozorni na vpliv okenske funkcije in to pravilno kompenzirati. Uporabimo prekrivanje in povprečenje frekvenčnih spektrov za boljše rezultate z manj šuma.

Za komunikacijo z zunanjim svetom skrbi operacijski sitem FreeRTOS, ki poleg zajemanja in obdelave vhodnega signala poganja FTP (ang. File Transfer Protocol) strežnik. Aplikacija je zasnovana tako, da čaka na datoteko, ki vsebuje potrebne parametre, ter hkrati proži meritev. Ko se meritev izvede, se rezultati shranijo v ustrezne datoteke, dostopne preko FTP povezave.

Za izvedbo koncepta merilnega sistema za meritve vibracij smo uporabili razvojno ploščo Zedboard, ki temelji na SoC (ang. System On Chip) Xilinx Zynq-7000. Posebnost tega mikrokrmilnika je, da je na isti rezini skupaj realiziran procesor in programirljiva logika FPGA (ang. Field Programmable Gate Array). FPGA vezja nam zaradi svoje zgradbe omogočajo paralelno računanje in so zato zelo primerna za izvajanje določenih nalog, kot sta na primer filtriranje in izračun FFT algoritma. Omogočajo, da določeno funkcionalnost prenesemo iz procesorskega sistema v programirljivo logiko in se na ta način znebimo dodatne zunanje periferije, ali pa razbremenimo procesor.

Namen naloge je bil poiskati alternativo številnim obstoječim merilnim sistemom, ki omogočajo meritve vibracij. Pogosto se uporablja oprema podjetja National Instruments, ki omogoča modularno zasnovo sistema. Sistem je sestavljen iz NI cDAQ-9184 ohišja [9], ki omogoča povezavo preko etherneta. Vanj pa lahko priključimo različne merilne kartice, na primer za vibracije je namenjena NI 9234 [10], ki je vstavljena prva z leve. Za uporabo tega merilnega sistema potrebujemo še osebni računalnik in programsko opremo, s katero zajemamo podatke in jih obdelamo.

Slika 1.1: Primer modularnega merilnega sistema podjetja NI [9]

2 Teorija

2.1 Meritev in analiza

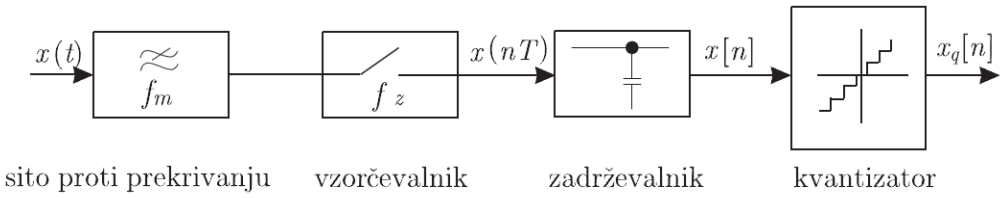
Predmet naloge je zajem signala vibracij, njegova analiza ter komunikacija z zunanjim svetom. Signal lahko analiziramo v časovnem ali frekvenčnem prostoru. V časovnem prostoru lahko izračunamo vrednosti P2P in RMS. Če želimo ovrednotiti vrednosti posameznih harmonskih komponent signala, moramo izračunati frekvenčni spekter signala z uporabo FFT algoritma. To pomeni, da signal transformiramo iz časovnega v frekvenčni prostor. Iz frekvenčnega spektra vibracij lahko prepoznamo prevladujoče frekvenčne komponente, ki so direktno povezane s posameznim delom stroja, iz katerega izhajajo.

Vrednost RMS je najbolj pomembno merilo signala vibracij, saj odraža vrednost signala v določenem časovnem ali frekvenčnem oknu in je ta vrednost direktno povezana z energijsko vsebnostjo signala ter posledično učinkom vibracij na strukturo.

2.2 Analogno digitalna pretvorba signala

Signal vibracij je v osnovi analogen časovno zvezen signal. Da bi tak signal lahko obdelali v digitalnem sistemu, ga moramo najprej pretvoriti v diskretni signal po času in po amplitudi. To običajno naredimo z vzorčenjem signala v enakomernih časovnih intervalih ter s kvantizacijo vzorcev. Govorimo o analogno-digitalni pretvorbi [2].

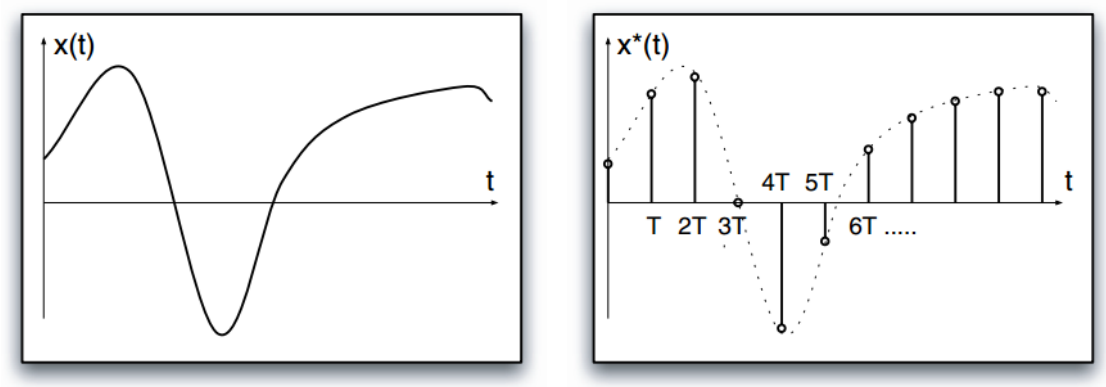
Slika 2.1 predstavlja AD pretvornik kot štiri zaporedne podsisteme: analogno sito proti prekrivanju, vzorčevalnik, zadrževalnik in kvantizator. Najprej si poglejmo, kako samo vzorčenje vpliva na signal.



Slika 2.1: Blokovni diagram osnovnih funkcij AD pretvornika [2]

Z idealnim vzorčenjem bi dobili vlak delta impulzov, katerih amplituda bi ustrezala signalu ob vzorčevalnih trenutkih. Pri vzorčenju v enakomernih časovnih intervalih s periodo T dobimo izhod iz vzorčevalnika [1]

$$\mathbf{x^*(t) = x^*(nT) = \sum x(t)\delta(t - nT)} \qquad \mathbf{(2.1)}$$



Slika 2.2: Časovno zvezen signal levo ter vzorčen signal desno [1]

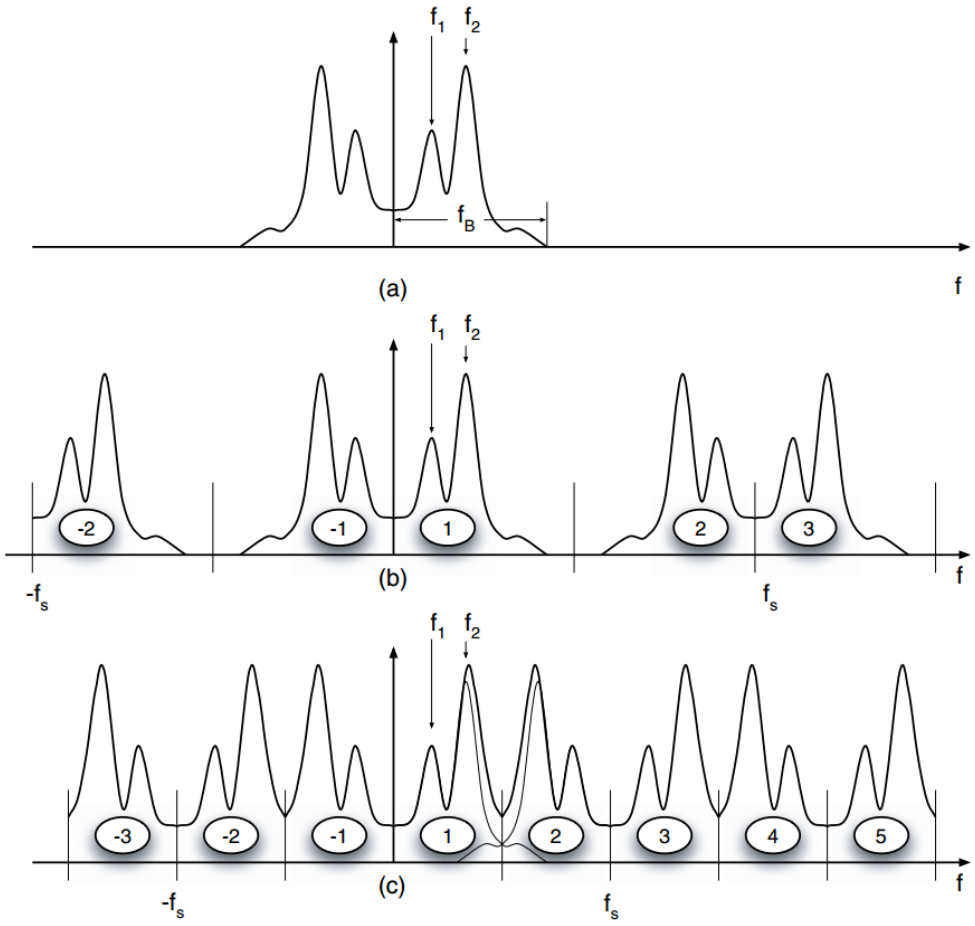
Enačba (2.1) prikazuje nelinearnost operacije vzorčenja. Spekter vhodnega signala je frekvenčno omejen, po množenju z vlakom delta impulzov pa je preslikan okoli večkratnikov vzorčevalne frekvence. Laplaceova transformacija neskončnih delta impulzov je:

$$\mathbf{L\left[\sum_{-\infty}^{\infty} \delta(t - nT)\right] = \sum_{-\infty}^{\infty} e^{-nsT}} \qquad \mathbf{(2.2)}$$

Iz enačbe (2.1) in enačbe (2.2) lahko zapišemo

$$\mathbf{L}[\mathbf{x}^*(\mathbf{nT})] = \sum_{-\infty}^{\infty}(\mathbf{X}(\mathbf{s} - \mathbf{jn}\boldsymbol{\omega}_\mathbf{s})) = \sum_{-\infty}^{\infty}\mathbf{x}(\mathbf{nT})\mathbf{e}^{-\mathbf{nsT}} \qquad \mathbf{(2.3)}$$

Enačba (2.3) prikazuje, da je spekter vhodnega časovno zveznega signala preslikan okoli večkratnikov vzorčevalne frekvence, ki so prestavljeni po frekvenčni osi za: $n * fs,\ n = 0, \pm1, \pm2, \ldots$



Slika 2.3: a) Dvostranski spekter časovno zveznega signala, b) spekter vzorčenega signala z fs/2 > fb, c) spekter vzorčenega signala s fs/2 < fb [1]

V primeru, da je naša vzorčevalna frekvenca fs, ter da je frekvenca merjenega signala višja kot Nyquistova frekvenca fs/2, se bo zgodilo prekrivanje replik spektra. To pomeni, da se bodo replike spektra zrcalile v osnovni spekter preko fs/2. V večini

aplikacij, kot tudi v tej, zrcaljenje ni zaželeno. Da preprečimo zrcaljenje, uporabimo nizko prepustni ("anti-aliasing") filter z mejno frekvenco pod Nyquistovo frekvenco. [1]. Predpostavimo, da je na sliki 2.3a predstavljen dvostranski spekter vhodnega signala, ki ima pasovno širino do frekvence fb. Nato je na sliki 2.3b predstavljen spekter vzorčenega signala, ki je vzorčen z fs > 2*fb. Iz slike je razvidno, da replike osnovnega spektra ne interferirajo med seboj. To nam omogoča, da lahko s filtriranjem iz vzorčenega signala rekonstruiramo originalni signal.

Slika 2.3c nam prikazuje, kaj se zgodi, če vzorčimo vhodni signal s prenizko vzorčevalno frekvenco. Replike originalnega spektra deloma interferirajo med seboj. Posledično iz vzorčenega signala ne moremo ovrednotiti prvotnih lastnosti vhodnega časovnega signala.

Teorem o vzorčenju. Signal x(t) je mogoče pri enakomernem vzorčenju popolnoma rekonstruirati iz vzorcev signala x[n] le, kadar je signal frekvenčno omejen in je frekvenca vzorčenja večja od dvakratne najvišje frekvence prisotne v signalu. Signal rekonstruiramo s pomočjo idealnega nizkega sita z mejno frekvenco, enako polovici vzorčne frekvence. Zavedati se moramo, da idealno sito ni izvedljivo, zato predstavlja teorem o vzorčenju le spodnjo teoretično mejo. [2]

2.3 Amplitudna kvantizacija

Ena od karakteristik AD pretvornika je njegova resolucija, ki nam določa, med koliko različnimi nivoji napetosti AD pretvornik razlikuje. To pomeni, da bo 16-bitni AD pretvornik razlikoval med 2^{16} = 65536 različnimi napetostnimi nivoji in bo vsak nivo predstavljen z določeno kodo. Za idealni AD pretvornik velja, da je odvisnost izhodne kode od vhodnega analognega signala linearna. To pomeni, da bo vsaka izhodna koda predstavljala enako območje vhodne napetosti.

Zaradi anomalij pri izdelavi realnih AD pretvornikov, se pojavijo odstopanja od idealne vrednosti posameznega napetostnega koraka. To imenujemo Differential Non-Linearity (DNL). Največje odstopanje, ki je posledica akumuliranih posameznih odstopanj napetostnega koraka, imenujemo Integral Non-Linearity (INL). Opisane nelinearnosti, skupaj z ostalimi, zmanjšujejo efektivno resolucijo AD pretvornika. [1]

Z AD pretvornikom merimo vhodni napetostni signal, predstavljen v voltih, in ga pretvorimo v binaren zapis z določenim številom bitov. V primeru, da hočemo naprej operirati z vrednostmi, ki predstavljajo napetost, moramo binarne podatke AD pretvornika pretvoriti v vrednost, ki predstavlja napetost. Podatke v celoštevilskem zapisu (integer) pretvorimo v zapis s plavajočo vejico (floating point) z množenjem z napetostjo najmanj pomembnega bita:

$$U_{LSB} = \frac{U_{min} - U_{max}}{2^n} \tag{2.4}$$

2.4 Fourierjeva transformacija

Spekter vzorčenega signala izračunamo z uporabo Laplaceove transformacije:

$$\mathbf{L}[\mathbf{x}^*(\mathbf{nT})] = \sum_{-\infty}^{\infty} \mathbf{x}(\mathbf{nT})\mathbf{e}^{-\mathbf{nsT}} \tag{2.5}$$

Oziroma z uporabo ekvivalentne Fourierjeve trnsformacije:

$$\mathbf{F}[\mathbf{x}^*(\mathbf{nT})] = \mathbf{X}^*(\mathbf{j\omega}) = \sum_{-\infty}^{\infty} \mathbf{x}(\mathbf{nT})\mathbf{e}^{-\mathbf{j\omega nT}} \tag{2.6}$$

Lastnost Fourierjeve transformacije je, da zahteva neskončno število vzorcev, ki v praksi niso na voljo, saj imamo običajno na voljo določeno število vzorcev N. Aproksimacija zgornje enačbe (2.6) je definirana kot Diskretna Fourierjeva Transformacija (DFT). DFT predpostavi, da je vhodni niz periodičen, x[(i+kN)T] = x(iT) za 0< i < (N-1) in za vse celoštevilske k-je. Tako postane vhodni niz periodičen z periodo N*T. Spekter je diskreten, sestavljen iz N komponent, ki se nahajajo na f_k = k/[T/(N-1)] za 0< k< (N-1) . Rezultat DFT je:

$$\mathbf{X(k)} = \frac{\mathbf{1}}{\boldsymbol{N}} \sum_{\mathbf{n=0}}^{\mathbf{N-1}} \mathbf{x(nT)} \mathbf{e}^{-\frac{\mathbf{j2\pi kn}}{\mathbf{N-1}}}, \qquad \boldsymbol{k = 0, \ldots, N-1} \tag{2.7}$$

Frekvenčne komponente X[k] periodičnega signala x[n] so v splošnem kompleksna števila, zato izračunamo amplitudo in fazo z naslednjima izrazoma [1].

$$|\mathbf{X(f_k)}| = \sqrt{\mathbf{Real}[\,\mathbf{X(f_k)}\,]^2 + \mathbf{Im}[\,\mathbf{X(f_k)}\,]^2} \tag{2.8}$$

$$\mathbf{Ph}\{\mathbf{X(f_k)}\} = \mathbf{tan^{-1}}[\frac{\mathbf{Im}[\,\mathbf{X(f_k)}\,]}{\mathbf{Real}[\,\mathbf{X(f_k)}\,]}] \tag{2.9}$$

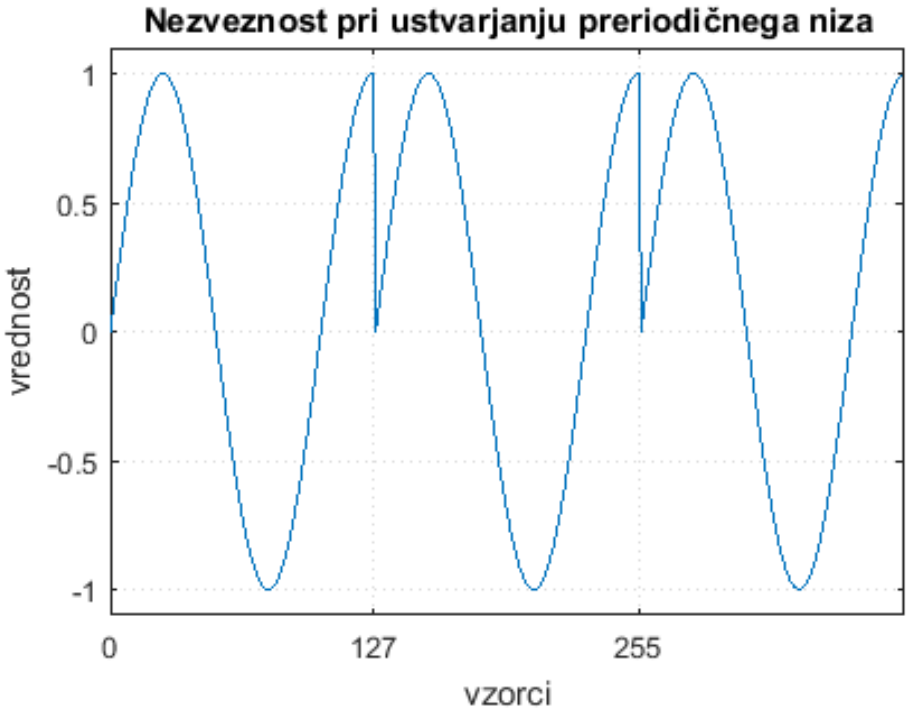
Izračun DFT po enačbi (2.7) je računsko zahteven, zato postane izračun algoritma za dolge nize vzorcev počasen. V praksi se zato uporablja FFT, ki bistveno zmanjša število potrebnih operacij. FFT je le skupno ime za različne algoritme, ki pohitrijo izračun DFT. Za prvi učinkovit algoritem lahko štejemo postopek Cooley-Turkey, ki se imenuje po svojih avtorjih. Število operacij se zmanjša iz N^2 na $N * log_2(N)$. Na primer, če imamo število vzorcev 8192, zmanjšamo število operacij za 630-krat. FFT algoritem je najučinkovitejši, ko je vhodno število vzorcev potenca števila 2 [2].

