UNIVERZITET UNION,
RAČUNARSKI FAKULTET,
BEOGRAD

DIPLOMSKI RAD

Tema: **Očitavanje analognih signala i njihova obrada u digitalnom formatu pomoću ESP32 mikrokontrolera**

Mentor:
Prof. dr Babić Djordje

Student:
Pavle Vukicevic

Sadržaj

1. Uvod

1.1 Karakteristike i ograničenja analognih signala

Da bismo razumeli probleme sa kojima se suočavamo pri obradi analognih signala, prvo je potrebno definisati šta su to analogni signali i kako se oni razlikuju od digitalnih. Signali se generalno dele u dve glavne kategorije: digitalni i analogni. Digitalni signal predstavlja nivo napona koji ima diskretna stanja, najčešće dva — logičku nulu (0) i logičku jedinicu (1). Ovaj tip signala se koristi još od samih početaka računarskih sistema i ostao je dominantan do danas zbog nekoliko ključnih razloga. Pre svega, digitalna komunikacija je otpornija na šum, jer je jednostavnije napraviti razliku između dve jasno definisane vrednosti (npr. 0V i 5V). Pored toga, digitalni signali omogućavaju bržu obradu i smanjuju potrošnju energije procesora. Za razliku od digitalnih, analogni signali su kontinuirani i mogu imati beskonačno mnogo vrednosti unutar određenog opsega. Na primer, dok digitalni signal može biti samo 0V ili 5V, analogni signal može imati vrednost od 1.37V, 2.89V, ili bilo koju drugu između minimalne i maksimalne granice. Ova kontinuirana priroda analognih signala donosi značajne izazove, a jedan od najvećih problema je šum.

Šum predstavlja neželjeni signal koji se meša sa glavnim signalom koji želimo da očitamo ili obradimo. Kao primer, možemo posmatrati razgovor na bučnom mestu — naš glas je glavni signal, dok buka iz okoline predstavlja šum. U elektronici, šum može dolaziti iz različitih izvora kao što su elektromagnetne smetnje, fluktuacije napona, i različiti talasi koji otežavaju precizno očitavanje analognog signala. Jedan od faktora koji dodatno komplikuje obradu analognih signala je **interferencija**. Interferencija se javlja zbog spoljašnjih smetnji koje potiču od drugih elektronskih uređaja kao što su punjači, ruteri, ili čak kablovi koji nose naizmeničnu struju. Ove smetnje mogu izazvati dodatne probleme pri očitavanju analognog signala, što zahteva upotrebu odgovarajućih filtera kako bi se uklonio šum.

Kada želimo da očitamo fizičku veličinu putem senzora, moramo odrediti koliko često ćemo očitati signal, što nazivamo **frekvencijom odabiranja**. Na primer, za merenje temperature okoline nije neophodno očitavati vrednost češće od svakih 15 minuta, dok senzor težine u digitalnoj vagi treba očitavati nekoliko puta u sekundi kako bi odmah pokazao promene u težini. Prikazivanje signala se obično vrši pomoću grafikona, gde je x-osa vreme, a y-osa predstavlja neku veličinu, često napon. U realnom svetu, analogna veličina može imati beskonačno mnogo vrednosti, ali da bismo ih zapisali u digitalni sistem, potrebno je **kvantizovati** te vrednosti, odnosno zaokružiti ih na određeni broj mogućih vrednosti. Ova procedura se naziva kvantizacija, i njoj ćemo se detaljnije posvetiti u poglavlju o ADC konverterima. Ovim smo definisali osnovne probleme sa kojima se suočavamo pri obradi analognih signala i pripremili teren za dalju diskusiju o rešenjima koja se koriste u praksi.

1.2 Pristup ograničenjima analognih signala

Sada kada smo objasnili osnovne pojmove vezane za analogne signale, možemo preći na konkretne probleme i načine njihovog rešavanja. Prvi korak u radu sa analognim signalima je njihovo pretvaranje u digitalni format, a to se postiže pomoću analogno-digitalnog konvertera (ADC). ADC modul može biti deo mikrokontrolera ili zasebno integrisano kolo. Kada je ADC integrisan unutar mikrokontrolera, njegov broj bita (tj. preciznost) i broj dostupnih analognih kanala su unapred određeni. Ovo može biti ograničavajuće kada nam je potrebna visoka preciznost ili veliki broj kanala. S druge strane, eksterni ADC pruža veću fleksibilnost, omogućavajući nam da izaberemo tačno one specifikacije koje su nam potrebne za određeni projekat. Ipak, eksterni ADC može dodati složenost dizajnu i povećati troškove. Na primer, mikrokontroler ATmega328p koristi 10-bitni ADC, što znači da može pretvoriti analogne vrednosti u digitalne brojeve između 0 i 1023. Ako nam je potrebna veća preciznost, možemo se odlučiti za eksterni ADC sa višim brojem bita, npr. 12-bitni ili 16-bitni konverter. Kada jednom pretvorimo analogni signal u digitalni format, možemo primetiti da signal sadrži šum. Šum se javlja na različitim frekvencijama, i u zavisnosti od izvora šuma, može biti stalni (kao šum od 60Hz iz električne mreže) ili impulsni (kao smetnje koje uzrokuju prekidači za svetlo). Da bismo smanjili šum, signal se filtrira pomoću odgovarajućih filtera.

Postoje dva načina implementacije filtera: analogni i digitalni. Analogni filteri koriste komponente poput otpornika i kondenzatora da uklone neželjene frekvencije pre nego što signal uopšte stigne do ADC-a. Digitalni filteri, s druge strane, obrađuju signal nakon što je on digitalizovan, što omogućava veću fleksibilnost i prilagodljivost u filtriranju različitih tipova šuma. U ovom radu ćemo se fokusirati na digitalne filtere, analizirati njihove karakteristike i načine implementacije na mikrokontroleru kao što je ESP32.

1.3 Hardver potreban za projekat

Za realizaciju ovog projekta korišćen je ESP32-WROOM-32 razvojni modul, koji integriše ESP32-D0WDQ6 čip kao glavni mikrokontroler. Ovaj razvojna ploča predstavlja jednostavno rešenje za prototipiranje jer nudi sve potrebne konektore i interfejse, dok sam ESP32-D0WDQ6 čip omogućava visok nivo performansi za obradu analognih signala u realnom vremenu. ESP32-D0WDQ6 čip poseduje dva nezavisna procesorska jezgra, što omogućava istovremeno izvršavanje zadataka i efikasnu obradu kompleksnih algoritama, poput filtriranja digitalnih signala. Ova jezgra mogu raditi na frekvencijama od 80 MHz do 240 MHz, pružajući fleksibilnost u optimizaciji performansi u zavisnosti od zahteva aplikacije.

Jedna od ključnih prednosti korišćenja ESP32 platforme je integrisani WiFi, Bluetooth, i Bluetooth Low Energy (BLE), koji omogućavaju bežičnu komunikaciju i lako povezivanje sa drugim uređajima ili serverima. Ove karakteristike su posebno korisne u IoT projektima gde je često potrebno prenositi podatke do udaljenih sistema ili aplikacija u realnom vremenu.

Za ovaj projekat, odabran je senzor težine kao primer izvora analognog signala. Senzor težine koristi strain gauge tehnologiju za merenje mehaničkog opterećenja pomoću specijalnih otpornika i generisanje analognog signala koji se zatim digitalizuje pomoću ADC modula na ESP32. Iako je senzor težine korišćen u ovom primeru, opisane metode filtriranja signala mogu se primeniti i na druge senzore, poput senzora temperature, svetlosti, ili pritiska.

1.4 Softver potreban za projekat

Softverska strana projekta oslanja se na korišćenje MATLAB-a za analizu i dizajn filtera, kao i na PlatformIO razvojno okruženje za implementaciju na ESP32. MATLAB je alat koji omogućava efikasno testiranje i simulaciju digitalnih filtera pre njihove stvarne implementacije. Koristeći MATLAB, možemo dizajnirati različite tipove filtera (niskopropusne, visokopropusne, opsegopropusne) i simulirati njihovo ponašanje. Na ovaj način možemo brzo identifikovati optimalne parametre, kao što su granične frekvencije i koeficijenti filtera, koji se zatim koriste u implementaciji na mikrokontroleru.

Nakon dizajna i analize filtera, PlatformIO razvojno okruženje omogućava jednostavno programiranje ESP32 razvojne ploče koristeći C/C++ jezike u kombinaciji sa Arduino API-jem. Ovakav pristup ubrzava razvoj i olakšava integraciju sa raznim senzorima i modulima.

Korišćenje MATLAB-a za dizajn filtera i PlatformIO za implementaciju osigurava fleksibilan i skalabilan proces razvoja. U daljim poglavljima biće objašnjeno kako se koeficijenti filtera generisani u MATLAB-u koriste za implementaciju digitalnog filtera na ESP32, kao i kako se signal efikasno filtrira u realnom vremenu koristeći dostupne resurse mikrokontrolera.

2. Arhitektura mikrokontrolera ESP32

2.1 Uvod u arhitekturu

ESP32 je snažan i brz mikrokontroler koji spada u kategoriju sistema na čipu (SoC). To znači da unutar jednog integrisanog kola kombinuje više različitih funkcionalnosti, što ga čini veoma atraktivnim rešenjem za različite projekte, od jednostavnih senzorskih aplikacija do kompleksnih IoT sistema. Jedna od ključnih prednosti ESP32 mikrokontrolera jeste integrisani hardver za WiFi i Bluetooth komunikaciju. Ovo ga razlikuje od mnogih drugih mikrokontrolera, koji često zahtevaju dodatne module za bežične komunikacije, povećavajući kompleksnost i cenu projekata.

Integracija WiFi i Bluetooth tehnologije direktno na čip pojednostavljuje dizajn sistema i smanjuje potrebu za dodatnim komponentama, što omogućava lakše prototipiranje i bržu implementaciju. Na primer, mikrokontroleri kao što su ATmega328 (koristi se u Arduino Uno) ili STM32 serije, često zahtevaju dodatne bežične module kao što su ESP8266 ili HC-05, što može predstavljati izazov u projektima gde su prostorna ograničenja i potrošnja energije ključni faktori. Međutim, ovakva arhitektura ima i svoje nedostatke. ESP32 zbog dodatnih funkcionalnosti između ostalog troši više energije u odnosu na pomenute mikrokontrolere. Zbog toga je važno razmotriti potrošnju energije kada se koristi u projektima koji zahtevaju rad na baterije ili energetski efikasna rešenja.

ESP32-D0WDQ6 je deo šire porodice ESP32 mikrokontrolera, koja uključuje različite module i verzije optimizovane za različite upotrebe. U ovom projektu koristi se ESP32-WROOM-32 razvojni modul, koji u svojoj osnovi ima ESP32-D0WDQ6 čip. Ovaj čip poseduje dva mikroprocesora Xtensa 32-bit LX6 što omogućava paralelnu obradu zadataka i efikasnije korišćenje resursa sistema. Svako jezgro se može podesiti da radi na frekvenciji od 80 MHz do 240 MHz, čime se pruža fleksibilnost u prilagođavanju performansi i potrošnje energije.

Glavna procesorska jezgra u ESP32-D0WDQ6 su bazirana na Xtensa LX6 arhitekturi, razvijenoj od strane Tensilica (sada deo Cadence Design Systems). Xtensa jezgra nude visoke performanse uz optimizovanu potrošnju energije i podržavaju specifične ekstenzije za obradu digitalnih signala, što je ključno za aplikacije koje uključuju filtriranje i analizu signala. S obzirom na to da se u ovom projektu obrađuju analogni signali sa senzora, arhitektura Xtensa omogućava optimizovanu implementaciju algoritama za obradu signala, kao što su digitalni filtri, koristeći **hardverske akceleratore** i specijalizovane instrukcije. Ovo značajno smanjuje vreme obrade i povećava efikasnost celokupnog sistema.

ESP32-D0WDQ6 procesor, kao deo ESP32-WROOM-32 modula, zasnovan je na Xtensa LX6 arhitekturi, koja nudi brojne prednosti za obradu signala i druge zahtevne zadatke. Neke od ključnih karakteristika koje doprinose njegovim visokim performansama uključuju:

1. 7-stage pipeline: Procesor koristi sedmostepeni protokol (pipeline), koji omogućava izvršavanje više instrukcija istovremeno. Ovaj pipeline povećava efikasnost procesora i omogućava rad na visokim frekvencijama do 240 MHz. Ovakva arhitektura značajno smanjuje vreme potrebno za izvršenje instrukcija, posebno u aplikacijama koje zahtevaju brzu obradu podataka.
2. 16/24-bitni instrukcioni set: Upotreba mešanih veličina instrukcija (16-bitnih i 24-bitnih) povećava efikasnost memorije i gustinu koda. Ovo omogućava optimalno korišćenje dostupne memorije, smanjuje prostor koji zauzimaju instrukcije i poboljšava performanse aplikacija, što je korisno za kompleksne algoritme kao što su oni za obradu signala.
3. Floating Point Unit (FPU): Jedinica za rad sa brojevima u pokretnom zarezu omogućava efikasne izračune sa realnim brojevima, što je ključno za precizne matematičke operacije. FPU omogućava brzo i precizno izvršavanje operacija kao što su množenje i deljenje brojeva u pokretnom zarezu, što je od velike važnosti u aplikacijama kao što su filtriranje signala ili analize podataka sa senzora.
4. DSP instrukcije: Procesor podržava specijalizovane instrukcije za digitalnu obradu signala (DSP), uključujući operacije množenja i sabiranja (MAC - Multiply-Accumulate). Ove instrukcije ubrzavaju ključne operacije u aplikacijama koje uključuju obradu audio signala, analizu podataka sa senzora i druge zadatke koji zahtevaju brzu obradu velikih količina podataka.
5. Rukovanje prekidima (Interrupt Handling): Sistem za rukovanje prekidima omogućava procesoru da efikasno upravlja više zadataka visokog prioriteta. Organizovani sistem prekida čini mikrokontroler veoma odzivnim na spoljne događaje, omogućavajući brzu reakciju na promene u okruženju, kao što su promene u očitavanju senzora ili ulaznih signala.