2.5 Oknenje

Oknenje je pomembno pri izračunu DFT oziroma FFT algoritma, saj ta predpostavlja, da je vhodna sekvenca periodična. Večina realnih signalov ni periodičnih. Z oknenjem dosežemo, da je vrednost niza vzorcev na začetku in na koncu enaka nič, in tako dosežemo ujemanje vzorcev z začetka in konca intervala. Na ta način odpravimo problem nezveznosti, ki se pojavi pri ustvarjanju periodičnega niza.

Slika 2.4 prikazuje niz 128 vzorcev sinusnega signala, ki je trikrat ponovljen. Niz vzorcev vsebuje 5/4 periode sinusnega signala. Na sliki je lepo vidno, da N-periodična transformacija privede do nezveznosti pri vzorcih 127, 255 in tako naprej.

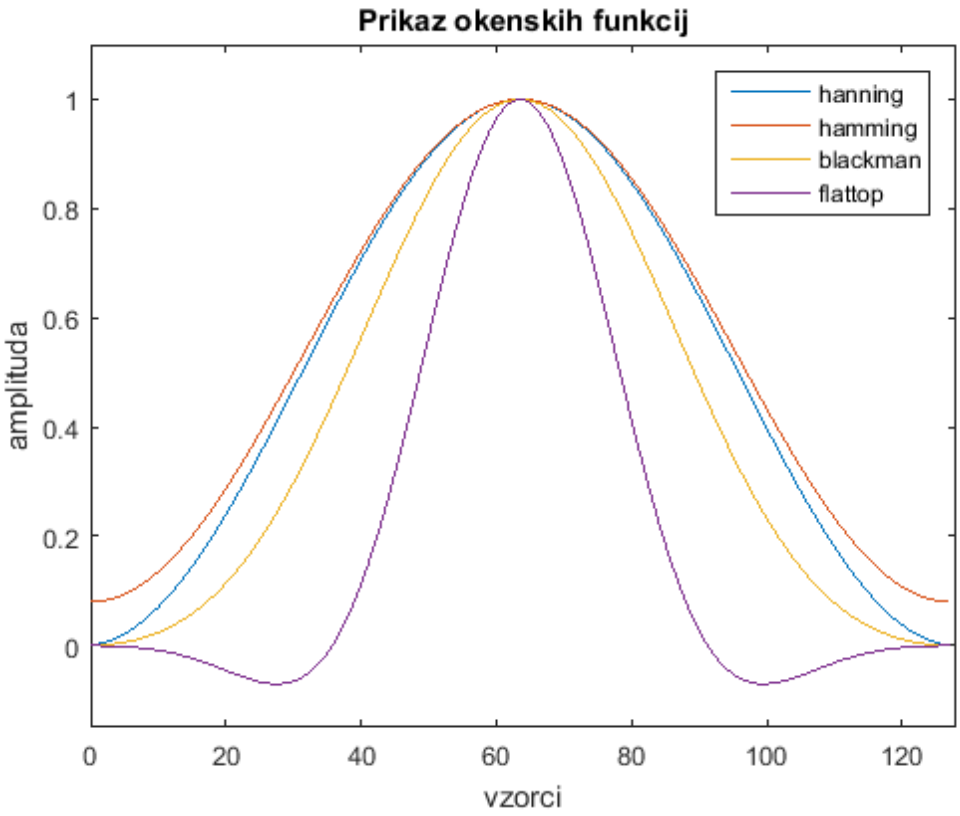
Rezultat tega je razširjen spekter, vsebujoč dodatne frekvenčne komponente, ki jih originalni sinusni signal ne vsebuje.



Slika 2.4: Prikaz nezveznosti pri ustvarjanju periodičnega niza

Oknenje opišemo kot množenje originalnega vektorja signala z okensko funkcijo W(k), kot to opisuje enačba (2.10) [1]. Slika 2.5 prikazuje nekaj najbolj pogostih okenskih funkcij.

$$\boldsymbol{x_w = x(kT) * W(k)} \qquad \textbf{(2.10)}$$



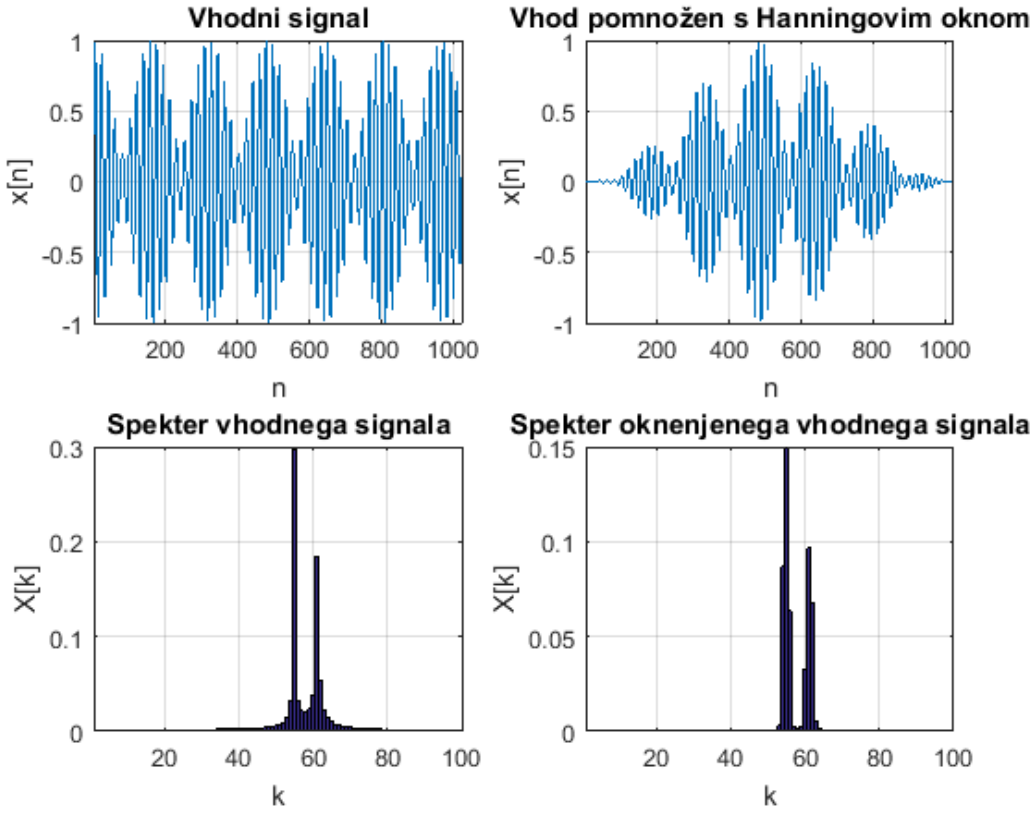
Slika 2.5: Prikaz nekaterih najbolj pogostih okenskih funkcij

Slika 2.6 prikazuje vpliv oknenja na vhodni signal, ki je sestavljen iz dveh sinusnih signalov. V idealnem primeru bi spekter vseboval dve izraziti komponenti, ostale pa bi bile enake 0. Zaradi nezveznosti pri transformaciji v periodičen niz, se pojavi spektralno puščanje, kar lahko vidimo na sliki 2.6. Ta pojav omilimo z oknenjem, pri tem pa opazimo, da amplituda frekvenčnih komponent pade za polovico. Najpomembnejše pa je, da se ohrani razmerje amplitud frekvenčnih komponent [1].

Vpliv oknenja na energijsko vsebnost ali amplitudo vhodnega signala lahko kompenziramo z naslednjimi korekcijskimi faktorji:

Korekcijski faktorji za Hanningovo okno:

- Amplitudna korekcija: signal po oknenju pomnožimo z 2
- Energijska korekcija: signal po oknenju pomnožimo z 1,63



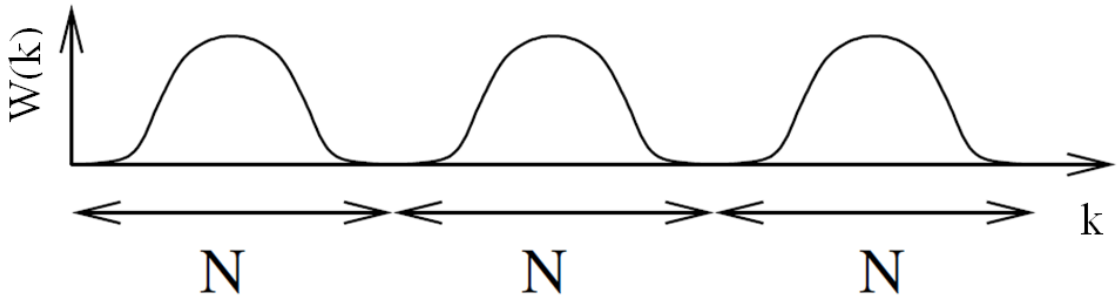
Slika 2.6: Vpliv oknenja na časovni signal ter spekter signala

2.6 Povprečenje in prekrivanje spektrov

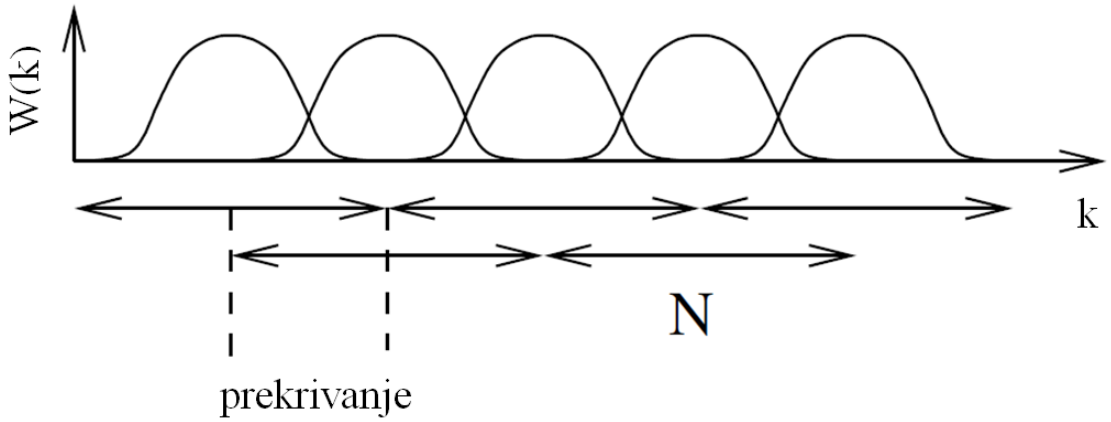
Če izračunamo spekter signala na način, kot smo ga do sedaj opisali (množenje enega segmenta časovnih vzorcev z izbrano okensko funkcijo, izračun DFT algoritma in skaliranje rezultatov), bomo običajno ugotovili, da rezultat vsebuje precej šuma.

Običajno je priporočljivo, da spektre povprečimo z M spektri, in s tem zmanjšamo standardno deviacijo za faktor $1/\sqrt{M}$. Ob tem moramo zagotoviti, da je signal v tem časovnem segmentu stacionaren. Povprečiti moramo močnostni spekter (ang. Power Spectrum - PS) in ne linearnega spektra (ang. linear spectrum - LS). Ta metoda je bolje poznana pod imenom "Welch's overlapped Segmented average".

Slika 2.7 prikazuje primer, ko imamo dolg signal preprosto razdeljen na posamezne neprekrivajoče se odseke, ki so nato procesirani z DFT ob uporabi okenske funkcije. Ker je lastnost okenskih funkcij, da so na robovih enake ali skoraj enake nič, je del signala v nadaljnji analizi prezrt. To je še posebej problematično, ko želimo iz časovno čim krajšega signala pridobiti čim več informacij. Rešitev je prekrivanje segmentov, kar prikazuje slika 2.8.

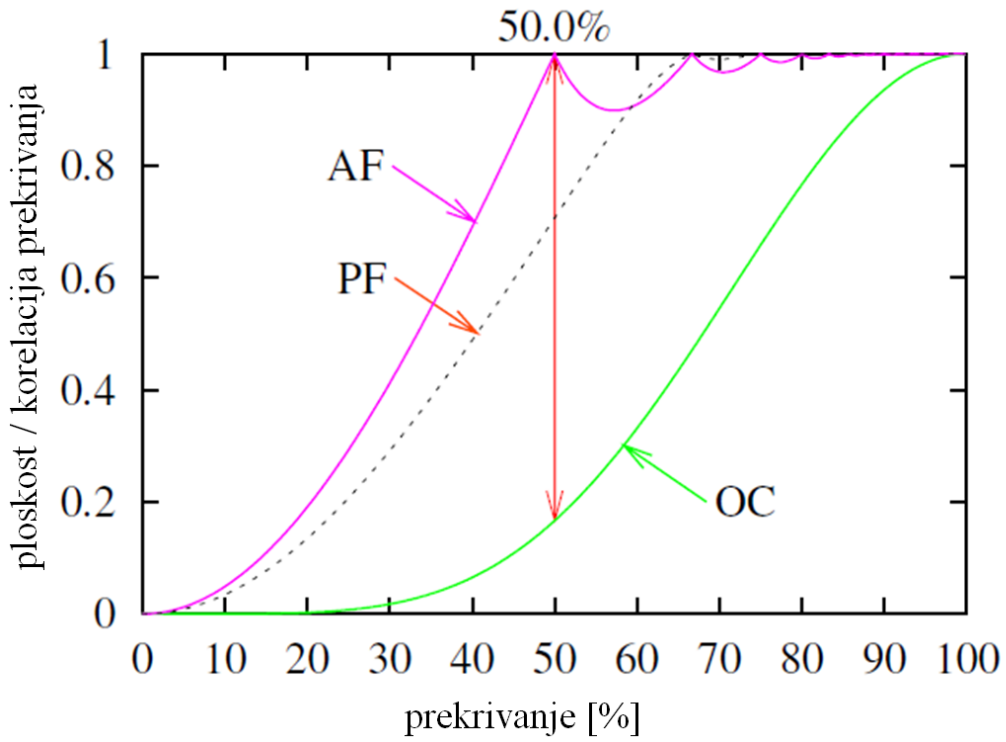


Slika 2.7: Signal razdeljen na neprekrivajoče se intervale z uporabo oknenja [7]



Slika 2.8: Signal je razdeljen na intervale, ki se prekrivajo [7]

Pojavi se vprašanje, kolikšen del naj se intervali prekrivajo med seboj. To je predvsem odvisno od okenske funkcije ter od naših zahtev. Za okna, ki so relativno široka v časovnem prostoru, kot je Hanningovo okno, je 50 % običajna vrednost prekrivanja. Za ožja okna, kot so flat-top okna, je priporočena višja vrednost prekrivanja, do 84 %. Na izbiro optimalne vrednosti prekrivanja vplivajo kriteriji, ki opisujejo razmerje med najmanjšim in največjim vplivom posameznega vzorca na amplitudni ali močnostni spekter, ter računska zahtevnost. Slika 2.9 prikazuje omenjene kriterije [7].

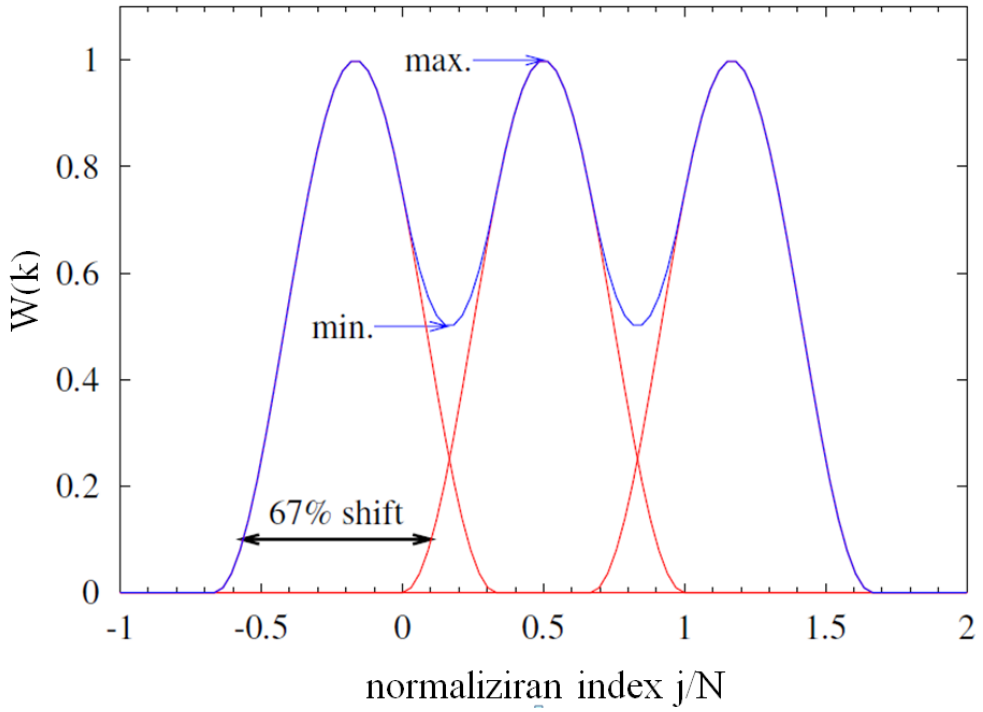


Slika 2.9: Karakteristika prekrivanja Hanningovega okna, [7]

Amplitudna ploskost: (ang. Amplitude Flatness - AF). Ko uporabljamo prekrivanje, so določeni vzorci vzorčeni večkrat z različno utežjo. Da določimo celotno utež posameznega vzorca, lahko seštejemo vse vrednosti okenske funkcije, s katerimi pomnožimo obravnavan vzorec. AF je razmerje med minimalno in maksimalno celotno utežjo, ki jo ima posamezen vzorec. Slika 2.10 nam prikazuje primer, kjer je prekrivanje 33 % in je manjše od optimalnega 50 % .