Ove karakteristike omogućavaju ESP32 mikrokontroleru da efikasno izvršava kompleksne algoritme za obradu signala, uključujući digitalne filtre i druge metode analize podataka. Ovo ga čini idealnim za aplikacije u realnom vremenu gde je potrebna visoka preciznost i brzina obrade.

2.2 Energetska efikasnost i potrošnja energije u ESP32

Obrada digitalnih signala na mikrokontroleru kao što je ESP32 zahteva specifičnu raspodelu energije kako bi se postigli optimalni rezultati. Kada se mikrokontroler koristi za zadatke kao što su filtriranje signala i obrada podataka sa senzora, važno je razumeti kako se energetska potrošnja deli između različitih komponenti. Na primer, energetski zahtevni zadaci poput prikupljanja podataka putem ADC modula i njihova dalja obrada u procesoru zahtevaju više energije u odnosu na zadatke kao što su jednostavna digitalna logika ili periodično buđenje iz stanja niske potrošnje energije.

Vazno je napomenuti da kod obrade signala, ovaj procesor često koristi DSP instrukcije i Floating Point Unit (FPU) za brze i precizne matematičke operacije. Ove funkcionalnosti povećavaju potrošnju energije, ali su ključne za aplikacije koje zahtevaju obradu visokofrekventnih signala, filtriranje šuma i druge analize podataka u realnom vremenu. Zbog toga je bitno pažljivo upravljati energetskim režimima mikrokontrolera kako bi se minimizirala potrošnja kada obrada signala nije u toku, a u isto vreme omogućila maksimalna efikasnost kada je potrebna brza analiza.

2.2.1 Energetski modovi rada

ESP32 poseduje nekoliko različitih režima potrošnje energije, što omogućava prilagođavanje potrošnje u zavisnosti od zadatka koji se obavlja:

- Active mode (Aktivni režim): U ovom režimu radio uređaj je uključen, što znači da čip može da prima, prenosi ili sluša podatke putem Wi-Fi ili Bluetooth-a. Ovo je režim sa najvećom potrošnjom energije jer su svi procesi aktivni.
- Modem-sleep mode (Modem-sleep režim): Procesor je operativan i frekvencija takta se može podešavati, dok su Wi-Fi/Bluetooth radio isključeni. Ovaj režim se koristi kada je procesor i dalje aktivan, ali nema potrebe za bežičnom komunikacijom.
- Light-sleep mode (Light-sleep režim): Procesor je pauziran, ali RTC (Real-Time Clock) memorija, RTC periferije i ULP (Ultra-Low Power) koprocesor ostaju aktivni. U slučaju da se desi neki događaj za buđenje (RTC tajmer ili eksterni prekidi), čip će se probuditi iz ovog režima.
- Deep-sleep mode (Deep-sleep režim): U ovom režimu samo RTC memorija i RTC periferije rade. Podaci o Wi-Fi i Bluetooth konekcijama se čuvaju u RTC memoriji. ULP koprocesor je funkcionalan, ali ostatak sistema je u potpunosti isključen, čime se značajno smanjuje potrošnja energije.
- Hibernation mode (Hibernacija): Interni oscilator od 8 MHz i ULP koprocesor su isključeni. RTC memorija za oporavak je takođe isključena. Aktivni su samo jedan RTC tajmer na sporom taktu i određeni RTC GPIO pinovi. Čip može biti probuđen iz hibernacije samo pomoću RTC tajmera ili aktivacijom RTC GPIO pinova.

Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 5-4 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered up.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered up.			150 μA
	ULP sensor-monitored pattern			100 μA @1% duty
	RTC timer + RTC memory			10 μA
Hibernation	RTC timer only			5 μA
Power off	CHIP_PU is set to low level, the chip is powered down.			1 μA

Slika 1. Modovi rada prikazani tabelarno — leva kolona sadrži naziv režima, srednja opisuje aktivne module i njihove način rada, dok desna prikazuje potrošnju struje u svakom režimu.

2.3 ADC modul

ESP32 mikrokontroler ima ugrađeni Analogno-Digitalni pretvarac (ADC), koji omogućava da se analogni signali pretvore u digitalne vrednosti i na taj način budu

obradivi od strane procesora. ADC je jedna od ključnih komponenti u sistemima za očitavanje analognih signala senzora. ESP32 poseduje dva ADC modula: ADC1 i ADC2, svaki sa po nekoliko analognih ulaza:
- ADC1 podržava do 8 kanala (GPIO 32 do GPIO 39).
- ADC2 podržava do 10 kanala (GPIO 0, GPIO 2, GPIO 4, GPIO 12 do GPIO 15, GPIO 25 do GPIO 27).

Svaki od ovih modula je 12-bitni ADC, što znači da može da pretvori analogni napon u digitalnu vrednost u opsegu od 0 do 4095. Što je veći broj bita, to je preciznija konverzija, pa se ovih 12 bita često smatra zadovoljavajućim za većinu aplikacija koje zahtevaju srednju preciznost. Možemo se podsetiti da ovih 12 bita predstavlja "jačinu" prethodno pomenute **kvantizacije**.

Jedna od stvari koju treba napomenuti je da ADC u ESP32 mikrokontroleru radi u opsegu napona od 0 do 3.3V, što je u skladu sa naponom napajanja. Ovo znači da ulazni napon koji prelazi 3.3V može izazvati oštećenje mikrokontrolera, pa se zato preporučuje korišćenje delitelja napona ili operacionih pojačavača kao zaštitu, ako se meri viši napon.

2.3.1 Referentni napon

ESP32 koristi referentni napon (Vref) od oko 1.1V za merenje analognih signala preko svojih ADC-ova, ali to ograničava maksimalni merni opseg na 1.1V. Da bi se omogućilo merenje viših napona, ESP32 koristi atenuaciju, koja smanjuje ulazni napon pre nego što stigne do ADC-a. Postoje četiri opcije atenuacije: 0dB , 2.5dB , 6dB i 11dB (slika ispod..). Atenuacija omogućava precizno merenje širih opsega napona, čime se omogućava korišćenje ADC-a za napone bliske naponu napajanja od 3.3V.