Želimo, da so vsi vzorci enako obravnavani oziroma, da imajo enako utež, torej da je AF enak 1. Za Hanningovo okno postane krivulja seštetih zamaknjenih okenskih funkcij popolnoma ravna pri 50 % prekrivanju.

Ploskost moči : (ang. Power Flatness - PF). Zgoraj opisano linearno seštevanje okenskih funkcij – ki določa kriterij AF – je primerno za sinusne signale. Nekoherentne signale, kot je šum, je potrebno najprej kvadrirati in nato sešteti. To velja tudi za okensko funkcijo, iz tega sledi kriterij PF.



Slika 2.10: Razmerje AF Hanningovega okna pri prekrivanju 33 % [7]

Korelacija prekrivanja: (ang. Overlap Correlation - OC). Če je prekrivanje preveliko, postane korelacija med sosednjimi okvirji velika tudi v primeru, da je signal naključen. Velika korelacija pa pomeni neučinkovito porabo računske moči. Korelacija prekrivanja za vrednost prekrivanja **r** je izračunana z enačbo 2.11:

$$\mathrm{OC(r)} = \frac{\sum_{j=0}^{rN-1} w_j w_{j+(1-r)N}}{\sum_{j=0}^{N-1} w_j^2} \qquad \textbf{(2.11)}$$

Kjer je Hanningovo okno definirano kot:

$$w_j = \frac{1}{2}\left[1 - cos\left(\frac{2\pi \cdot j}{N}\right)\right], \qquad j = 0, \ldots N\text{-}1 \qquad \textbf{(2.12)}$$

Ko se odločamo o vrednosti prekrivanja, želimo, da sta pogoja AF in PF čim večja, z željo, da so vsi vhodni podatki enako uteženi. Po drugi strani pa želimo čim manjšo korelacijo prekrivanja, da ne trošimo računske moči. Definiramo lahko priporočeno vrednost prekrivanja ROV (ang. Recomended Overlap), kot vrednost

prekrivanja, kjer razlika med kriterijem AF in OC doseže maksimum. To prikazuje slika 2.9.

2.7 Odstranjevanje enosmerne komponente

Običajno ima vzorčen signal – ki ga želimo frekvenčno ovrednotiti – srednjo vrednost različno od nič. Srednja vrednost ustreza enosmerni komponenti signala in predstavlja prvo frekvenčno komponento v izračunanem spektru. Če uporabimo okensko funkcijo, se bo ta enosmerna komponenta razlezla v sosednje frekvenčne komponente in bo lahko prekrila nizkofrekvenčne komponente vhodnega signala.

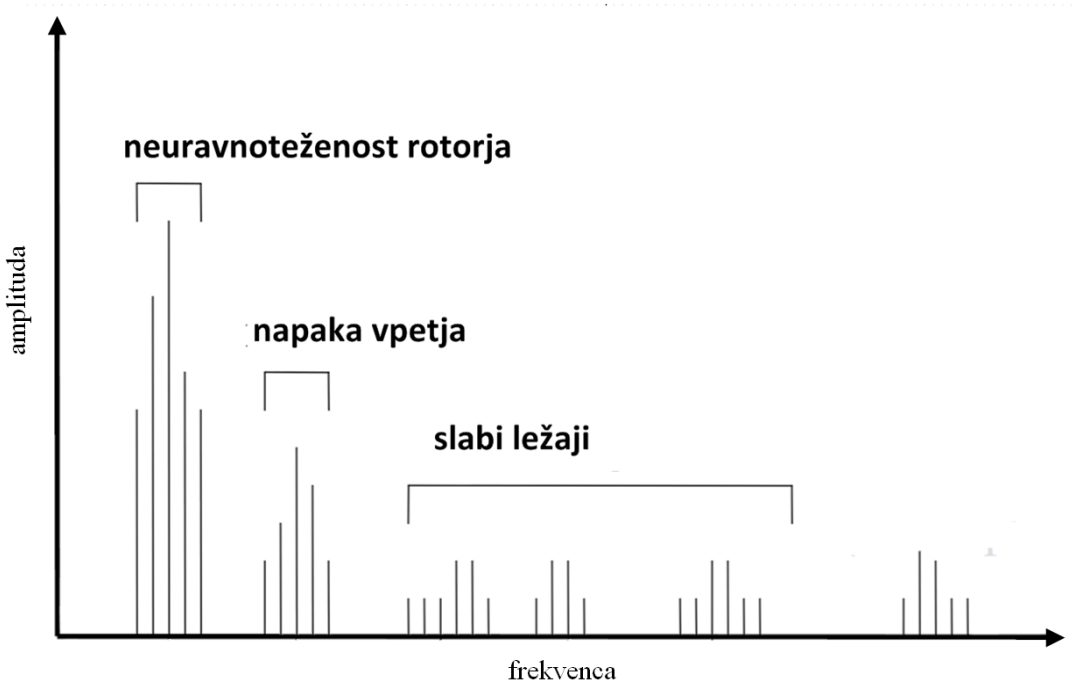
Za določanje srednje vrednosti vhodnega signala poznamo bolj primerne metode (povprečenje, nizkoprepustno filtriranje), zato vrednotenje enosmerne komponente iz spektra ni v našem interesu. Iz napisanega sledi, da je enosmerno komponento priporočljivo odstraniti iz vhodnega signala [7].

2.8 Izračun moči iz amplitudnega spektra

Za izračun moči iz frekvenčnega spektra uporabimo Parsevalov teorem. Ta razlaga, da je povprečna moč signala enaka vsoti povprečnih moči posameznih frekvenčnih komponent. Enačba (2.13) predstavlja Parsevalov teorem: [2]

$$\frac{1}{N}\sum_{n=0}^{N-1}|x[n]|^2 = \sum_{k=0}^{N-1}|X[k]|^2 \qquad \textbf{(2.13)}$$

Računanje moči iz močnostnega spektra je uporabno pri določanju porazdelitve moči v odvisnosti od frekvence. Na ta način lahko določimo moč vhodnega signala v določenem frekvenčnem območju in to lahko uporabimo kot cenilko oziroma kazalec za določeno napako merjenca (električnega stroja) [8]. Slika 2.11 prikazuje primer, kako lahko povežemo povečano moč v določenem frekvenčnem pasu z napako na merjencu (npr. elektromotorju).



Slika 2.11: Porazdelitev moči v odvisnosti od frekvence [8]

Tako lahko po enačbi 2.14 izračunamo RMS vrednost signala iz frekvenčnih komponent, kjer so |X[k]| absolutne vrednosti posameznih frekvenčnih komponent, izračunanih z FFT algoritmom.

$$\boldsymbol{Vrms} = \sqrt{\sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} |\boldsymbol{X[k]}|^{\boldsymbol{2}}} \tag{2.14}$$

V primeru, da delamo s polovičnim spektrom, pa uporabimo enačbo 2.15.

$$\boldsymbol{Vrms} = \sqrt{|\boldsymbol{X[0]}|^{\boldsymbol{2}} + \boldsymbol{2} * \sum_{\boldsymbol{k=1}}^{\left(\frac{\boldsymbol{N}}{\boldsymbol{2}}\right)\boldsymbol{-1}} |\boldsymbol{X[k]}|^{\boldsymbol{2}}} \tag{2.15}$$

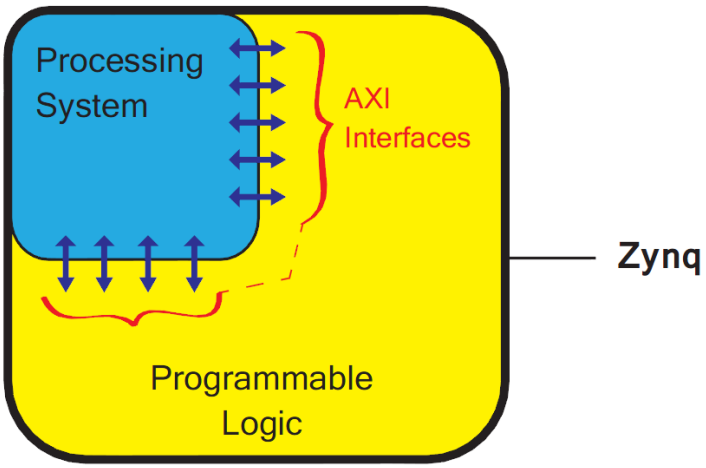
3 Strojna oprema

Za izdelavo merilnega sistema smo želeli uporabiti zmogljivo platformo, ki bi pozneje omogočala enostavno nadgradnjo sistema. Tukaj se je ponudila možnost izvedbe projekta z razvojno ploščo Zedboard, ki temelji na Xilinx Zynq-7000 SoC (ang. System On Chip). Posebnost Zynq SoC je v tem, da je na isti rezini skupaj realiziran dvojedrni procesor ARM Cortex-A9 s programabilno logiko FPGA (ang. Field Programmable Gate Array). ARM Cortex-A9 je zmogljiv procesor, zmožen poganjati operacijske sisteme, kot sta FreeRTOS ali Linux, programabilna logika pa je osnovana na FPGA serije 7, proizvajalca Xilinx. Procesor je povezan z FPGA delom preko vodil industrijskega standarda AXI, ki zagotavljajo hiter prenos podatkov in nizke zakasnitve. Uporaba SoC nam omogoča, da procesor in FPGA uporabljamo za opravila, ki jih posamezen del najbolj učinkovito opravlja, prihranimo pa pri načrtovanju tiskanega vezja s fizično ločenim procesorjem in FPGA-jem.

Koncept SoC je na trgu prisoten že nekaj časa. Pomeni, da na enem silicijevem čipu implementiramo funkcionalnost celotnega sistema, namesto da bi uporabili različne čipe za posamezno funkcionalnost. V preteklosti se je izraz SoC uporabljal predvsem v povezavi z ASIC (ang. Application Specific Integrated Circuit) čipi, ki so bili načrtovani specifično za posamezno aplikacijo, ki so lahko vključevala različne podsestave. Prednosti SoC so cena, višje hitrosti, nižja poraba, manjše dimenzije in večja zanesljivost. Glavna slabosti ASIC SoC pristopa pa so čas in cena razvoja ter neprilagodljivost sistema. Tak pristop je primeren predvsem za veliko serijske izdelke, ki ne zahtevajo nobenih popravkov v življenjski dobi.

Iz napisanega je jasno, da si želimo bolj prilagodljivo rešitev. Prva izbira je dolgo bil FPGA, v katerem lahko prav tako realiziramo procesor, če je to potrebno. Zynq pa nam ponuja še več. Imamo zmogljiv procesor in polje FPGA, ki ga lahko kadarkoli preprogramiramo in prilagodimo novim zahtevam. Ta kombinacija je uporabna predvsem takrat, ko želimo procesorsko jedro razbremeniti paralelnih ter

drugih težjih nalog v realnem času. FPGA je idealen za obdelavo toka (ang. streama) podatkov, kot je na primer digitalno filtriranje ali izračun FFT algoritma. Na drugi strani se na procesorju izvaja programska koda ter po potrebi operacijski sistem. To pomeni, da je funkcionalnost celotne naprave razdeljena med procesorjem (PS) in programabilno logiko (PL), kot prikazuje slika 3.1. [2]

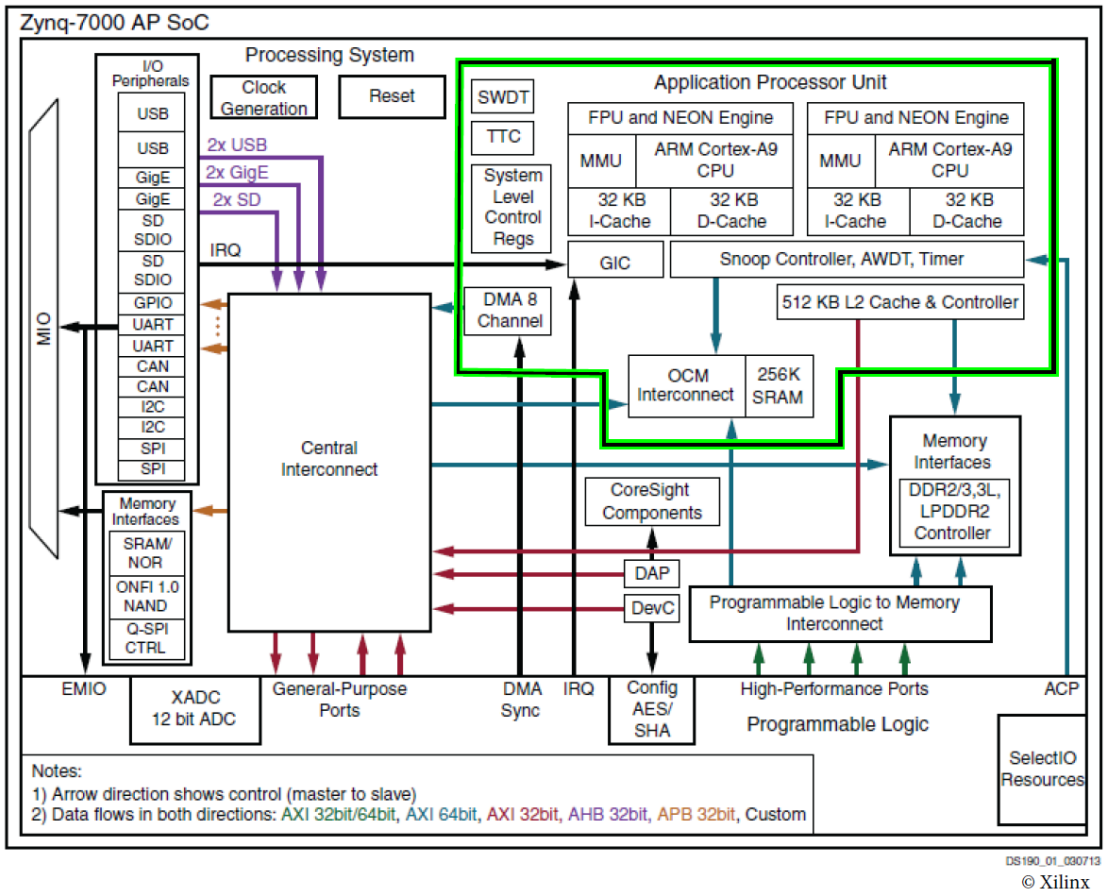


Slika 3.1: Razdelitev na procesorski sitem in programabilno logiko [3]

3.1 Zynq SOC

Celotna družina Zynq naprav ima enako arhitekturo, ki jo lahko razdelimo na procesorski del PS (ang. Processing System) in programabilno logiko PL (ang. Programable Logic).

Jedro PS je dvojedrni ARM Cortex-A9 procesor. To je tako imenovana »hard« IP komponenta, realizirana je kot optimiziran del silicija, v primerjavi s »soft« IP procesorjem, kot je Xilinx MicroBlaze, ki je realiziran kot programiran del FPGA programirljive logike. Implementacija »soft« procesorja je enaka implementaciji katere druge IP komponente. Slika 3.2 prikazuje podrobnejšo sestavo PS dela Zynq SoC. Povezava z zunanjostjo poteka preko MIO (ang. Multiplexed Input/Output), ki zagotavlja 54 fleksibilnih priključkov. V primeru, da jih potrebujemo več, lahko do zunanjosti pridemo tudi preko EMIO (ang. Extended MIO), ki poteka preko PL. Slika 3.2 prikazuje, da imamo na razpolago I/O (ang. Input/Output) s standardnimi komunikacijskimi vmesniki USB(x2), SPI(x2), I2C(x2), UART(x2), CAN(x2), SD(x2), GigE(x2), GPIO(4x32bit) .



Slika 3.2: Prikaz zgradbe procesorskega sistema [3]

Drugi del Zynq arhitekture pa je PL. Ta je osnovan na FPGA družini Artix-7 ali Kintex-7 proizvajalca Xilinx. Artix-7 se uporablja za manj zmogljive naprave, Kintex-7 pa za zmogljivejše.

3.2 Vodila AXI

Kot že omenjeno, je glavna prednost Zynq arhitekture ravno v povezavi procesorskega jedra z programabilno logiko na istem kosu silicija in zmožnost, da ju uporabljamo kot enovit zaključen sistem. To povezavo omogočajo AXI vodila, ki so most med PS in PL.

AXI (ang. Advanced eXtensible Interface) – trenutna verzija je AXI4 – je del ARM AMBA 3.0 odprtega standarda, ki ga je razvilo podjetje ARM za uporabo znotraj mikroprocesorjev. Sčasoma je to postal uveljavljen standard za komunikacijo

znotraj čipov ter med komponentami SoC. Tu imamo v mislih povezave med več procesorji ali IP komponentami v programirljivi logiki [3, stran 31-33].

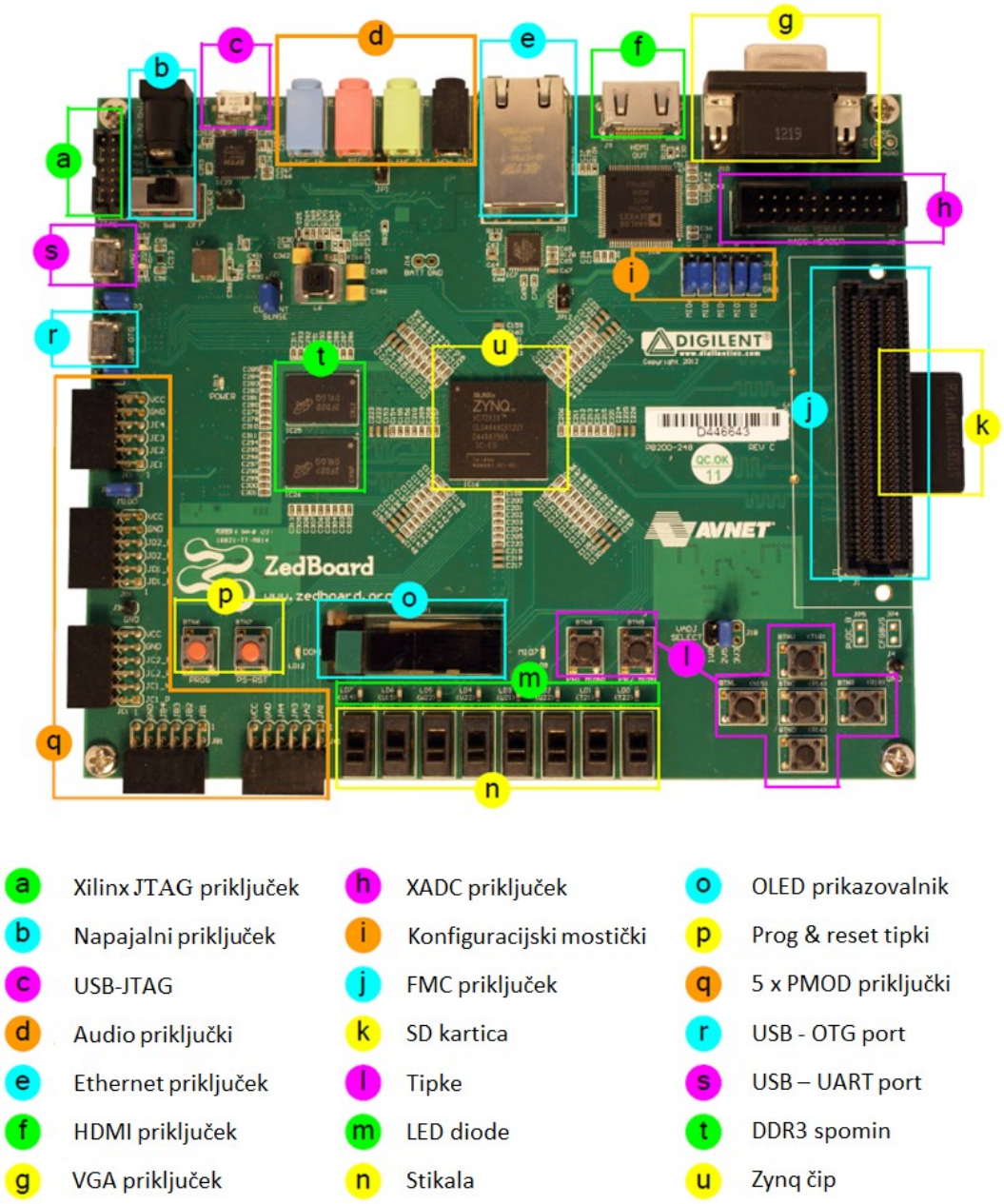
Obstajajo tri različice AXI vodil. Izbira je odvisna od tega, kakšne so zahtevane lastnosti povezave:

- **AXI4** – Uporablja se za pomnilniško preslikan (ang. memory mapped) prenos podatkov z najvišjo hitrostjo. Najprej je poslan naslov, nato mu sledi niz podatkov. Dolžina niza podatkov je odvisna od verzije AXI protokola. Za verzijo 3 je omejitev 16 podatkovnih besed na poslan naslov, za verzijo 4 pa 256 podatkovnih besed na poslan naslov.
- **AXI4 -Lite** – Je poenostavljeno vodilo AXI4. Prav tako je pomnilniško preslikan. Ne omogoča prenosa niza podatkov, ampak le ene podatkovne besede za vsak naslov.
- **AXI4 Stream** – Primeren za hiter prenos podatkov. Omogoča pošiljanje neomejenega števila golih podatkov. Naslova ne pošiljamo, primerno je za direkten tok podatkov med izvorom in ponorom.

Pomnilniško preslikan protokol pomeni, da je naslov (pisanja ali branja) poslan skupaj s podatki, ki jih pošlje gospodar (ang. master) vodila. Ta naslov ustreza nekemu naslovu v naslovnem prostoru sistema. V primeru vodila AXI4-Lite, ki ga podpira prenos enega podatka na posamezno transakcijo, je podatek prebran ali zapisan na naslov, ki je poslan skupaj s podatkom. V primeru AXI4 vodila; ko pošljemo en naslov in več podatkov, mora podrejena naprava (ang. slave) sama izračunati naslove podatkov, ki sledijo.

3.3 Zedboard

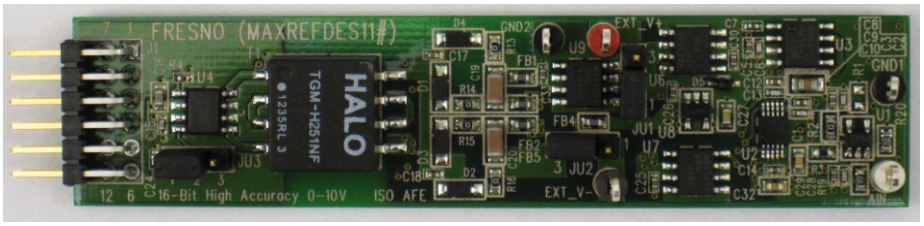
Jedro razvojne plošče ZedBoard predstavlja vgrajen ZC7Z020 Zynq čip. To je eden manjših predstavnikov Zynq družine in ima ARM procesorsko jedro spojeno z Artix-7 programirljivo logiko. Zraven ima še XADC »hard« IP komponento, ki ima funkcionalnost 12-bitnega AD pretvornika. Slika 3.3 nam prikazuje podrobnejšo zgradbo same razvojne plošče.



Slika 3.3: Predstavitev razvojnega sistema Zedboard [3]

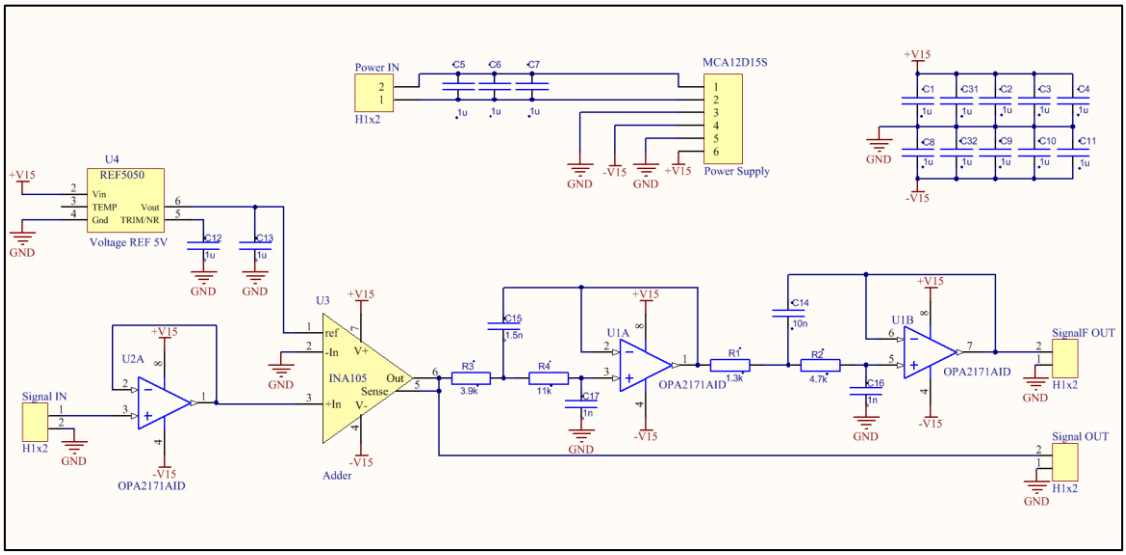
3.4 Zajem signala

Za zajem signala je uporabljen referenčni dizajn Fresno MAXREFDES11 podjetja Maxim. To je 16-bitni AD pretvornik, ki ima območje vhodne napetosti 0 – 10 V. Izvedeno ima tudi galvansko ločitev med vhodnim signalom in samo razvojno ploščo. Namenjen je uporabi predvsem v povezavi z industrijskimi senzorji, pri procesni avtomatizaciji [4].



Slika 3.4: AD pretvornik Fresno [4]

Vhodni signal iz pospeškomerja je bipolaren v območju -5 – 5 V, zato smo izdelali preprosto vezje, ki prestavi vhodni signal v območje 0 – 10 V. Vezje je sestavljeno iz napetostnega sledilnika na vhodu, nato sledi seštevalnik, ki vhodnemu signalu prišteje 5 V. Na koncu je še dodan aktivni nizkopasovni filter 4. reda. Sestavljen je iz dveh zaporednih filtrov 2. reda topologije Sallen-Key. Mejno frekvenco nastavimo z ustrezno izbiro elementov filtra in jo prilagodimo frekvenci vzorčenja v končni aplikaciji v skladu s teoremom o vzorčenju, ki je opisan v poglavju 2.2.

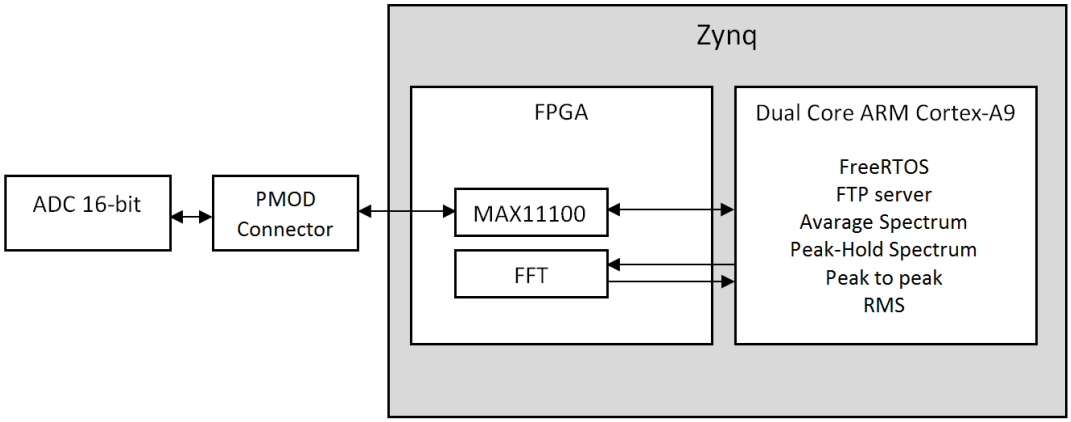


Slika 3.5: Shema vezja za napetostno prilagoditev

3.5 Arhitektura sistema

Slika 2.6 prikazuje zgradbo merilnega sistema. Zynq SOC ima vgrajen 12-bitni AD pretvornik, kar pa ne zadostuje zahtevam. Zato smo preko priključka PMOD priključili zunanji AD pretvornik Fresno, ki smo ga opisali zgoraj. Komunikacija z AD

pretvornikom poteka preko namenske IP komponente MAX11100, ki implementira SPI (ang. Serial Peripheral Interface) protokol. Podatki, ki jih zajemamo z AD pretvornikom, se nato shranijo v RAM (ang. Random Access Memory). V PS delu se ovrednoti časovni signal, nato pošljemo vzorce v PL del, kjer se izračuna spekter signala z uporabo FFT IP komponente. Ko so rezultati iz FFT komponente na voljo, jih prenesemo spet v RAM. Rezultati so kompleksne vrednosti. Z uporabo enačbe (2.7) izračunamo amplitudni in Peak Hold spekter. Izračunamo še RMS vrednosti v določenih frekvenčnih območjih, ki so nastavljena v konfiguracijski datoteki.

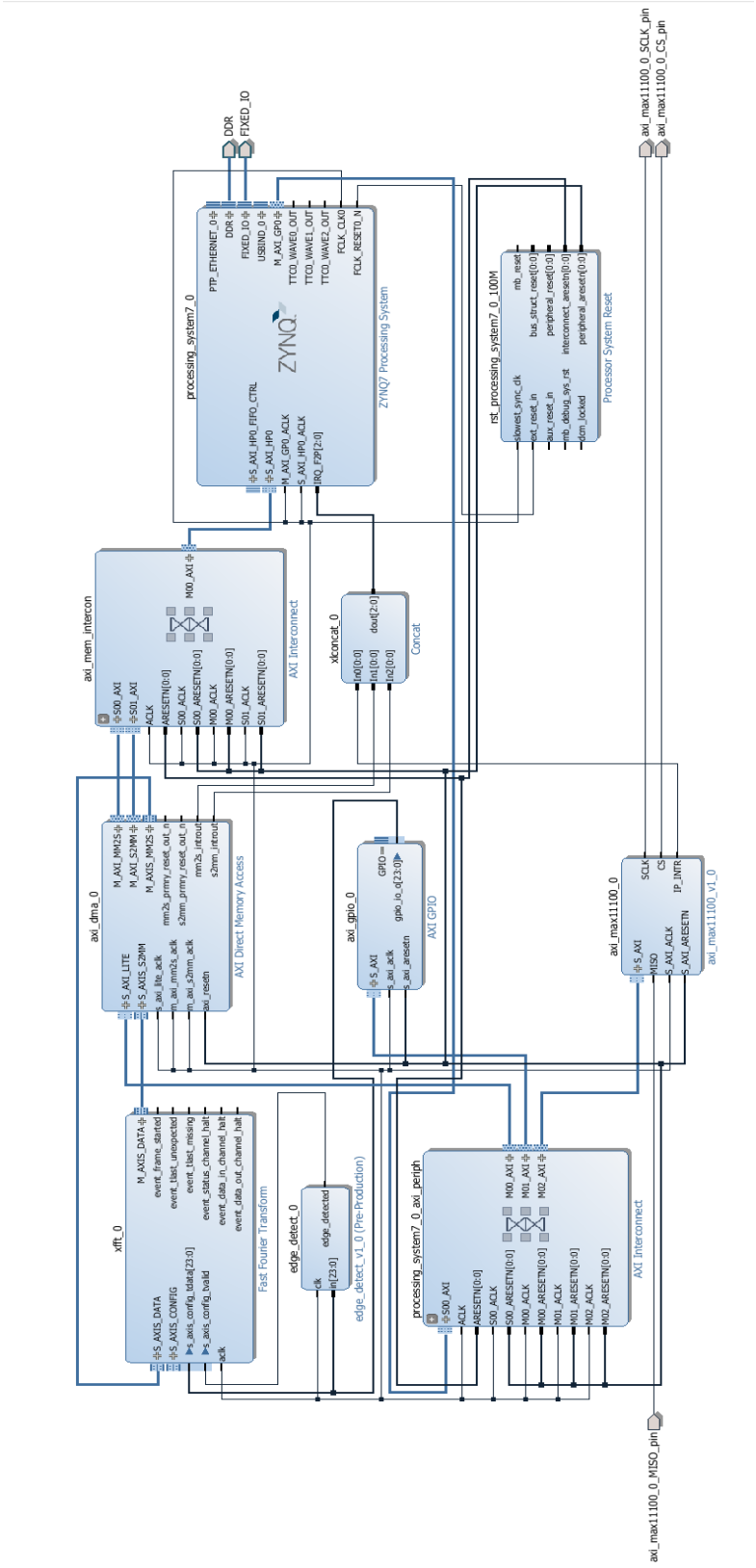


Slika 3.6: Blokovni diagram merilnega sistema

3.6 Izdelava strojne opreme

Za izdelavo sistema smo uporabili orodje Vivado 2015.2 proizvajalca Xilinx, ki vključuje tudi okolje za razvoj programske opreme SDK (ang. Software Development Kit). Ko se odločimo ustvariti projekt in želimo ustvariti svojo strojno opremo, najprej začnemo z orodjem Vivado ter ustvarimo nov projekt. Vivado nam omogoča ustvarjanje novih IP komponent ali pa uporabo že obstoječih. V našem primeru smo imeli vse IP komponente že na voljo in smo lahko začeli z ustvarjanjem blokovnega diagrama sistema.

Slika 3.7 prikazuje izdelan blokovni diagram v okolju Vivado 2015.2. V nadaljevanju poglavja so podrobneje opisani pomembnejši gradniki.



Slika 3.7: Blokovni diagram strojne opreme

3.6.1 Komponenta Zynq

Komponenta ZYNQ7 Processing System predstavlja procesorski sistem. V nastavitvah IP komponente moramo omogočiti funkcije, potrebne za uspešno povezovanje z ostalimi IP komponentami v blokovnem načrtu, izpostavili bomo nekaj pomembnejših.

- za povezavo z AXI GPIO omogočimo M AXI GP0 interface pod zavihkom PS_PL Configuration/ GP Master AXI Interface;
- za povezavo z AXI DMA omogočimo M AXI GP0 interface pod zavihkom PS_PL Configuration/ HP Slave AXI Interface;
- za nadzor nad DMA prenosi omogočimo prekinitve pod zavihkom Interrupts/ Fabric Interrupts/ PL-PS Interrupt Ports/ IRQ_F2P
- za povezavo Ethernet mora biti pod MIO Configuration/ I/O Peripherals omogočen ENET 0;

3.6.2 Komponenta FFT

FFT IP komponento smo generirali z uporabo Xilinx LogiCORE IP Fast Fourier Transform v9.0 generatorja. Ta implementira Cooley-Turkey FFT algoritem, ki je računsko manj zahtevna metoda izračuna DFT.

Glavne lastnosti:

- Dolžina transforma N = 2^m, m = 3 – 16
- Dolžina transforma je nastavljiva med delovanjem
- Med delovanjem lahko nastavljamo dolžino in smer kompleksnega FFT
- Tip podatkov:
  - Unscaled (full- precision) fixed-point
  - Scaled fixed-point
  - Block floating-point

V primeru, da so vhodni podatki v FFT IP komponenti tipa fixed-point, moramo zagotoviti, da med potekom algoritma ne pride do preliva (ang. overflow). FFT algoritem se izvaja v več zaporednih stopnjah. Število stopenj je določeno z dolžino transforma ter z uporabljeno arhitekturo (Radix-2 ali Radix-4). Določiti moramo skalirne faktorje, s katerimi se delni rezultati delijo na posamezni stopnji.