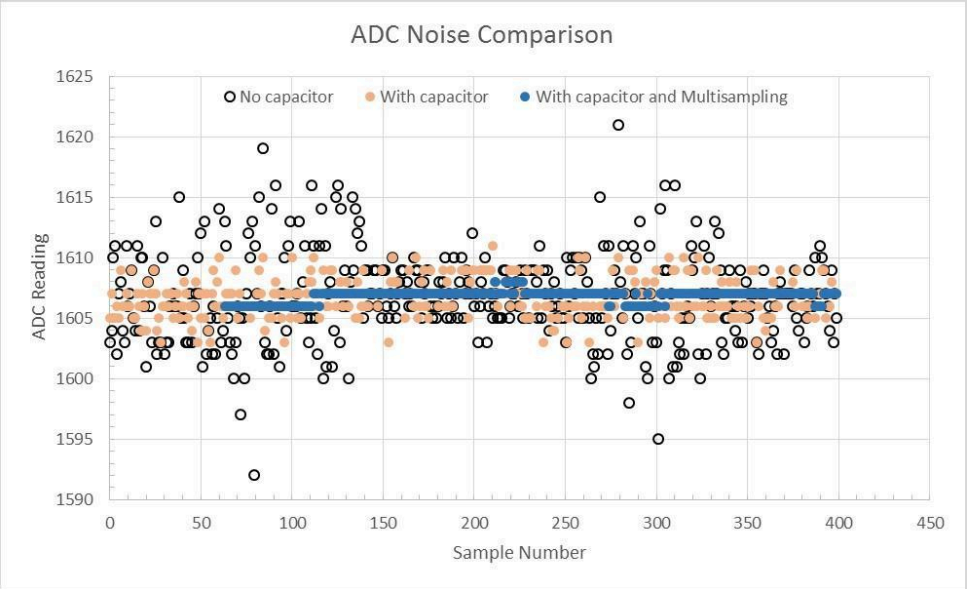
Ranije, preciznost ADC-a na ESP32 nije bila njegova jača strana, prvenstveno zbog internog Vref napona koji se kreće oko 1.1V. Problem je što ova vrednost varira između 1.0V i 1.2V zavisno od modula, što dovodi do nepreciznosti. Pri korišćenju atenuacije od -11 dB, koja omogućava merenje do 3.3V, dolazi do nelinearnosti u odgovoru ADC-a, što znači da merenja nisu pouzdana. Ipak, od 2019. godine, ESP32 mikrokontroleri dolaze fabrički kalibrisani, a dodat je i softverski popravak za poboljšanje preciznosti. Kod novijih modela, poput ESP32-S3, problem je dodatno rešen korišćenjem internog čipa za hardversku kalibraciju, što značajno poboljšava tačnost merenja.

2.3.2 Elektronski šum u sistemu ESP32

Attenuation	Measurable input voltage range
ADC_ATTEN_DB_0	100 mV ~ 950 mV
ADC_ATTEN_DB_2_5	100 mV ~ 1250 mV
ADC_ATTEN_DB_6	150 mV ~ 1750 mV
ADC_ATTEN_DB_11	150 mV ~ 2450 mV

Slika 2. Različiti nivoi atenuacije i njima srazmerni naponi

ADC na ESP32 mikrokontroleru može biti podložan šumovima iz okruženja, što često dovodi do nepravilnih i varirajućih očitavanja, naročito kada radimo sa niskonaponskim signalima. Ovaj problem se manifestuje kao velika nesigurnost u vrednostima koje ADC prijavljuje, što može značajno uticati na preciznost merenja. Da bi se smanjio uticaj šuma, preporučuje se povezivanje keramičkog kondenzatora od 100 nF direktno na ulazni pin ADC-a. Ovaj kondenzator funkcioniše kao "bypass" kondenzator, filtrirajući visokofrekventni šum i stabilizujući signal pre nego što uđe u ADC. Pored ovog fizičkog rešenja, moguće je primeniti i softversku tehniku višestrukog uzorkovanja (multisampling). Ova metoda podrazumeva uzimanje više uzoraka za isti signal i potom računanje prosečne vrednosti tih uzoraka, što dodatno pomaže u eliminaciji šuma i osigurava preciznija očitavanja. Korišćenjem oba pristupa, moguće je značajno poboljšati kvalitet i pouzdanost merenja analognog signala na ESP32.



Slika 3. prikaz smanjenja šuma korišćenjem keramičkog kondenzatora i tehnikom višestrukog uzorkovanja (multisampling) sa 64 uzorka po merenju

Ovakva odstupanja se mogu ispraviti softverskom kalibracijom ADC-a. Na ovaj način će se ispraviti većina nepravilnosti i dobiti tacnije informacije. ADC pretvarač u ESP32 je izuzetno koristan alat za projekte koji zahtevaju očitavanje analognih signala. Iako postoje određeni izazovi sa šumom i preciznošću, uz pravilno podešavanje i kalibraciju, može se postići zadovoljavajući nivo tačnosti. Zahvaljujući velikom broju kanala i fleksibilnosti u podešavanju, ADC u ESP32 je idealan za širok spektar aplikacija koje uključuju obradu signala.

2.4 DAC modul

ESP32 mikrokontroler pored ADC modula poseduje i Digitalno-Analogni konverter (DAC). Ovaj modul omogućava pretvaranje digitalne vrednosti (niza binarnih brojeva) u odgovarajući analogni napon. U praksi, to znači da možemo generisati analogne signale direktno sa mikrokontrolera bez potrebe za eksternim DAC čipom. ESP32 ima dva DAC izlaza koji su povezani na pin 25 i pin 26. Oni omogućavaju izlaz napona u opsegu od 0 do 3.3V, zavisno od vrednosti koju zadamo. Ovaj DAC modul radi sa rezolucijom od 8 bita, što znači da digitalne vrednosti koje možemo uneti u DAC kreću se u rasponu od 0 do 255. Vrednost 0 generiše napon od 0V, dok vrednost 255 generiše maksimalni napon, koji je jednak napajanju mikrokontrolera, odnosno približno 3.3V.

Korisnici često koriste DAC za generisanje jednostavnih zvukova, simulaciju senzorskih izlaza. Pored toga, ovaj modul se može koristiti i za testiranje raznih analognih kola u sistemima gde je potrebno kontrolisati napon putem softvera. Iako ESP32 nudi funkcionalan DAC modul, treba napomenuti da on nije namenjen za visokoprecizne aplikacije. DAC na ESP32 ima određene limitacije kada je u pitanju linearnost i preciznost izlaznog signala, što znači da može doći do manjih odstupanja u naponskim vrednostima u poređenju sa idealnim izlazom. Ipak, za mnoge praktične primene, ove male varijacije ne predstavljaju problem i DAC modul se može koristiti sa zadovoljavajućim rezultatima.

DAC modul omogućava da nakon što obradimo signal digitalno, možemo da ga pretvorimo nazad u analogni oblik i na taj način izlazni signal postane kompatibilan sa uređajima koji očekuju analogne ulaze. Na primer:
- Obrada audio signala - Ako se radi obrada zvuka na mikrokontroleru (npr. dodavanje efekata, filtriranje), DAC modul može da pretvori obrađeni digitalni signal u analogni zvuk koji se zatim može pustiti kroz zvučnik.
- Testiranje analognih kola - Digitalno generisani talasi (kao što su sinusoidni, trougaoni ili kvadratni talasi) mogu se konvertovati u analogne signale putem DAC-a i koristiti za testiranje odgovora različitih analognih kola kao što su filteri ili pojačala.

U suštini, DAC nije potreban za ostvarenje ovog projekta, jer se svi podaci obrađuju digitalno i u većini slučajeva šalju dalje bez potrebe za njihovim pretvaranjem u analogni oblik. Ipak, pomenuli smo mogućnost njegove upotrebe, ukoliko bi bilo potrebno proslediti audio signal ili testirati funkcionalnost signala na drugim analognim uređajima.

3. Osnove digitalne obrade signala

U prethodnim poglavljima osvrnuli smo se na osnovne karakteristike ESP32 mikrokontrolera i njegove module, pominjući kako oni omogućavaju očitavanje i obradu analognih signala. Sada prelazimo na glavni deo rada – digitalnu obradu signala. Ovo poglavlje će se fokusirati na ključne koncepte i tehnike koje omogućavaju analizu i filtriranje signala, što je od presudnog značaja za mnoge inženjerske aplikacije. Poseban akcenat stavićemo na transformacije signala u frekvencijskom domenu, razliku između tipova filtara i njihove primene, čime ćemo postaviti temelje za praktičnu implementaciju u narednom poglavlju.