Končne rezultate moramo nato pomnožiti z produktom uporabljenih skalirnih faktorjev, da dobimo pravilne vrednosti rezultatov. [6]

Če nastavimo FFT IP komponento, da sprejema vhodne podatke tipa floating-point, nam ni potrebno nastavljati skalirnih faktorjev, saj IP komponenta sama poskrbi za to. Pridobimo tudi pri natančnosti izračuna, kot bo vidno v nadaljevanju. Cena za to pa je večja poraba FPGA programirljive logike. Odločili smo se za uporabo floating-point tipa, saj imamo na voljo dovolj celic FPGA programirljive logike in je v našem primeru bistvena natančnost izračuna.

FFT IP komponenta uporablja AXI4-Stream vodilo. V primeru izračuna s tipom floating-point so vhodni podatki 64-bitni. Prvih 32 bitov je realni del, naslednjih 32 bitov pa imaginarni del. Kot vhodne imaginarne podatke pošljemo ničle, saj so vzorčeni podatki iz AD pretvornika realna števila. Izhodni podatki so prav tako 64-bitni, prvih 32 bitov je realni del, naslednjih 32 bitov pa imaginarni del.

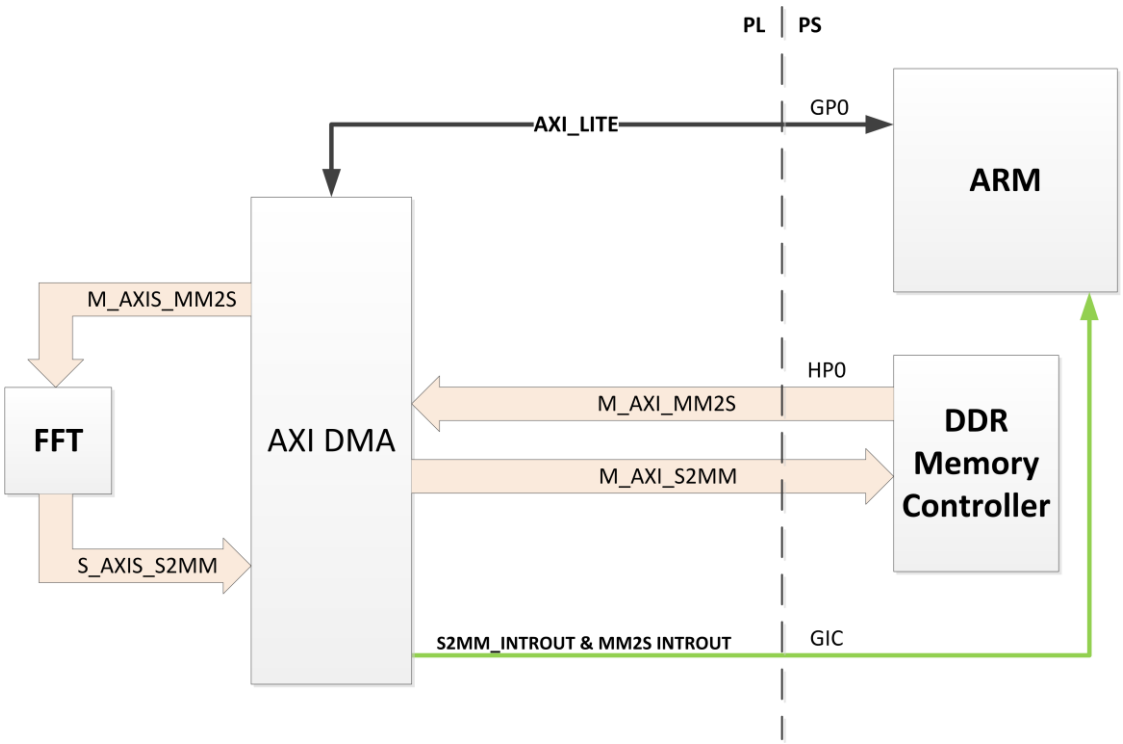
3.6.3 Komponenta MAX11100

Komponenta axi_max11100_0 skrbi za komunikacijo z AD pretvornikom preko SPI vodila. Do komponente dostopamo preko AXI4-Lite vodila z uporabo AXI GPIO komponente. Komponenti najprej nastavimo delovne registre, nato nam s proženjem prekinitve sporoča, kdaj je na voljo rezultat za branje.

3.6.4 Komponenta DMA

AXI DMA (ang. Direct Memory Access) omogoča hitro povezavo med sistemskim pomnilnikom in komponentami, ki uporabljajo AXI4-stream vodilo, v našem primeru je to FFT IP komponenta. Slika 3.8 nam prikazuje uporabo DMA IP komponente. Procesor in krmilnik spomina DDR se nahajata v Zynq PS delu, DMA in FFT IP komponenti pa v Zynq PL delu. AXI4-Lite vodilo omogoča procesorju, da komunicira z DMA za potrebe nastavitve in zagona prenosa podatkov. AXIS_MM2S in AXIS_S2MM sta AXI4-stream vodili, preko katerih poteka prenos toka (ang. stream) podatkov med FFT IP komponento ter DMA enoto. Prenos podatkov med DMA in DDR pomnilnikom poteka preko HP porta (ang. High performance AXI4 port), ki omogoča višjo hitrost prenosa, v primerjavi z GP portom (ang. General Purpose AXI4 port). Ob končanem prenosu v smeri S2MM (ang. stream to memory

mapped) ali MM2S (ang. memory mapped to stream) DMA sproži točno določeno prekinitev, ki oznanja, kdaj so podatki na voljo za branje.



Slika 3.8: Blokovna shema komunikacije z DMA enoto

Nastavitve DMA:

- Enable Read Chanel
- Enable Write Chanel
- Memory map Data Width = 64
- Stream data Width = 64
- Max Burst Size = 8
- Allow Unaligned Transfers
- Width of Buffer Length Register (8-23) = 22
- Address Width (32-64) = 32

4 Programska oprema

4.1 Operacijski sistem FreeRTOS

Ko smo pripravili strojno opremo, smo se lahko posvetili programskemu delu. Ena od zahtev za izdelavo merilnega sistema je bila ta, da poteka komunikacija preko FTP protokola. Zato je bilo smiselno raziskati, ali obstaja operacijski sistem, ki to omogoča. Hkrati smo želeli, da bi bil čim bolj enostaven za uporabo ter brezplačen. Trenutna aplikacija ne zahteva realno-časnosti, želeli pa smo, da bi aplikacijo lahko po potrebi nadgradili tudi v tej smeri. Tako smo se odločili za operacijski sistem FreeRTOS.

FreeRTOS je brezplačen odprtokoden realno-časni operacijski sistem, razvit s strani skupine Real Time Engineers Ltd. Razvit je bil z namenom, da bi se ga lahko uporabljalo na manj zmogljivih vgrajenih sistemih. Napisan je bil v programskem jeziku C. V času pisanja naloge je podprt za okoli 35 različnih arhitektur in je na trgu pod GPL (ang. General Public Licence) licenco, z izjemo, ki pravi, da je jedro odprtokodno, aplikacija, ki ga uporablja pa je lahko zaprto kodna.

Ponuja minimalno število funkcij, kot so: upravljanje z opravili, upravljanje s spominom, osnovne funkcije sinhronizacije. Omogoča prioritetno ali round robin razvrščanje, podpira sporočilne vrste (ang. message queues) binarne in številne semaforje. Odvisno od verzije za posamezen mikrokrmilnik, lahko ponuja še FAT datotečni sistem in HTTP oziroma FTP strežnik. [4]

4.2 Glavna funkcija

Spodaj se nahaja glavna funkcija (main) operacijskega sistema FreeRTOS. Če jo podrobneje pogledamo jo lahko razdelimo na:

- **Inicializacijo**
  - `prvMiscInitialisation();`
  - `FreeRTOS_IPInit( ucIPAddress, ucNetMask, ucGatewayAddress, ucDNSServerAddress, ucMACAddress );`
- **Kreiranje taskov**
  - `xTaskCreate( prvServerWorkTask, »SvrWork«, mainTCP_SERVER_STACK_SIZE, NULL, tskIDLE_PRIORITY + 2, &xServerWorkTaskHandle );`
  - `xTaskCreate( prvADCTask, »ADC«, mainADC_STACK_SIZE, NULL, tskIDLE_PRIORITY + 1, &xADCTaskHandle );`
- **Zagon razvrščevanika**
  - `vTaskStartScheduler();`
- **Neskončna zanka**
  - `for( ;; ){}`

Najprej se izvede inicializacija potrebne periferije, nadalje kreiramo opravila. Opravilo prvServerWorkTask nam poganja FTP server. Opravilo prvADCTask pa izvaja zajem in obdelavo vhodnega signala.

Ko so opravila ustvarjena, poženemo razvrščevalnik operacijskega sistema, ki skrbi za izvajanje kreiranih opravil. Opravila razvrša s prioritetnim algoritmom; to pomeni, da se izvaja opravilo z najvišjo prioriteto. V primeru, da imata obe opravili enako prioriteto, se izvajata izmenično.

Na koncu se nahaja neskončna zanka, do katere nikoli ne pridemo, če je z ustvarjenimi opravili ter inicializacijo vse v redu [2].

```
int main( void )
{
	/* Miscellaneous initialisation including preparing the logging and seeding
	the random number generator. */
	prvMiscInitialisation();

	/* Initialise the network interface.

	***NOTE*** Tasks that use the network are created in the network event hook
	when the network is connected and ready for use (see the definition of
	vApplicationIPNetworkEventHook() below).  The address values passed in here
	are used if ipconfigUSE_DHCP is set to 0, or if ipconfigUSE_DHCP is set to 1
	but a DHCP server cannot be  contacted. */
	FreeRTOS_printf( ( »FreeRTOS_IPInit\n« ) );
	FreeRTOS_IPInit( ucIPAddress, ucNetMask, ucGatewayAddress,
	ucDNSServerAddress, ucMACAddress );


	/* Create the task that handles the FTP and HTTP servers.  This will
	initialise the file system then wait for a notification from the network
```

```
        event hook before creating the servers.  The task is created at the idle
        priority, and sets itself to mainTCP_SERVER_TASK_PRIORITY after the file
        system has initialised. */
        xTaskCreate( prvServerWorkTask, »SvrWork«, mainTCP_SERVER_STACK_SIZE, NULL,
        tskIDLE_PRIORITY + 2, &xServerWorkTaskHandle );

        /* ADC   */
        xTaskCreate( prvADCTask, »ADC«, mainADC_STACK_SIZE, NULL, tskIDLE_PRIORITY +
        1, &xADCTaskHandle );

        /* Start the RTOS scheduler. */
        FreeRTOS_debug_printf( (»vTaskStartScheduler\n«) );
        vTaskStartScheduler();

        /* If all is well, the scheduler will now be running, and the following
        line will never be reached.  If the following line does execute, then
        there was insufficient FreeRTOS heap memory available for the idle and/or
        timer tasks    to be created.  See the memory management section on the
        FreeRTOS web site for more details.
        for( ;; )
        {
        }
}
```

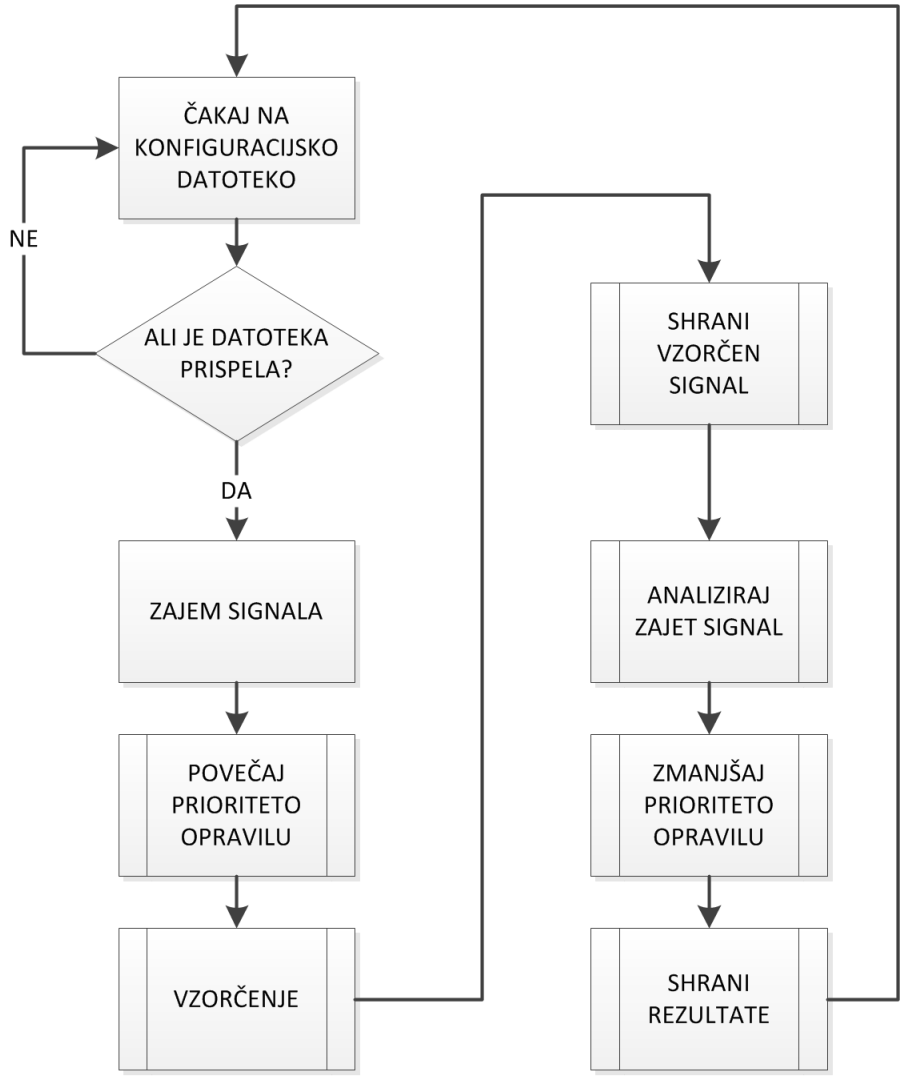
4.3 Opravilo za zajem in obdelavo signala

Opravilo za zajem in obdelavo signala (prvADCTask) lahko razdelimo na dve glavni stanji:

- čakaj na konfiguracijsko datoteko
- zajem in obdelava signala

V stanju »čakaj na konfiguracijsko datoteko« čakamo na konfiguracijsko datoteko, ki vsebuje nastavitve za meritev in obdelavo podatkov, obenem nam tudi proži meritev. Ko preko FTP povezave prispe konfiguracijska datoteka, jo preberemo in tako pridobimo potrebne nastavitve za začetek meritve.

Po uspešnem branju sledi stanje »zajem in obdelava signala«. Najprej zajamemo vzorce signala z nastavitvami, ki smo jih prebrali iz konfiguracijske datoteke, potem vzorce zapišemo v datoteko in jo shranimo v RAM ali na SD kartico. Prav tako so vse datoteke, ki jih generiramo, dostopne preko FTP strežnika. Sledi analiza zajetih vzorcev. Po analizi shranimo rezultate v datoteke Results.bin, AvarageSpectrum.bin in PHSpectrum.bin.



Slika 4.1: Blokovni diagram opravila ADCtask

Primer konfiguracijske datoteke je spodaj:

```
sampleMode = 1;
sampleRate = 10;
sampleSize = 8192
fftFrame = 8192;
overlapping = 0;
direction = 1;
windowing = 0;
LF = 0.2;
MF = 0.5;
HF = 1;
sensitivity = 0.01131;
```

Opis datoteke konfiguracija.txt:

- sampleMode:
  - 0 - meritev se proži vsako sekundo in se izpiše na UART
  - 1 – meritev se izvede za določeno število vzorcev z nastavljeno vzorčevalno frekvenco
- sampleRate:
  - 189 – vzorčevalna frekvenca je nastavljena na 189.4 kHz
  - 100 – vzorčevalna frekvenca je nastavljena na 100 kHz
  - 20 – vzorčevalna frekvenca je nastavljena na 20 Hz
  - 10 – vzorčevalna frekvenca je nastavljena na 10 kHz
  - 1 – vzorčevalna frekvenca je nastavljena na 1 kHz
  - Za vse ostale vrednosti je vzorčevalna frekvenca nastavljena na 0.1 kHz
- sampleSize: nastavimo lahko poljubno število vzorcev, ki jih želimo zajeti.
- fftFrame: nastaviti moramo dolžino FFT okvirja, ki je potenca števila 2: od 64 do 8192
- overlapping: nastavimo vrednost prekrivanja, med posameznimi FFT okvirji. Območje vrednosti je (0 – 100 %).
- direction: nastavimo tip FFT transforma
  - 0 – invertiran FFT transform
  - 1 – FFT transform
- windowing:
  - 0 – ne uporabimo oknenja
  - 1 – izvedemo oknenje nad vhodnim signalom
- LF: nastavimo območje nizkih frekvenc (0 - LF*fs/2)
- MF: nastavimo območje srednjih frekvenc (LF*fs/2 - MF*fs/2)
- HF: nastavimo območje visokih frekvenc (MF*fs/2 - HF*fs/2)
- sensitivity: nastavimo občutljivost pospeškomerja, v našem primeru 11,31 mV/g

```
while(1){
	switch(nMenuState)
	{
	case WAIT_FOR_FILE:		// macro definicija v menu.h
		xil_printf("\n\rIn the WAIT_FOR_FILE state \n\r");

		/* Preverjanje ali je prispela konfiguracijska datoteka -----*/
		fileExist = readFile(mainRAM_DISK_NAME);
		if (fileExist == 0) {
			vTaskDelay( 800 / portTICK_RATE_MS );
			nMenuState = WAIT_FOR_FILE;
		}
		else{
			nMenuState = START_SAMPLING;
		}
		break;
	case START_SAMPLING:
		/*-- VZORČENJE ------------------------------------------------*/
		printf("Sampling...\n");
		nNumberOfSamplesReceived=start_sampling(sampleSize,sampleRate,
		auSamples);

		/*-- SHRANJEVANJE VZORCEV -------------------------------------*/
		prvWriteADCSamplesFile( mainRAM_DISK_NAME );

		/*-- ANALIZA PODATKOV -----------------------------------------*/
		xil_printf("\n\rPerforming FFT\n\r");
		frequencyDomainAnalysis();
		timeDomainAnalysis(auSamples, unNumberOfSamplesReceived);

		/*-- SHRANJEVANJE REZULTATOV ----------------------------------*/
		prvWriteResultsFile( mainRAM_DISK_NAME );

		/* nastavim prioriteto IDLE taska */
		xil_printf("decrease priority\n\r");
		vTaskPrioritySet( NULL, uxPriority);
		nMenuState = WAIT_FOR_FILE;
		break;
	default:
		printf("Error, in the DEFAULT state");
		delay(ABOUT_ONE_SECOND * 5);
		nMenuState = WAIT_FOR_FILE;
		break;
	}
}
```

Analizo vhodnega signala razdelimo na dva dela:

- analiza v časovnem prostoru
  - določimo vršno vrednost signala vibracij: Vpp
  - izračunamo RMS vrednost : Vrms
- analiza v frekvenčnem prostoru:
  - izračunamo povprečen spekter
  - določimo peak-hold spekter
  - določimo RMS vrednosti v posameznih frekvenčnih območjih

Naslednji izsek kode nam predstavlja izračun vrednosti od vrha do vrha Vpp in efektivne vrednosti RMS. Na voljo imamo možnost izbire dolžine signala, kjer se izračunajo omenjene vrednosti, ki jih podamo s številom vzorcev. Rezultati posameznih okvirjev se nato zapišejo v zbirko rezultatov. Trenutno je nastavljena dolžina okvirja, enaka celotni dolžini signala, kar pomeni, da izračunamo eno vrednost Vpp ter Vrms za celoten vhodni signal. Po potrebi bi lahko vhodni signal razdelili na več odsekov.

```
// faktor za pretvorbo iz AD kode v enote pospeška [g]
Bit2G = BIT2V * (1/sensitivity);

numbersOfFrames = (numberOfSamples/frameLength) ; // izracunamo stevilo okvirjev
for(i=0; i < numbersOfFrames; i++ ){       // zunanja zanka tece po vsih okvirjih
       sum = 0;                                    // reset sum
       value = ((double) *samples - DC) * Bit2G;  // prvi vzorec
       max_value = value;           // reset max, priredim prvo vrednost v okvirju
       min_value = value;           // reset min, priredim prvo vrednost v okvirju
       for(j=0; j < frameLength; j++){ //  sesteva vzorce od neke pozicije naprej
              index = j + i * frameLength;        // index vzorca s katerim delam
                     value = ((double) samples[index] - DC) * Bit2G;
                     sum +=  pow(value,2) ;       // najpej kvadriramo in sestejemo
                     if (value > max_value)       // iscemo novi max
                            max_value = value;
                     if (value < min_value)       // iscemo novi min
                            min_value = value;
       }
       rms_buf[i] = (float) sqrt(sum / frameLength); // delimo z dolzino okvirja
       vpp_buf[i] = (float) (max_value - min_value);
       Vpp = vpp_buf[i];                           // shranimo Vpp vrednost
}
```

Spodaj je predstavljen odsek kode, s katerim izračunamo povprečen in Peak-Hold spekter. Ker je naš signal lahko poljubno dolg, dolžina FFT okvirja pa je omejena ($N = 2^m$, $m = 3 – 13$), moramo najprej izračunati število izračunov FFT. Število okvirjev je odvisno od dolžine zajetega signala, dolžine sekvence FFT ter vrednosti prekrivanja sosednjih okvirjev. Ob uporabi oknenja je potrebno od vhodnega signala odšteti srednjo vrednost, ki ustreza enosmerni komponenti, saj v nasprotnem oknenje slabo vpliva na izračun spektra vhodnega signala.

Zajeti vzorci z AD pretvornika so shranjeni v zbirki tipa u16 (ang. unsigned integer velikosti 16 bitov), kar je ustrezno za 16-biten AD pretvornik, in tako ne zasedamo pomnilnika po nepotrebnem. Za nadaljnjo obdelavo pa je zapis v u16 neustrezen, saj ne omogoča zapisa v ustrezni natančnosti.

Sledi odštevanje enosmerne komponente in oknenje po enačbi (2.10). Rezultat shranimo v zbirko tipa float. To je zapis z plavajočo vejico, ki ustreza potrebam po natančnosti.

Naslednji korak je priprava zbirke za pošiljanje FFT IP komponenti. Ta je konfigurirana tako, da sprejme vhodne podatke, ki so tipa float. Prav tako pričakuje, da so vzorci kompleksna števila, ki so sestavljena iz dveh števil tipa float, prvo predstavlja realni del, drugo pa imaginarni del. V našem primeru, ko imamo realen signal, moramo za vsak vzorec poslati najprej realni del, ki je zajet z AD pretvornikom tipa float, in nato še imaginarni del, ki ga postavimo na nič, prav tako tipa float.

Sledi prenos podatkov z uporabo DMA. Ko podatki prispejo iz FFT IP komponente, jih moramo pravilno interpretirati. Tako kot poslani podatki, so to kompleksna števila, sestavljena iz dveh števil realnega in imaginarnega dela. Rezultat, ki nam ga vrne FFT blok, je linearni amplitudni kompleksni spekter vhodnih vzorcev in ga shranimo v zbirko X[i]. Nas zanima močnostni spekter, zato X[i] kvadriramo in normiramo z dolžino NFFT. V primeru, ko imamo na voljo dovolj vzorcev, ponovimo izračun FFT večkrat ter povprečimo rezultate. Hkrati shranjujemo maksimalne vrednosti posamezne frekvenčne komponente, ki nam tvorijo Peak-Hold spekter.

```
// parameter za pretvorbo v enoto pospeška g
Bit2G = BIT2V * (1/sensitivity);

// pobrisem avarage in PH array
for(i=0; i<NFFT; i++){
	avarageSpectrum[i] = 0;
	PHSpectrum[i] = 0;
}
// izbira dolžine okna
switch (NFFT){
	case 16 :	window_ptr = hann16;	break;
	case 32 :	window_ptr = hann32;	break;
	case 64 :	window_ptr = hann64;	break;
	case 128 :	window_ptr = hann128;	break;
	case 256 :	window_ptr = hann256;	break;
	case 512 :	window_ptr = hann512;	break;
	case 1024 :	window_ptr = hann1024;	break;
	case 2048 :	window_ptr = hann2048;	break;
	case 4096 :	window_ptr = hann4096;	break;
	case 8192 :	window_ptr = hann8192;	break;
	default :	xil_printf("Error in choosing window \n\r ");
}

// Zapišem parametre FFT
status = fft_config(fft_params);
if (status != XST_SUCCESS){
	xil_printf("ERROR! Failed to configure the FFT core.\r\n", status);
	return status;
```

```
}

// Izračun srednje vrednosti
DC = 0;
for(i=0; i<unNumberOfSamplesReceived; i++){
	DC = DC + ((double) stim_buf[i] / unNumberOfSamplesReceived);
}

// Izračun števila FFT okvirjev
numberOfFFTFrames = ((unNumberOfSamplesReceived - NFFT) / (int) (NFFT - ((float)
overlapping/100) *NFFT)) + 1 ;	// delimo s 100 da dobimo procente

// zanka gre čez vse okvirje
for(j=0; j<numberOfFFTFrames; j++){

	// začetna pozicija v zbirki kjer so shranjeni vzorci
	offset = j * (int) (NFFT - ((float) overlapping/100) * NFFT);
	if (j == numberOfFFTFrames-1)
		xil_printf("offset = %d\r\n", offset);

	// odštejem srednjo vrednost in po potrebi izvedem oknenje
	if (windowing == 1){
		for(i=0; i<NFFT; i++){
			stim_w[i] = *(window_ptr+i) * (((float) stim_buf[offset + i]) -
			DC) * Bit2G;
		}
	}
	else{
		for(i=0; i<NFFT; i++){
			stim_w[i] = (((float) stim_buf[offset + i]) - DC) * Bit2G;
		}
	}

	// kreiram zbirko, katera ima dodane ničle za imaginarni del
	for(i=0; i<NFFT; i++)
	{
		stim_w_im[2*i] = stim_w[i];
		stim_w_im[2*i+1] = 0;		// imaginarni del postavimo na 0
	}
	Sproži DMA prenos
	status = dma_xfer(fft_params, tempResultFFT, stim_w_im);//
	if (status != XST_SUCCESS){
		xil_printf("ERROR! Failed to kick off dma transfer.\r\n", status);
		return status;
	}

	// ko so na voljo podatki iz DMA
	// izračunam kvadrat amplitudnega spektra
	for (i = 0; i <= halfSpectrum; i++){	//poberem samo pol spektra
		xk_re = tempResultFFT[2*i];	// Lower bits RE
		xk_im = tempResultFFT[2*i+1];	// Upper bits IM
		X[i] = xk_re + xk_im * I;
		if (windowing == 1){
			// Korekcija energije ob uporabi Hann okna 1.63^2
			// kvadrat polovice spektra
			XX[i] = HannCorectionFactor * ((creal(X[i]) * creal(X[i]) +
			cimag(X[i]) * cimag(X[i])) / (NFFT*NFFT));
		}
		else{
			// kvadrat polovice spektra
			XX[i] = (creal(X[i]) * creal(X[i]) + cimag(X[i]) * cimag(X[i]))
			/ (NFFT*NFFT);
		}
```

```
                if(XX[i] > PHSpectrum[i] ){
                        // osvezevanje Peak Hold Spektra, global
                        PHSpectrum[i] = XX[i];
                }
                // global
                avarageSpectrum[i] =  avarageSpectrum[i] + ((float)
                1/numberOfFFTFrames) * XX[i];
        }
}
```

4.4 Funkcija za ovrednotenje frekvenčnega spektra

Naloga funkcije za ovrednotenje spektra (evaluateSpectrum) je izračun RMS vrednosti signala vibracij v posameznem frekvenčnem območju. Običajno se pri meritvah vibracij celotno območje razdeli na tri dele: LF (ang. Low Frequency), MF(ang. Medium Frequency) in HF (ang. High Frequency), ki opredeljujejo frekvenčno območje v naslednjih mejah:

- LF (0 – 100 Hz)
- MF (100 – 1000 Hz)
- HF(1 kHz – Fs/2)

Po enačbi 2.15 izračunamo RMS vrednost signala v celotnem ter v posameznih frekvenčnnih območjih, ki so nastavljena v konfiguracijski datoteki. Na koncu jih še shranimo v globalne spremenljivke. Le-te bere funkcija za shranjevanje rezultatov.

```
for (i = 0; i <= halfSpectrum; i++) //vzamem pol spektra
{
        if( (i != 0){
        pxx[i] = 2.0 * avarageSpectrum[i];  //AC
        }
        else {
                pxx[i] = avarageSpectrum[i]; //DC
        }
        if(i<=(LF*halfSpectrum) ){
                pxx_LF += pxx[i];
        }
        else if( (i>(LF*halfSpectrum )) && (i<=(MF*halfSpectrum)) ){
                pxx_MF += pxx[i];
        }
        else{
                pxx_HF += pxx[i];
        }
        sum_pxx += pxx[i];
}
// shranim v globalne spremenljivke
RMS_pxx = sqrt(sum_pxx);
RMS_LF = sqrt(pxx_LF);
RMS_MF = sqrt(pxx_MF);
RMS_HF = sqrt(pxx_HF);
```

4.5 Branje konfiguracijske datoteke

Kot že zgoraj opisano, aplikacija nenehno preverja ali je prispela konfiguracijska datoteka, ki nam proži meritev. Najprej jo preberemo in shranimo potrebne nastavitve, nato jo izbrišemo. Izbrisati jo moramo zato, da lahko zaznamo nov zahtevek za meritev, ko datoteka ponovno prispe prek FTP povezave.

Za branje konfiguracijske datoteke skrbi funkcija readFile. Z vhodnim parametrom povemo, iz katerega direktorija naj funkcija bere. Ob klicu funkcije se najprej preveri, ali konfiguracijska datoteka obstaja. Če ne, se vrnemo, če pa datoteka obstaja, funkcija najprej poveča prioriteto opravilu prvADCTask. Nadalje se izvede branje ter shranjevanje parametrov v globalne spremenljivke. Funkcija vrne vrednost 1 ob uspešno prebrani datoteki, 0 pa v nasprotnem primeru. Prioriteto opravila prvADCTask spet zmanjšamo, ko se zaključi zajem signala vibracij ter obdelava.

```
int readFile(const char *pcMountPath )
{
	int i,j=0;
	char var1[20], var2[15], var3[15], var4[15], var5[15], var6[15], var7[15],
	var8[15], var9[15], var10[15], var11[15];  // imena spr. iz konfiguracija.txt
	char buf[CONFIG_DATA_PACKET_LEN+1];
	char string[1000];    // hrani vsebino datoteke konfiguracija.txt
	unsigned portBASE_TYPE uxPriority;

	/*-- FAT datotečni sistem----*/
	size_t xItemsRead;
	int32_t lResult;
	FF_FILE *pxFile;
	char *pcRAMBuffer, *pcFileName;

	/* Alociranje pomnilnika, ki hrani podatke, za pisanje/branje na  disk, ter
	imena datotek */
	pcRAMBuffer = ( char * ) pvPortMalloc( fsRAM_BUFFER_SIZE * 3 );
	pcFileName = ( char * ) pvPortMalloc( ffconfigMAX_FILENAME );
	configASSERT( pcRAMBuffer );
	configASSERT( pcFileName );

	/* vprasamo po prioriteti tega taska */
	uxPriority = uxTaskPriorityGet( NULL );
	xil_printf("Priority of ADC task = %d\n\r", uxPriority);

	/* Zagotovimo da se nahajamo v izbrani mapi. */
	lResult = ff_chdir( pcMountPath );
	configASSERT( lResult >= 0 );

	/* Kreiranje imena datoteke */
	snprintf( pcFileName, ffconfigMAX_FILENAME, "konfiguracija.txt" );

	/*Pridobi ime direktorija kjer se nahajamo ter izpiši ime datoteke ter
	direktorij iz katerega beremo*/
	ff_getcwd( buf, fsRAM_BUFFER_SIZE );
	/* Odpri datoteko za branje */
	pxFile = ff_fopen( pcFileName, "r" );
```

```
	if (pxFile == 0) {
		/* Zapri datoteko. */
		ff_fclose( pxFile );

		/* Sprosti zaseden prostor */
		vPortFree( pcRAMBuffer );
		vPortFree( pcFileName );

		return 0;
	}
	else {
	/*-- Branje datoteke ---*/

	/* nastavimo na najvisjo prioriteto */
	vTaskPrioritySet( NULL, ( uxPriority + 3 ) );

	/* Na začetku pobrišemo pomnilnik */
	memset( pcRAMBuffer, 0x00, fsRAM_BUFFER_SIZE );
	xItemsRead = ff_fread( pcRAMBuffer, 1, fsRAM_BUFFER_SIZE, pxFile );
	if(xItemsRead == 0)   xil_printf("Error reading konfiguracija.txt file");

	/* pisemo na zacetek zbirke */
	j=0;
	i=0;
	while(pcRAMBuffer[i] != '*'){
		xil_printf("%c", pcRAMBuffer[i]);
		if(pcRAMBuffer[i] == ';')
			pcRAMBuffer[i] = ' ';
		if(pcRAMBuffer[i] == '=')
			pcRAMBuffer[i] = ' ';
		if( (pcRAMBuffer[i] != '\n') &&  (pcRAMBuffer[i] != '\r')){
			string[j] = pcRAMBuffer[i];
			j++;
		}
		i++;
	}
	/* Zaprem datoteko */
	ff_fclose( pxFile );

	/* Zbrišem datoteko */
	ff_remove( "/ram/konfiguracija.txt" );

	/* sprostim pomnilnik */
	vPortFree( pcRAMBuffer );
	vPortFree( pcFileName );

	/* preberem string in ga shranim kot integer v spremenljivko */
	sscanf( string,"%s %d %s %d %s %d %s %d %s %d %s %d %s %d %s %f %s %f %s %f",
	var1, &samplingMode,
			var2, &sampleRate,
			var3, &sampleSize,
			var4, &fftFrame,
			var5, &overlapping,
			var6, &direction,
			var7, &windowing,
			var8, &LF,
			var9, &MF,
			var10, &HF);
	return 1;
	}
}
```

4.6 Shranjevanje rezultatov

Shranjevanje rezultatov nam opravi funkcija prvWriteResultsFile. Na voljo imamo dve lokaciji, kamor lahko shranjujemo. Lahko shranimo v RAM ali pa na SD kartico.

```
/*-- SHRANJEVANJE REZULTATOV --*/

// shranjevanje v RAM
	prvWriteResultsFile( mainRAM_DISK_NAME );
// shranjevanje na SD kartico
	prvWriteResultsFile( mainSD_CARD_DISK_NAME );
```

Sledi opis funkcije prvWriteResultsFile. Najprej shranimo povprečen spekter v datoteko AvarageSpectrum.bin. Potem sledi shranjevanje Peak Hold spektra v datoteko PHSpectrum.bin. Sledi še shranjevanje posameznih cenilk v datoteko Results.bin. Shranjevanje poteka v naslednjih korakih:

- dodelitev dinamičnega pomnilnika, ki hrani podatke,
- zapis želenih podatkov v dodeljen pomnilnik,
- kreiranje datoteke v katero bomo pisali,
- pisanje v datoteko,
- datoteko zapremo,
- sprostitev dinamičnega pomnilnika.

```
static void prvWriteResultsFile( const char *pcMountPath )
{
	int32_t lItemsWritten;
	int32_t lResult;
	FF_FILE *pxFile;
	double *pcRAMBuffer;
	char *pcFileName;
	int i;

	/* Allociranje pomnilnika, ki hrani podatke, za pisanje/branje na  disk, ter
	imena datotek */
	pcRAMBuffer = (double *)pvPortMalloc((fftFrame/2)* sizeof(double));
	pcFileName = ( char * ) pvPortMalloc( ffconfigMAX_FILENAME );
	configASSERT( pcRAMBuffer );
	configASSERT( pcFileName );

	/* Zagotovimo da se nahajamo v izbrani mapi. */
	lResult = ff_chdir( pcMountPath );
	configASSERT( lResult >= 0 );

	/* Avarage Spectrum ------------------------------------------------------*/
	/* Vpis v RAM buffer */
	for(i = 0; i < (fftFrame/2); i++)
		pcRAMBuffer[i] = avarageSpectrum[i];

	/* Kreiranje imena datoteke */
	snprintf( pcFileName, ffconfigMAX_FILENAME, "AvarageSpectrum.bin");
	xil_printf( "\n\rCreating file %s\n", pcFileName );
```

```
	/* Odpiranje datoteke ali ustvarjanje nove če ne obstaja  */
	pxFile = ff_fopen( pcFileName, "w" );
	configASSERT( pxFile );

	/* Zapis v datoteko */
	lItemsWritten = ff_fwrite( pcRAMBuffer, (fftFrame/2) * sizeof(double), 1,
	pxFile );
	configASSERT( lItemsWritten == 1 );

	/* Zaprem datoteko */
	ff_fclose( pxFile );

	/* PHSpectrum ----------------------------------------------------------*/
	/* Vpis v RAM buffer */
	for(i = 0; i < (fftFrame/2); i++)
		pcRAMBuffer[i] = PHSpectrum[i];

	/* Kreiranje imena datoteke */
	snprintf( pcFileName, ffconfigMAX_FILENAME, "PHSpectrum.bin");
	xil_printf( "\n\rCreating file %s\n", pcFileName );

	/* Odpiranje datoteke ali ustvarjanje nove če ne obstaja  */
	pxFile = ff_fopen( pcFileName, "w" );
	configASSERT( pxFile );

	/* Zapis v datoteko */
	lItemsWritten = ff_fwrite( pcRAMBuffer, (fftFrame/2) * sizeof(double), 1,
	pxFile );
	configASSERT( lItemsWritten == 1 );

	/* Zaprem datoteko */
	ff_fclose( pxFile );

	/* Rezultati RMS_pxx, Vpp, DC ------------------------------------------*/
	/* Vpis v RAM buffer */
	pcRAMBuffer[0] = RMS_pxx;
	pcRAMBuffer[1] = RMS_LF;
	pcRAMBuffer[2] = RMS_MF;
	pcRAMBuffer[3] = RMS_HF;
	pcRAMBuffer[4] = Vpp;
	pcRAMBuffer[5] = DC*BIT2V;


	/* Kreiranje imena datoteke */
	snprintf( pcFileName, ffconfigMAX_FILENAME, "Results.bin");
	xil_printf( "\n\rCreating file %s\n", pcFileName );

	/* Odpiranje datoteke ali ustvarjanje nove če ne obstaja  */
	pxFile = ff_fopen( pcFileName, "w" );
	configASSERT( pxFile );

	/* Zapis v datoteko */
	lItemsWritten = ff_fwrite( pcRAMBuffer, 6 * sizeof(double), 1, pxFile );
	configASSERT( lItemsWritten == 1 );

	/* Zaprem datoteko */
	ff_fclose( pxFile );

	/* Sprosti zaseden prostor */
	vPortFree( pcRAMBuffer );
	vPortFree( pcFileName );
}
```

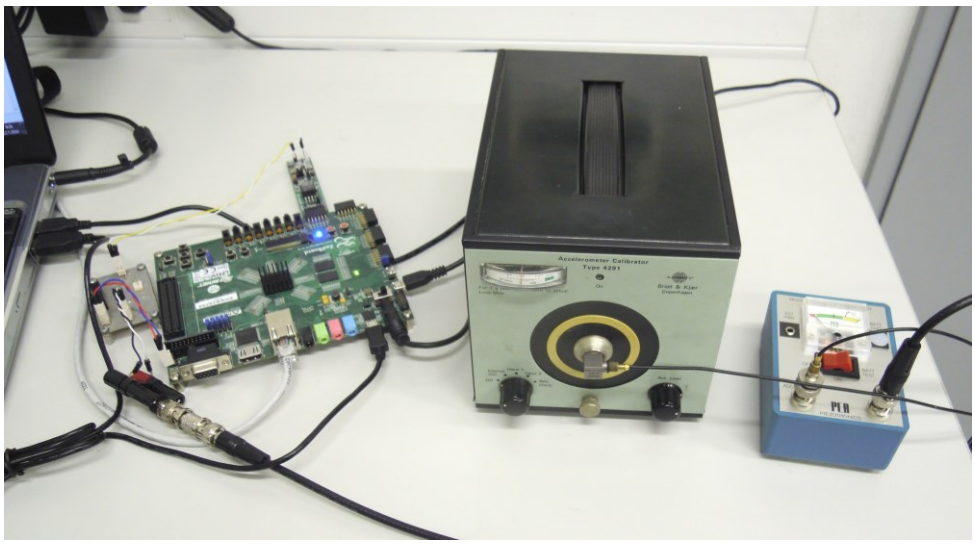
5 Meritev

5.1 Meritve vibracij

Za meritev vibracij potrebujemo senzor – pospeškomer, ki ga pritrdimo na površino, ki vibrira. Pospeškomer pretvori mehanski pospešek v ustrezno napetost. Nato z merilnim sistemom zajamemo napetostni signal iz pospeškometra ter ga s programsko opremo ustrezno ovrednotimo. Najprej ga ovrednotimo v časovnem prostoru, in nato še v frevenčnem, kar nam omogoča globlje razumevanje karakteristike vibracij ter natančnejše razpoznavanje dobrih ali slabih merjencev. Odvisno od merjenega stroja določimo nivo vibracij, ki je ustrezen ali ne.

5.2 Opis meritve

Na sliki 5.1 je prikazana testna meritev, s katero smo primerjali izdelan merilni sistem s profesionalnim merilnim sistemom LMS Scadas. Na sredini je naprava, ki se uporablja za kalibracijo pospeškomerjev in smo jo uporabili kot vir vibracij.



Slika 5.1: Prikaz testne meritve

Generira vibracije s frekvenco 80 Hz ter amplitudo 1g. Na desni je napajanje senzorja (napajanje s konstantnim tokom 2 mA), kot ga zahteva pospeškomer. Na levi strani pa je Zedboard s priključenim AD pretvornikom preko PMOD priključka. Ker je vhodni signal bipolaren v območju -5 – 5 V, vhodno območje AD pretvornika pa je 0 – 10 V, je vmes še vezje, ki vhodnemu signalu pripne 5 V. Hkrati ima še vlogo filtra proti prekrivanju. Slika 5.2 prikazuje tokovno napajanje pospeškomerja proizvajalca PCB Piezotronics. Napaja se lahko z zunanjim napajanjem ali preko baterij. Na XDCR vhod priklopimo pospeškomer, na SCOPE izhod pa priključimo merilni sistem. Analogni prikazovalnik nam prikazuje le stanje baterij.

Slika 5.2: Napajanje pospeškomerja s konstantnim tokom

Slika 5.3 prikazuje uporabljen enoosni IEPE (ang. Integrated Electronic Piezielectric) pospeškomer proizvajalca Bruel&Kjaer. IEPE pespeškomerji so zvrst piezoelektričnih pospeškomerjev, ki imajo vgrajen nabojni ojačevalnik. Prednost tega je, da ne potrebujejo dodatnega zunanjega ojačevalnika, majhna velikost ter priklop z enojnim kaoksialnim kablom. Kot že omenjeno, zahtevajo napajanje s konstantnim tokom, v našem primeru je to 2 mA, običajno pa se vrednost giblje med 2 mA do 4 mA.

Slika 5.3: Uporabljen pospeškomer B&K 4507B

5.3 Primerjava meritev

Slika 5.4 prikazuje zajem signala z Zedboardom ter obdelavo. Prvi graf na sliki prikazuje zajet referenčni signal vibracij jakosti 1g in frekvence 80 Hz. Amplituda signala je 11,3 mV kar da ob občutljivosti pospeškomerja 11,31 mV/g ravno amplitudo vibracij 1 g.

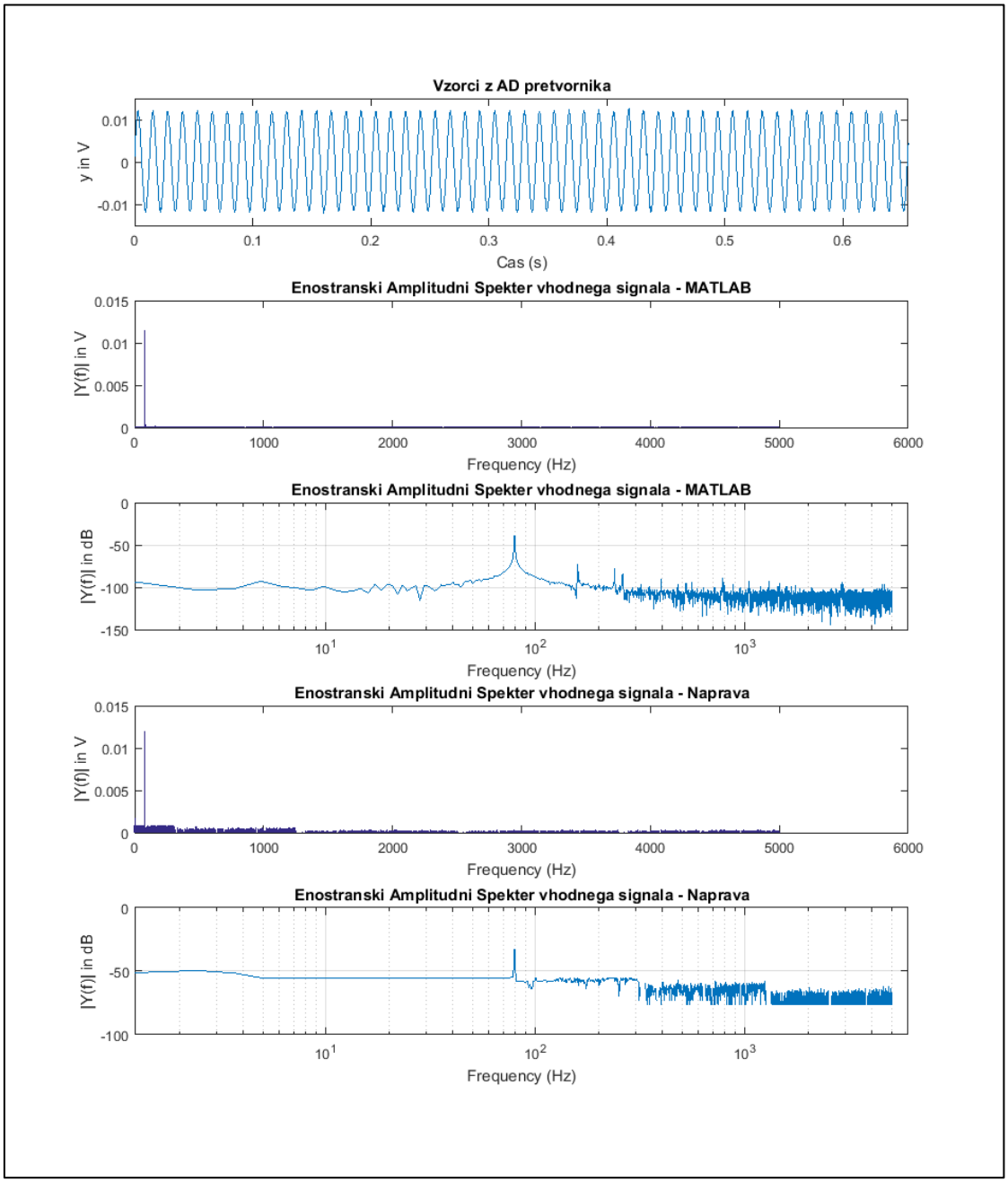
Parametri :

- Vzorčevalna frekvenca: fs = 10 kHz
- Število vzorcev: N = 8192
- Dolžina onvirja FFT: NFFT = 8192
- Uporabljeno okno: Hanning

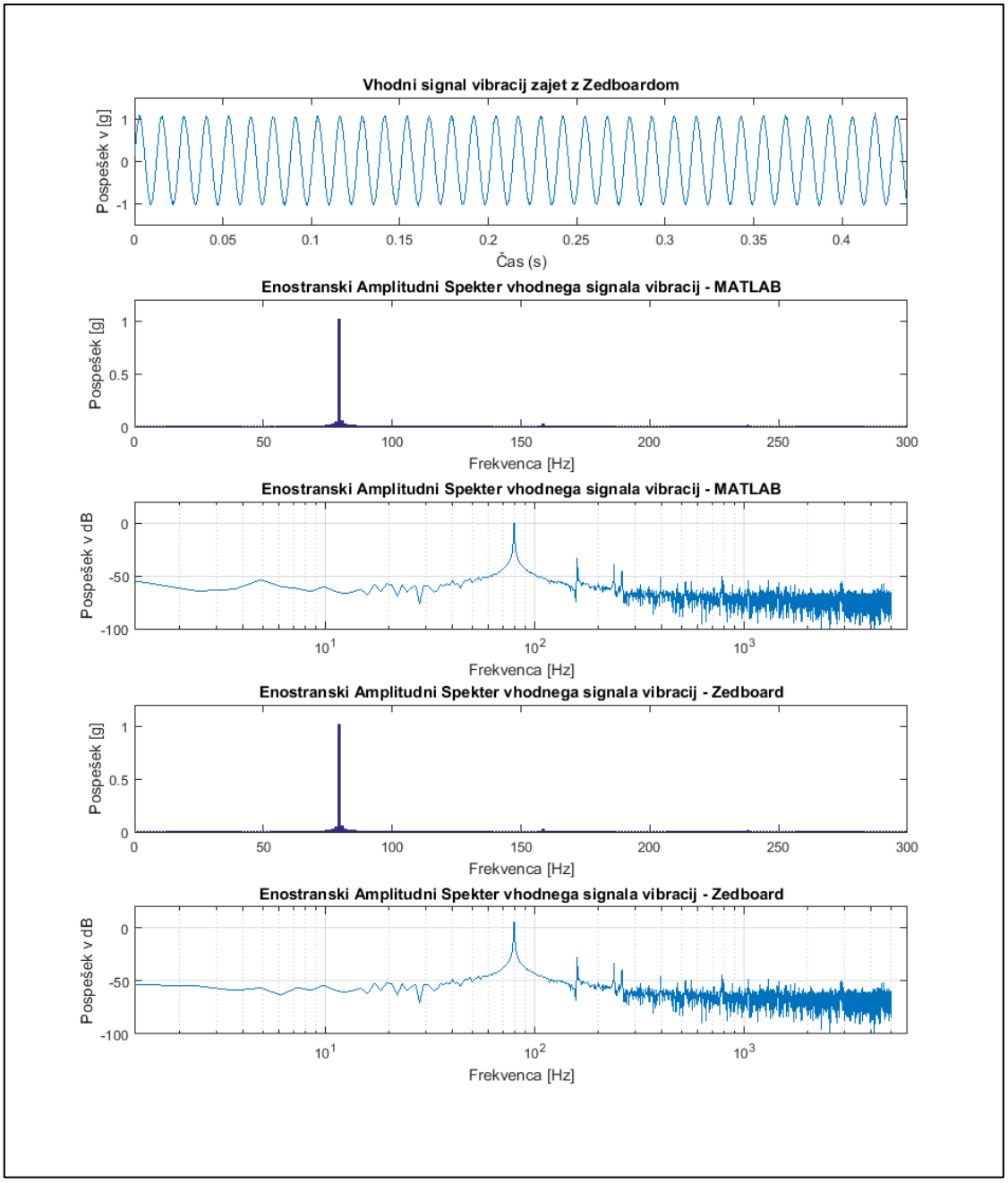
Na drugem grafu je izrisan linearni enostranski amplitudni spekter, izračunan s programskim paketom Matlab, vidna je frekvenčna komponenta pri 80 Hz. Na tretjem grafu je prikazan isti amplitudni spekter, le skala amplitude je v decibelih. Na tem grafu so bolje vidne višje frekvenčne komponente, ki imajo precej manjšo amplitudo.

Na četrtem grafu je prikazan linearni amplitudni spekter, izračunan z Zedboardom. FFT IP blok je bil konfiguriran tako, da je sprejel števila tipa integer (cela števila z natančnostjo 32 bitov). V primerjavi z izračunom, ki je narejen s programskim paketom Matlab, je razvidno, da so se pojavile višje frekvenčne komponente, ki so posledica omejene natančnosti izračuna. Na petem grafu je prikazan isti linearni amplitudni spekter vhodnega signala vibracij z lagaritemsko skalo amplitude. Razmerje signal šum SNR (ang. signal to noise ratio) je tu 20dB v primerjavi z izračunom opravljenim z Matlabom, kjer je SNR 60dB. Iz tega lahko zaključimo, da uporaba števil tipa integer za izračun FFT algoritma ni primerna, ko je želena čim višja ločljivost, ki jo lahko dosežemo z 16-bitnim AD pretvornikom. Boljša

rešitev je izračun FFT algoritma s tipom števil float (zapis števil s plavajočo vejico, enojna natančnost 32 bitov). Cena za to je večja poraba vezja FPGA, kar pa v našem primeru ni bila omejitev. Razvidno je razmerje signal šum 60dB v obeh primerih. Enakost spektrov je vidna na sliki 5.5, kjer je skala amplitude preračunana v enote pospeška [g]. Sliki 5.4 in 5.5 prikazujeta izračun frekvenčnega spektra brez uporabe oknenja.