3.1 Furijeova transformacija

Za Furijeovu transformaciju se kaze da je najvazniji algoritam u svetu. Da bismo uopšte razumeli značaj Furijeove transformacije, moramo se vratiti na problem koji je doveo do njenog razvoja. Jožef Furije je 1822. godine postavio temelje za ovu metodu kada je istraživao kako opisati složene funkcije koje se pojavljuju u analizi toplote. On je pokazao da se određene funkcije mogu predstaviti kao beskonačna suma **sinusnih** i **kosinusnih** talasa, poznatih kao harmonici. Ova ideja je bila revolucionarna, jer je omogućila prelazak sa složenih funkcija u vremenskom domenu na jednostavnije komponente u frekvencijskom domenu.
Furijeova teorija je rešila mnoge probleme u analizi periodičnih signala i postala osnovni alat u nauci i inženjerstvu. U digitalnoj eri, njen koncept je prilagođen diskretnim podacima, što je dovelo do Diskretne Furijeove transformacije (**DFT**). DFT pruža mogućnost analize digitalnih signala u frekvencijskom domenu, omogućavajući nam da identifikujemo dominantne frekvencije ili neželjene šumove. Upravo zbog toga, DFT i njena efikasna verzija, Brza Furijeova transformacija (**FFT**), imaju ključnu ulogu u obradi signala u projektima poput ovog.

3.1.1 Analogija sa sokom

Zamislimo čašu soka na stolu. Ono što vidimo i pijemo deluje kao jedna homogena tečnost, ali mi znamo da taj sok nije samo voda ili šećer – on je kombinacija različitih sastojaka poput pomorandže, limuna, jagoda ili možda čak malo đumbira. Sada, zamislimo da želimo da saznamo tačan recept za taj sok.

Kako to da uradimo? Kada bismo imali specijalnu cediljku koja može da “procedi” sok na način da iz njega izvuče sve sastojke pojedinačno, znali bismo tačno šta je u njemu. Na primer, kroz jednu cediljku bismo izvukli samo sok od pomorandže, kroz drugu samo limun, a treća bi možda odvojila jagode. Svaka od ovih cediljki bi nam dala tačne proporcije sastojaka koje čine naš originalni napitak. Na kraju bismo dobili

ne samo informacije o svim sastojcima, već i o njihovim količinama – odnosno, dobili bismo recept.

Ova ideja možda zvuči kao naučna fantastika kada govorimo o fizičkom soku, ali u svetu signala imamo matematičku alatku koja radi upravo ovo – Furijeovu transformaciju. Ona je "cediljka" za signale i omogućava nam da razložimo kompleksni signal na njegove osnovne sastojke, odnosno frekvencije.

3.1.2 Furijeova transformacija i "sastojci" signala

Kada radimo sa signalima, bilo da su to zvučni zapisi, medicinski signali poput EKG-a ili EEG-a, ili čak elektromagnetni talasi, ono što dobijamo kao ulaz je često kompleksan signal. Na primer, uzmimo snimak razgovora. Taj snimak nije samo glas osobe koja govori – tu su i pozadinski zvukovi poput buke automobila, šuma vetra ili možda zujanja električnih aparata. Sve te informacije su "pomešane" u jednom signalu, što ga čini teškim za analizu.

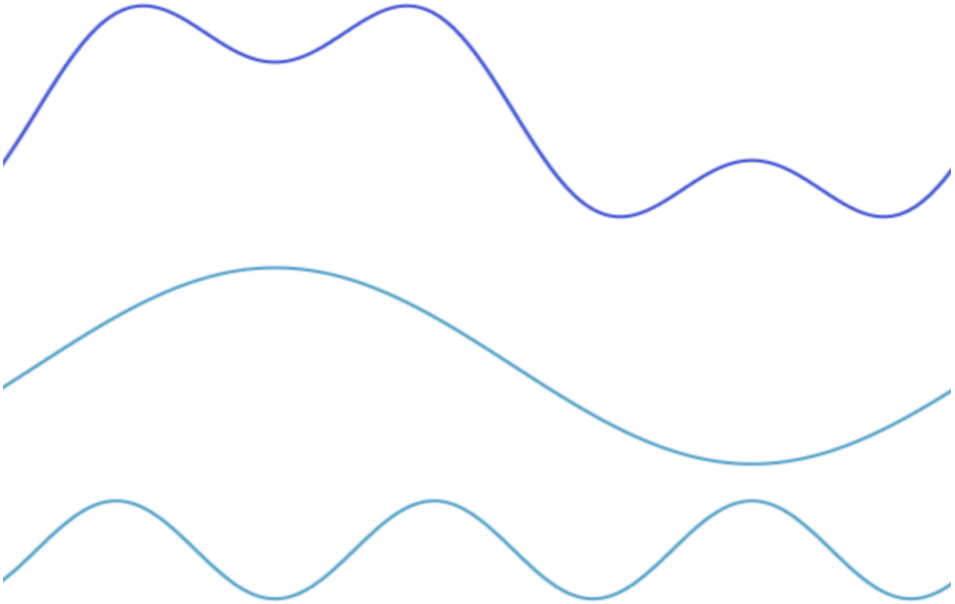
Furijeova transformacija dolazi kao rešenje jer nam omogućava da "procedimo" signal i razdvojimo ga na njegove osnovne frekvencije. Svaka frekvencija u signalu odgovara jednom od "sastojaka". Na primer:

- Glas osobe može odgovarati niskim frekvencijama.
- Buka automobila može se pojaviti u opsegu srednjih frekvencija.
- Zujanje aparata često je na visokim frekvencijama.

Kada primenimo Furijeovu transformaciju, mi dobijamo nešto što se zove frekvencijski domen – prikaz koji nam pokazuje sve sastojke signala i njihove jačine (amplitude). Na taj način možemo analizirati šta čini signal i, što je još važnije, možemo izolovati ono što nam treba i ukloniti ono što ne želimo.

Da bismo bolje razumeli, pogledajmo hipotetički primer. Zamislimo signal koji se sastoji od dve različite frekvencije. Prva frekvencija odgovara zvuku klavira, dok druga predstavlja zvuk flaute. Kada oba zvuka pustimo istovremeno, dobijamo složen signal koji u vremenskom domenu deluje kao jedna kontinuirana oscilacija. Međutim, primenom Furijeove transformacije, možemo razdvojiti ove dve frekvencije i identifikovati svaku ponaosob.

Ova mogućnost razlaganja kompleksnog signala na njegove osnovne komponente ključna je za mnoge primene u realnom svetu.



Slika 4. signal sastavljen od dve različite frekvencije (gore), dok su u donjem delu prikazane njegove pojedinačne frekvencijske komponente.

3.1.3 Diskretna Furijeova Transformacija: Teorija i Primena

Furijeova transformacija se zasniva na ideji da svaki signal može biti predstavljen kao suma sinusnih i kosinusnih talasa različitih frekvencija. Ovi talasi su osnovni gradivni blokovi signala, baš kao što su brašno, jaja i šećer osnovni sastojci torte. Kada primenimo Furijeovu transformaciju, mi zapravo "testiramo" signal za prisustvo različitih sinusnih talasa. Na kraju dobijamo spektar koji nam pokazuje koje frekvencije su prisutne i u kojoj meri doprinose originalnom signalu.