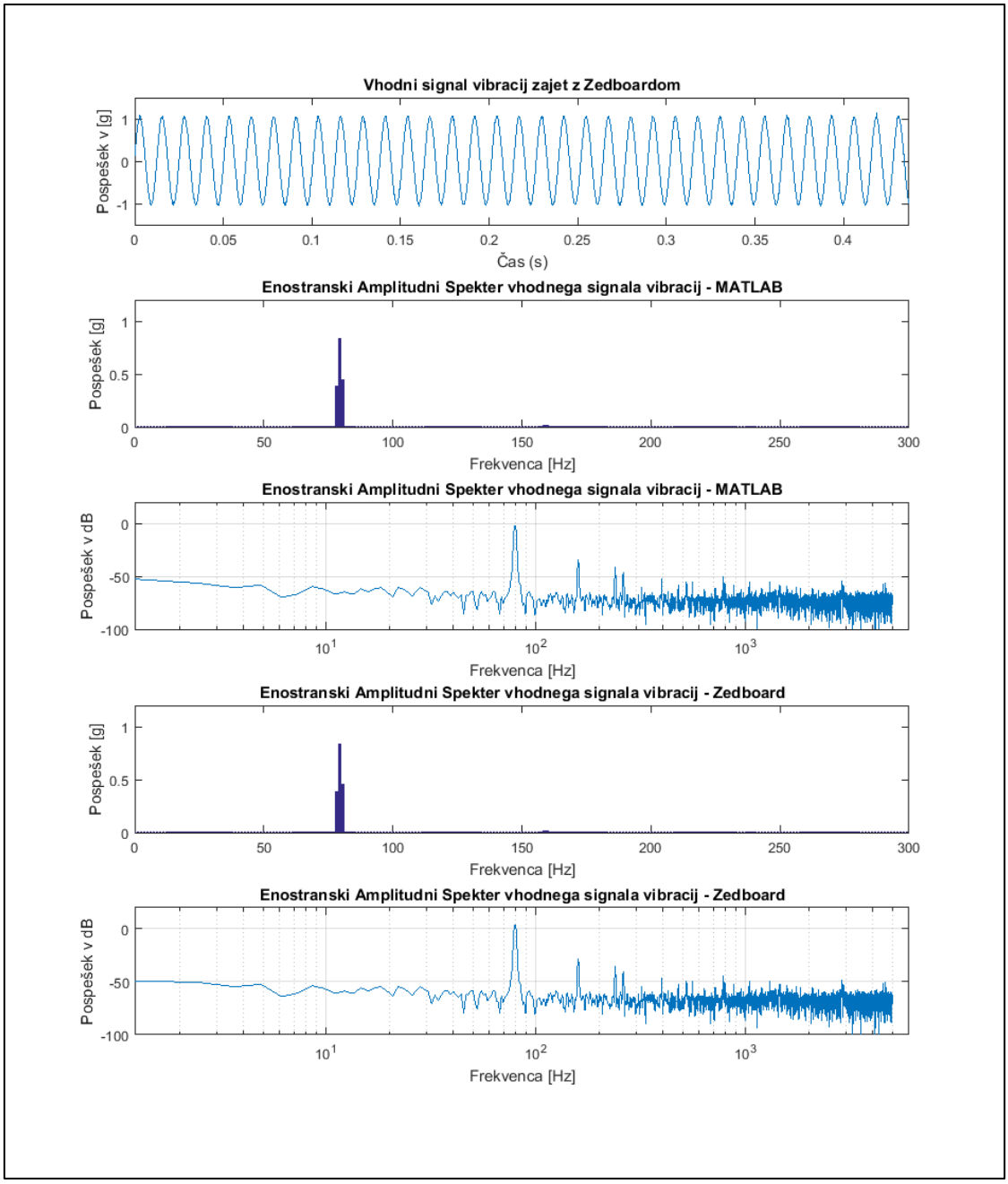


Slika 5.4: FFT izračun z tipom števil integer



Slika 5.5: FFT izračun s tipom števil float– brez uporabe oknenja

Slika 5.6 prikazuje uporabo oknenja. V primerjavi s sliko 5.5 lahko opazimo, da so frekvenčne komponente pri 170 Hz in 250 Hz precej lažje razpoznavne. Prav tako se frekvenčna komponenta pri 80 Hz precej manj razleze v sosednje frekvenčne komponente. Ob primerjavi slik 5.5 in 5.6 je vidna prednost uporabe okenskih funkcij.



Slika 5.6: FFT izračun z tipom števil float– uporaba oknenja

Meritev in rezultati obdelave vhodnega signala so dostopni preko FTP povezave. Na Zedboardu teče operacijski sistem FreeRTOS, ki poganja FTP strežnik. Nanj se povežemo s programom, ki omogoča FTP povezavo.

Slika 5.7 prikazuje dostop s programom FileZilla. Na voljo imamo naslednje datoteke:

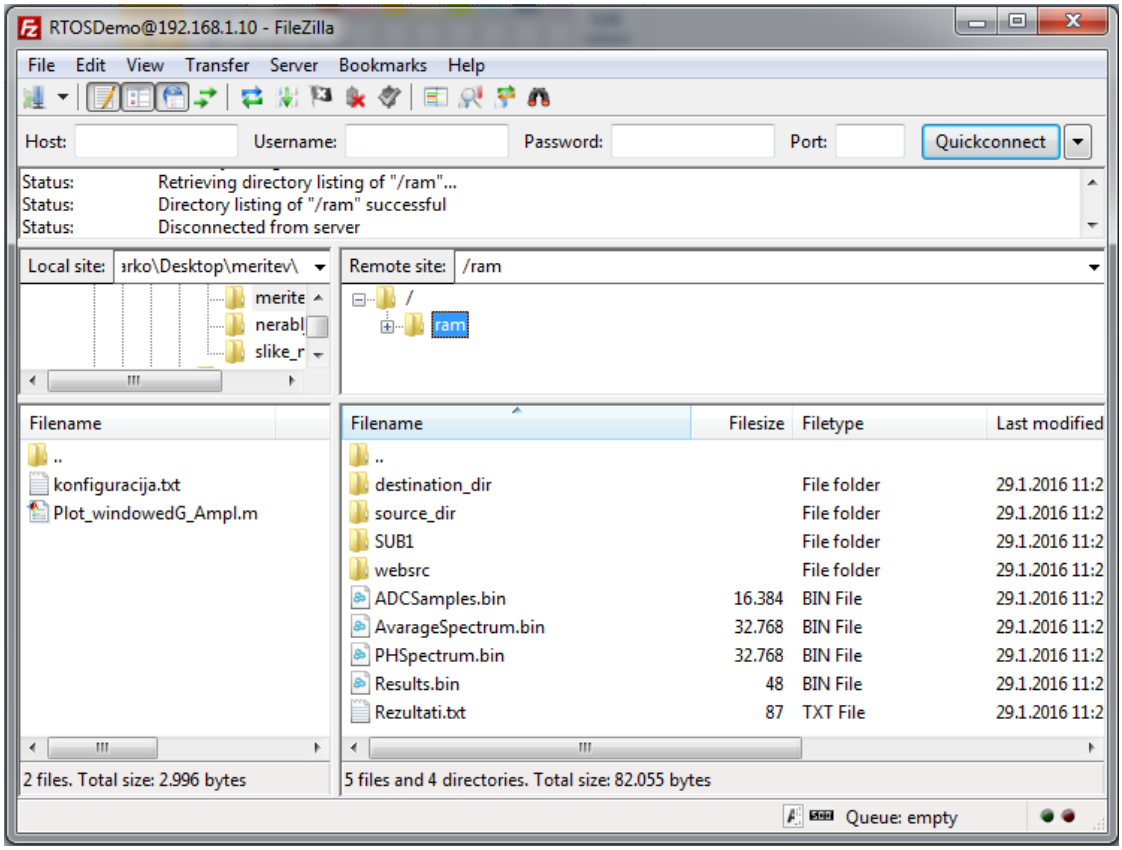
- ADCSamples.bin – vsebuje vzorce meritve.
- AvarageSpectrum.bin – vsebuje povprečen enostranski spekter vhodnega signala
- PHSpectrum.bin – vsebuje Peak Hold spekter vhodnega signala
- Results.bin – vsebuje rezultate meritve zapisane v binarnem zapisu
- Rezultati.txt - vsebuje rezultate meritve zapisane v .txt datoteki
  - RMS: RMS vrednost vhodnega signala
  - RMS_LF: RMS vrednost v spodnjem frekvenčnem območju vhodnega signala
  - RMS_MF: RMS vrednost v srednjem frekvenčnem območju vhodnega signala
  - RMS_HF: RMS vrednost v visokem frekvenčnem območju vhodnega signala
  - Vpp: vrednost od vrha do vrha vhodnega signala

Prikaz datoteke Rezultati.txt:

```
RMS = 0.7380
RMS_LF = 0.7379
RMS_MF = 0.0081
RMS_HF = 0.0091
Vpp = 2.19
```

Za primerjavo izračunamo RMS vhodnega signala še v časovnem prostoru RMS = 0.737g. Po Parsevalovem teoremu (enačba 2.13) se mora moč – izračunana v frekvenčnem prostoru – ujemati z močjo izračunano v časovnem prostoru. Rezultata se ne ujemata popolnoma, kar je posledica zaokrožitvenih napak ter nepopolne amplitudne korekcije oknenja.

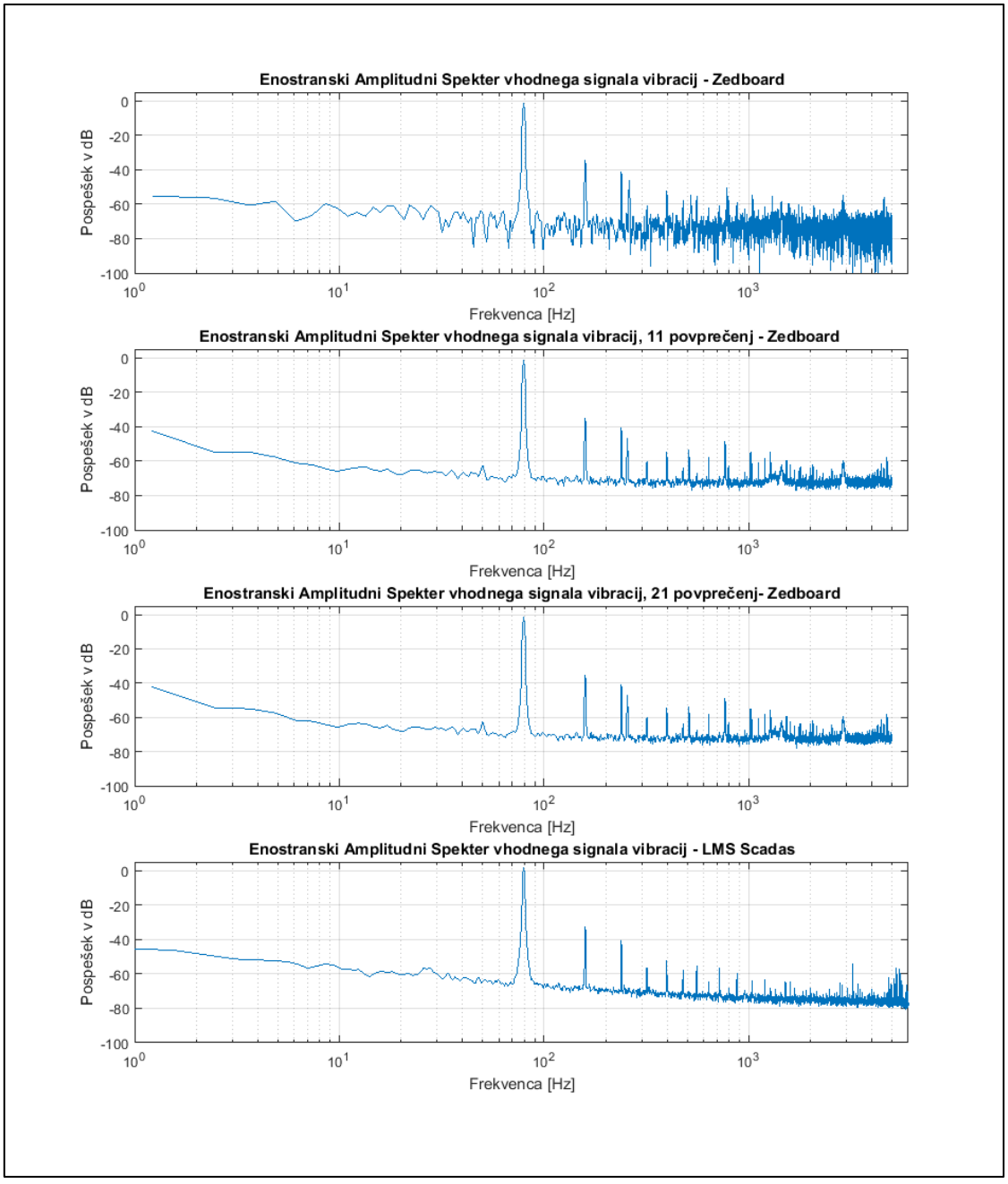
Zgoraj so izpisani rezultati, izračunani z Zedbordom. Razvidno je, da je RMS vrednost vhodnega signala 0.724 g ter, da se vsa moč nahaja v spodnjem frekvenčnem pasu. To je tudi pričakovano, saj ima vhodni signal frekvenco 80 Hz. Vrednost 0.724 g ustreza vhodnemu signalu amplitude 1g.



Slika 5.7: Prikaz dostopa do rezultatov s programom FileZilla

Na sliki 5.8 je prikazana primerjava meritve izdelanega merilnega sistema s profesionalnim merilnim sistemom LMS Scadas. Prvi graf prikazuje izračunan spekter brez uporabe povprečenja. Drugi prikazuje povprečen spekter 11 spektrov s 50 % prekrivanjem. Tretji prikazuje povprečen spekter 21 spektrov, prav tako s 50 % prekrivanjem. Četrti, zadnji pa prikazuje izračunan spekter z inštrumenti LMS Scadas, povprečje je 21 spektrov.

Kot je razvidno, lahko z uporabo povprečenja dosežemo, da so posamezne frekvenčne komponente bolje vidne ter zmanjšamo šum. Zavedati pa se moramo, da mora biti vhodni signal ob uporabi povprečenja stacionaren. Iz rezultatov se vidi, da smo se kar približali meritvi, ki je bila izmerjena s profesionalno opremo. Pojavi se frekvenčna komponenta pri 255 Hz, ki jo s profesionalno opremo nismo izmerili, je pa 46 dB manjša od glavne komponente vhodnega signala. Sklepamo, da je izvor motnje povezan z uporabo neustreznega ožičenja. Uporabiti bi morali koaksialne kable ter BNC priključke.



Slika 5.8: Primerjava merilnega sistema z LMS Scadas

6 Sklep

Cilj naloge je bil razviti koncept merilnega sistema za meritev vibracij z uporabo razvojne plošče Zedboard. Primerjava meritve izdelanega merilnega sistema z opremo LMS Scadas je pokazala, da je meritev izdelanega merilnega sistema primerljiva in dovolj natančna za ovrednotenje vibracij elektromotorja. Za končno rešitev pa je priporočljivo narediti še nekaj nadgradenj, ki bi izboljšale kakovost meritve in uporabnost merilnega sistema, ter znižale ceno izdelka.

Najprej bi bilo potrebno uporabiti koaksialne kable z BNC priključki za zmanjšanje motenj. Signal vibracij je v osnovi bipolaren, uporabili pa smo AD pretvornik, ki ima vhodno napetostno območje 0 – 10 V. Bolje bi bilo uporabiti takšnega, ki je v osnovi bipolaren z vhodnim napetostnim območjem -5 – 5 V. Ob uporabi takšnega AD pretvornika, bi se izognili uporabi dodatnega prilagoditvenega vezja, ki bipolarni vhodni signal prestavi navzgor za 5 V . Vseeno pa bi potrebovali filter proti prekrivanju (ang. anti aliasing filter). Prav tako bi bilo zaželeno imeti več kanalov, kar bi lahko rešili tako, da bi priključili več AD pretvornikov ali enega samega, večkanalnega. V tem primeru bi bolj prišla do izraza možnost paralelne obdelave podatkov v FPGA delu Zynq SoC. Samo programsko opremo bi bilo mogoče še precej optimizirati. V primeru, da bi aplikacija zahtevala nenehno zajemanje vhodnega signala, bi za izračun in ovrednotenje spektra ustvarili novo opravilo, ki bi se izvajalo vzporedno z zajemanjem signala. Za izvedbo merilnega sistema bi lahko uporabili razvojno ploščo Zybo, cenejšo od Zedboarda, hkrati pa več kot dovolj zmogljivo.

Zaključimo lahko, da Zynq SoC in uporaba razvojnega okolja Vivado s SDK, omogočata izvedbo projektov, kjer je zahtevana visoka računska zmogljivost ter prilagodljivost sistema v smislu priključitve periferije preko priključkov PMOD. Našteli smo nekaj možnih nadgradenj in s tem nam ostanejo izzivi za prihodnost.

Literatura

1 Maloberti, Franco. Data converters. Springer Science & Business Media, 2007.

2 S. Tomažič, S. Leonardis: Diskretni signali in sistemi, Ljubljana: Založba Fakultete za elektrotehniko, 2001.

3 Crockett, Louise H., et al. The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media, 2014.

4 FRESNO (MAXREFDES11#) [Online]. Dosegljivo: https://www.maximintegrated.com/en/design/reference-design-center/system-board/5563.html. [Dostopano: 15.5.2016]

5 FreeRTOS [Online]. Dosegljivo: http://www.freertos.org/RTOS.html. [Dostopano: 15.5.2016]

6 Fast Fourier Transform v9.0, LogiCORE IP Product Guide [Online]. Dosegljivo: http://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf. [Dostopano: 15.5.2016]

7 Heinzel Gerhard, Rüdiger Albrecht, Schilling Roland. Spectrum and spectral density estimation by the Discrete Fourier transform (DFT), including a comprehensive list of window functions and some new flat-top windows 2002. [Online] Doseglijivo: http://edoc.mpg.de/395068. [Dostopano: 15.5.2016]

8 Dennis H. Shreve, Signal processing for efective vibration analysis, IRD Mechanalysis, IncColumbus, Ohio, November 1995

9 National Instruments, "NI cDAQ-9184 [Online.]" Doseglijivo: http://sine.ni.com/nips/cds/view/p/lang/sl/nid/210669. [Dostopano: 15.5.2016]

10 National Instruments, "NI 9234 [Online]." Doseglijivo: http://sine.ni.com/nips/cds/view/p/lang/sl/nid/208802. [Dostopano: 15.5.2016]