U digitalnoj obradi signala, radimo sa diskretnim signalima – signalima koji su uzorkovani u određenim vremenskim intervalima. Za ove signale koristimo Diskretnu Furijeovu transformaciju (DFT), koja nam daje frekvencijski spektar u digitalnom obliku. DFT funkcioniše tako što uzima određeni broj uzoraka signala i analizira ih koristeći matematičke formule. Rezultat je skup podataka koji opisuje sve frekvencije prisutne u signalu i njihove amplitude.

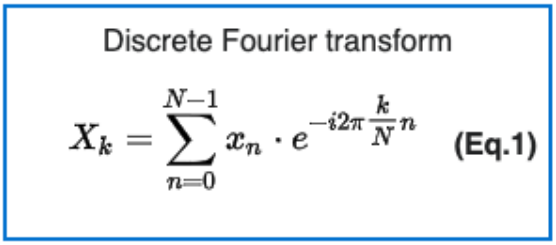
Do sada smo Furijeovu transformaciju posmatrali kroz analogije sa receptima za sokove ili torte kako bismo intuitivno razumeli njen princip – ideju o „razlaganju" kompleksnog signala na njegove osnovne frekvencije. Međutim, da bismo u potpunosti razumeli ovu temu i njenu praktičnu primenu, neophodno je da pređemo sa figurativnog objašnjenja na matematičku formulaciju. Matematika je ta koja nam

omogućava da precizno „izmerimo“ sastojke signala i predstavimo ih u frekvencijskom domenu.

U osnovi, Furijeova transformacija pruža način da svaki signal, koliko god bio složen, predstavimo kao zbir sinusnih i kosinusnih talasa različitih frekvencija. Ovi talasi predstavljaju osnovne „gradivne blokove“ od kojih je signal sačinjen. Matematička definicija Furijeove transformacije omogućava nam da kvantitativno odredimo koji talasi čine signal, kao i njihove amplitude i faze.

U ovom projektu se bavimo isključivo diskretnim signalima, jer se digitalna obrada signala oslanja na signale koji su uzorkovani u diskretnim vremenskim trenucima. Dakle, mi ćemo koristiti DFT za analizu signala. Razlog zašto koristimo DFT leži u prirodi računarskih sistema i mikrokontrolera, koji mogu obraditi samo diskretne vrednosti. Signali iz stvarnog sveta, kao što su zvučni signali, najpre se uzorkuju pomoću analogno-digitalnih konvertora (ADC), koji ih pretvaraju u niz diskretnih vrednosti. Nakon uzorkovanja, možemo primeniti DFT kako bismo dobili prikaz signala u frekvencijskom domenu.

Ključni simboli i njihovo značenje
Pre nego što pređemo na samu formulu DFT, objasnićemo njene glavne elemente kroz jednostavne definicije:
1. N : Ukupan broj uzoraka signala. Ovo predstavlja broj diskretnih vremenskih trenutaka u kojima smo izmerili signal.
2. n : Trenutni uzorak koji analiziramo. Vrednost n varira od 0 do N-1, gde svaki n odgovara određenom vremenskom trenutku.
3. x[n] : Vrednost signala u trenutku n. Ovo je amplituda signala (npr. nivo zvuka ili napona) u datom vremenskom uzorku.
4. k : Frekvencija koju trenutno analiziramo. Vrednost k takođe varira od 0 do N-1, gde svaki k predstavlja određenu frekvenciju signala, u rasponu od 0 Hz do maksimalne frekvencije definisane Nyquistovim teoremom.
5. X[k] : Vrednost koja predstavlja „količinu“ frekvencije k u signalu. Ovo je kompleksan broj koji sadrži informacije o amplitudi i fazi te frekvencije u originalnom signalu.



Slika 5. Formula diskretne furijerove transformacije

Diskretna Furijeova transformacija se zasniva na ideji da svaki signal može biti predstavljen kao suma sinusnih i kosinusnih talasa različitih frekvencija. DFT, kroz svoju formulu, računa koliki je doprinos svake frekvencije k u originalnom signalu x[n]. Matematički, formula za DFT izgleda ovako:

Iako formula za DFT deluje jednostavno (samo „sabiranje kompleksnih brojeva“), njena implementacija u programiranju često zahteva konverziju u pravougaoni

koordinatni sistem (kosinus i sinus komponenta). To je zato što mnogi programski jezici ne podržavaju rad sa kompleksnim brojevima direktno. U praksi se, dakle, svaka komponenta računa kao zbir realnih i imaginarnih delova:
• Realni deo: $\sum(n=0 \text{ do } N-1)\ x[n] * \cos(2\pi kn/N)$
• Imaginarni deo: $-\sum(n=0 \text{ do } N-1)\ x[n] * \sin(2\pi kn/N)$

Na ovaj način, rezultat X[k] dobija se kao kompleksni broj sa realnim i imaginarnim delovima, koji se kasnije mogu interpretirati u smislu amplitude i faze.

3.1.4 Brza Furijeova Transformacija (FFT)

Nakon što smo usvojili osnove diskretne Furijeove transformacije (DFT), možemo jednostavnije razumeti princip rada brže Furijeove transformacije (FFT). FFT, kako i samo ime sugeriše, predstavlja algoritam koji omogućava značajno ubrzanje računanja Furijeove transformacije. Ovaj algoritam koristi optimizovane matematičke postupke kako bi eliminisao suvišne ili redundantne proračune prisutne u DFT-u.

Za ilustraciju, možemo koristiti jednostavnu analogiju: zamislimo da želimo da izbrojimo ukupan broj polja u pravougaonikom rasporedu tabele. Jedan način bi bio da brojimo svako polje pojedinačno. Međutim, efikasniji pristup bio bi da pomnožimo broj redova i kolona tabele, čime bismo dobili isti rezultat uz manje koraka. Na sličan način, FFT koristi strukturu podataka i svojstva simetrije kako bi postigao značajno ubrzanje u odnosu na DFT.

Naravno, razlika između ova dva algoritma zahteva detaljno matematičko objašnjenje, koje uključuje analizu kompleksnosti algoritama i njihove implementacije. Međutim, kako bismo ostali u okviru ovog projekta, izbegavaćemo dublju matematičku razradu i fokusiraćemo se na praktične aspekte primene FFT-a.

3.1.5 Frekvencijski domen i osnovni pojmovi

Da bismo razmotrili kako filtrirati signal, potrebno je prvo razumeti ključne pojmove i principe koji stoje iza filtracije. Filtracija je proces kojim se određeni delovi frekvencijskog spektra signala pojačavaju, slabe ili potpuno uklanjaju, u zavisnosti od željenog rezultata. Osnovna ideja je prepoznati frekvencije koje su korisne i treba ih sačuvati (propustiti) i one koje treba smanjiti ili eliminisati.

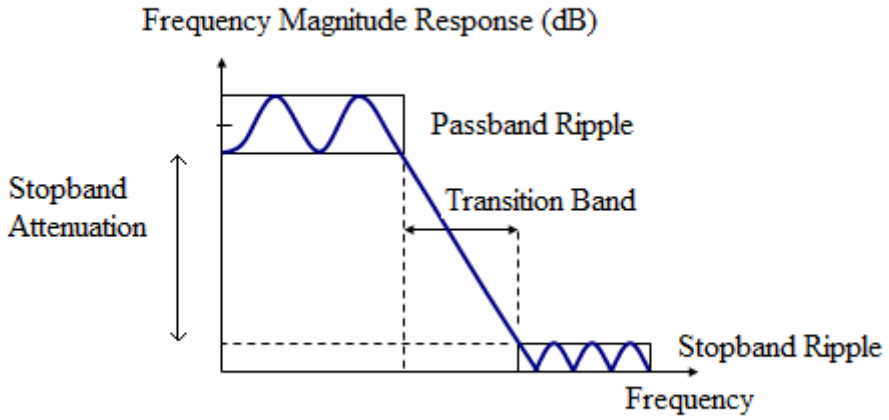
Signal se može posmatrati u vremenskom i frekvencijskom domenu. Vremenski domen predstavlja signal u odnosu na vreme, dok frekvencijski domen pokazuje kako su različite frekvencije raspodeljene u signalu. Transformacija signala iz vremenskog u frekvencijski domen vrši se pomoću Furijeove transformacije. U frekvencijskom domenu, svaka frekvencija može biti posmatrana kao pojedinačna komponenta signala, što omogućava precizno definisanje delova koje želimo filtrirati.

3.1.6 Digitalni filtri i osnovna podela

Postoji nekoliko osnovnih tipova filtara, od kojih svaki ima specifičnu ulogu:

1. **Low-pass filter** (niskopropusni filter): Ovaj filter propušta niske frekvencije, dok visokofrekvencijske komponente prigušuje ili eliminiše. Na primer, koristi se za uklanjanje šuma visokih frekvencija iz audio signala.
2. **High-pass filter** (visokopropusni filter): Suprotno low-pass filteru, high-pass filter propušta visoke frekvencije, dok niske frekvencije prigušuje. Na primer, koristi se za uklanjanje niskofrekvencijskih smetnji, poput brujanja mrežnog napona (50 Hz).
3. **Band-pass filter** (pojasnopropusni filter): Ovaj filter propušta samo određeni opseg frekvencija, dok ostale frekvencije prigušuje. Koristi se, na primer, u telekomunikacijama za izolaciju određenog kanala.
4. **Band-stop filter** (pojasnoprigušni filter): Ovaj filter radi suprotno od band-pass filtera – prigušuje određeni opseg frekvencija dok propušta sve ostale.

Kada se analizira rad filtara, javljaju se sledeći ključni pojmovi:
Passband: Opseg frekvencija koje filter propušta bez značajnijeg prigušenja. Na primer, kod low-pass filtera, to je opseg niskih frekvencija koje ostaju netaknute.
Stopband: Opseg frekvencija koje filter treba da potpuno priguši. Kod idealnog high-pass filtera, to su sve niske frekvencije ispod granične vrednosti.
Transition band (tranzicioni opseg): Deo između passbanda i stopbanda gde dolazi do prelaska između propusnog i nepropusnog opsega. U realnim filtrima, tranzicioni opseg nije idealno oštar i zavisi od karakteristika dizajna filtera.
Idealni filter: U teoriji, idealni filter ima savršeno definisan passband i stopband sa oštrom granicom između njih. Međutim, takvi filteri nisu mogući u praksi zbog fizičkih i matematičkih ograničenja.



Slika 6. frekvencijski odziv low pass filtra i prikaz osnovnih pojmova

Na slici x iznad, vidimo pojmove ripple (talasanje) i attenuation (prigušenje).
Stopband attenuation jednostavno označava jačinu filtra, odnosno jačinu kojom filtar priguši neželjene frekvencije, dok **ripple** predstavlja talasanje u prolaznom opsegu filtera – male promene u jačini signala koje filter propušta. U idealnom svetu oba ova pojma ne postoje, dok u realnom svetu predstavljaju vazan faktor svakog filtra i neizostavni su.

U kontekstu digitalnih sistema, kao što je mikrokontroler ESP32, filtri se implementiraju algoritamski. Ovo omogućava visok nivo preciznosti i

prilagodljivosti, ali zahteva poznavanje metoda kao što su **FIR** (Finite Impulse Response) i **IIR** (Infinite Impulse Response) filtri. FIR filtri su stabilni i imaju precizno definisane karakteristike, dok IIR filtri omogućavaju efikasniju realizaciju s manje resursa, ali mogu biti manje stabilni.

FIR i IIR filteri su dve osnovne kategorije digitalnih filtera koje se često koriste u obradi signala. Iako oba tipa imaju za cilj modifikaciju signala, razlikuju se u osnovnom načinu rada, matematičkoj strukturi, performansama i oblastima primene. Razumevanje ovih razlika ključno je za odabir odgovarajućeg filtera u zavisnosti od zahteva konkretnog projekta.

FIR filteri, odnosno filteri sa konačnim impulsnim odzivom, imaju svojstvo da impulsni odziv traje konačno vreme, odnosno završi se nakon određenog broja uzoraka. Ovo je zato što FIR filteri nemaju povratne veze (feedback) u svojoj strukturi, što znači da izlaz filtera zavisi samo od trenutnog i prethodnih ulaznih uzoraka. Matematički, FIR filter se izražava kao linearna kombinacija ulaznih uzoraka, što rezultuje stabilnim odzivom bez rizika od oscilacija ili nestabilnosti. Jedna od glavnih prednosti FIR filtera je njihova fazna karakteristika – mogu se dizajnirati tako da imaju tačno linearnu fazu, što znači da ne uvode fazno izobličenje u signal. Ovo ih čini pogodnim za aplikacije gde je očuvanje oblika signala od presudnog značaja, kao što su audio sistemi ili komunikacioni sistemi. Međutim, FIR filteri često zahtevaju veći broj koeficijenata (i time više računske snage) da bi postigli oštar prelaz između propusnih i zaustavnih opsega, što može biti nedostatak u aplikacijama gde su resursi ograničeni.

S druge strane, IIR filteri, ili filteri sa beskonačnim impulsnim odzivom, imaju povratne veze u svojoj strukturi, što znači da izlaz filtera zavisi ne samo od ulaznih uzoraka, već i od prethodnih izlaza. Ovo omogućava IIR filterima da sa manjim brojem koeficijenata postignu sličnu selektivnost kao FIR filteri, što ih čini efikasnijim u pogledu računske složenosti. Matematički gledano, IIR filteri su slični analognim filterima, jer se često baziraju na diskretizaciji poznatih analognih struktura kao što su Butterworth, Chebyshev ili Elliptic filteri. Međutim, zbog povratne veze, IIR filteri mogu biti skloni nestabilnosti ako nisu pravilno dizajnirani. Još jedna mana IIR filtera je to što obično nemaju linearnu fazu, što može dovesti do faznog izobličenja signala. Uprkos tome, njihova efikasnost i mogućnost implementacije u realnom vremenu čine ih popularnim izborom u aplikacijama kao što su filtriranje signala u realnom vremenu, obrada slike ili medicinski uređaji.

Kada biramo između FIR i IIR filtera, potrebno je uzeti u obzir nekoliko faktora. Ako je stabilnost i linearna faza ključna, FIR filteri su bolji izbor, iako su zahtevniji u pogledu procesorske snage. Ako je cilj brza i računski efikasna implementacija sa ograničenim resursima, IIR filteri će verovatno biti adekvatniji, pod uslovom da se pažljivo dizajniraju kako bi se izbegla nestabilnost. Konačan izbor često zavisi od specifičnih zahteva aplikacije, kao i od kompromisa između performansi i složenosti.

Razumevanje filtracije ključno je za mnoge primene:

1. Audio obradi, low-pass filtri uklanjaju šum, dok high-pass filtri eliminišu neželjene niske frekvencije.

2. Obrada slike, band-pass filtri mogu se koristiti za prepoznavanje određenih obrazaca.
3. Telekomunikacije, filtri se koriste za separaciju kanala i eliminaciju smetnji.

Razumevanjem ovih pojmova možemo preciznije analizirati i oblikovati digitalne filtre u okviru sistema baziranih na mikrokontrolerima poput ESP32. Ovo otvara mogućnost za realizaciju različitih projekata, od obrada senzorskih signala do naprednih komunikacionih sistema.

4. Implementacija filtra u MCU

4.1.1 Odabir filtra za različite signale

U ovom projektu, kao što je ranije pomenuto, koristimo fotorezistor kao senzor za detekciju intenziteta svetla. Fotorezistor ima otpornost koja varira u zavisnosti od osvetljenja, i to od približno 1 kΩ kada je izložen jakom svetlu do oko 33 kΩ kada se nalazi u mraku. Da bismo mogli očitavati te promene, povezali smo fotorezistor u razdelnik napona zajedno sa otpornikom od 2.2 kΩ. Na ulaz razdelnika dovodimo napon od 5 V, dok na izlazu razdelnika dobijamo napon koji varira u zavisnosti od intenziteta svetla – od 0 V do maksimalnih 2 V.

Izlaz razdelnika napona povezali smo na pin GPIO34 na ESP32 mikrokontroleru, koji je konfigurisan kao ulaz za ADC (Analog-to-Digital Converter). ESP32 ima ugrađeni ADC koji može očitavati analogne signale i pretvarati ih u digitalne vrednosti. ADC na ESP32 radi u rezoluciji od 12 bita, što znači da signal od 0 do 2 V možemo mapirati u opseg od 0 do 4095, gde svaka jedinica predstavlja mali korak u očitanoj vrednosti napona.

Za realizaciju ovog projekta koristili smo ESP-IDF okruženje, zajedno sa PlatformIO alatom za razvoj. ESP-IDF omogućava direktan pristup hardveru mikrokontrolera i preciznu kontrolu funkcionalnosti, uključujući ADC modul. Da bismo očitali signal sa fotorezistora, napisali smo program koji koristi ADC drajver za čitanje podataka sa GPIO34 pina. Očitane vrednosti zatim obrađujemo u mikrokontroleru i šaljemo na računar putem serijske komunikacije.

Za slanje podataka koristili smo funkciju printf, koja omogućava ispis digitalnih vrednosti signala na serijski port. Na računaru smo razvili jednostavnu Python skriptu

koja čita podatke sa serijskog porta u realnom vremenu i čuva ih u CSV fajl. Ovaj fajl sadrži očitane vrednosti signala, koje su dalje spremne za analizu.

CSV fajl omogućava da dobijemo pregled očitavanja u formatu koji je lak za manipulaciju i analizu u raznim softverima, poput MATLAB-a ili Excel-a. Ovaj proces je ključan jer omogućava detaljnu analizu ponašanja signala, kao i primenu različitih digitalnih filtera koje ćemo implementirati u sledećim koracima projekta.

Korišćenjem ovog pristupa, uspeli smo da napravimo jednostavan i funkcionalan sistem za očitavanje, prikupljanje i čuvanje podataka sa senzora svetla, koji je ujedno modularan i spreman za proširenja. U sledećim fazama projekta fokusiraćemo se na primenu digitalnih filtera za obradu ovog signala i izdvajanje korisnih informacija iz njega.

```c
#include <stdio.h>              // Standardna biblioteka za ulaz/izlaz
#include "freertos/FreeRTOS.h"  // FreeRTOS osnovne definicije
#include "freertos/task.h"      // FreeRTOS task funkcije
#include "driver/adc.h"         // Biblioteka za rad sa ADC-om

// ADC konfiguracija
#define ADC_CHANNEL ADC1_CHANNEL_6 // Koristi se kanal 6 (GPIO34) za očitavanje
#define SAMPLE_RATE 1000 // Uzorkovanje signala 1000 puta u sekundi (1000 Hz)

// Glavna funkcija
void app_main() {
    // Podešavanje rezolucije ADC-a na 12 bita
    adc1_config_width(ADC_WIDTH_BIT_12);

    // Podešavanje slabljenja (attenuation) za izabrani kanal
    adc1_config_channel_atten(ADC_CHANNEL, ADC_ATTEN_DB_11);

    while (1) {
        // Očitavanje vrednosti sa definisanog pina
        int adc_raw = adc1_get_raw(ADC_CHANNEL);

        // Slanje očitane vrednosti na serijski port u formatu sa tri decimale
        printf("%.3f\n", adc_raw);

        // Pauza između uzoraka – određuje brzinu uzorkovanja
        vTaskDelay(pdMS_TO_TICKS(1000 / SAMPLE_RATE));
    }
}
```

Slika 7. Kod za očitavanje signala i njegovo pojašnjenje

Ovaj kod implementira osnovno očitavanje analognog signala sa fotorezistora pomoću ADC modula na ESP32 i njegovu serijsku komunikaciju. Na početku, ADC modul se konfiguriše tako da radi u 12-bitnom režimu rezolucije, što omogućava očitavanje vrednosti od 0 do 4095. Povezani senzor se nalazi na GPIO34 (kanal ADC1_CHANNEL_6), a koristi se pojačanje od 11 dB (ADC_ATTEN_DB_11), čime se omogućava merenje napona do približno 3,6 V. U beskonačnoj petlji, program čita trenutnu vrednost sa ADC kanala pomoću funkcije adc1_get_raw() i šalje je putem serijskog porta koristeći printf(). Pauza između uzorkovanja se precizno kontroliše funkcijom vTaskDelay() na osnovu definisane frekvencije uzorkovanja (SAMPLE_RATE), što osigurava da se očitavanja vrše ravnomerno. Ovaj kod

predstavlja jednostavnu i efikasnu osnovu za obradu signala i dalje korišćenje u filterima.

Da bi se podaci dobijeni sa mikrokontrolera dalje obrađivali, potrebno ih je prebaciti na računar na kojem se vrši dizajn filtera. Za prenos podataka napisan je jednostavan Python program koji očitava dolazne informacije sa serijskog porta računara. Nakon toga, kreira se CSV fajl u koji se upisuju svi očitani podaci. Čitanje i upisivanje podataka se prekida pritiskom na tastere Ctrl+C. Na taj način dobijamo fajl koji sadrži signal i koji je spreman za dalju obradu.

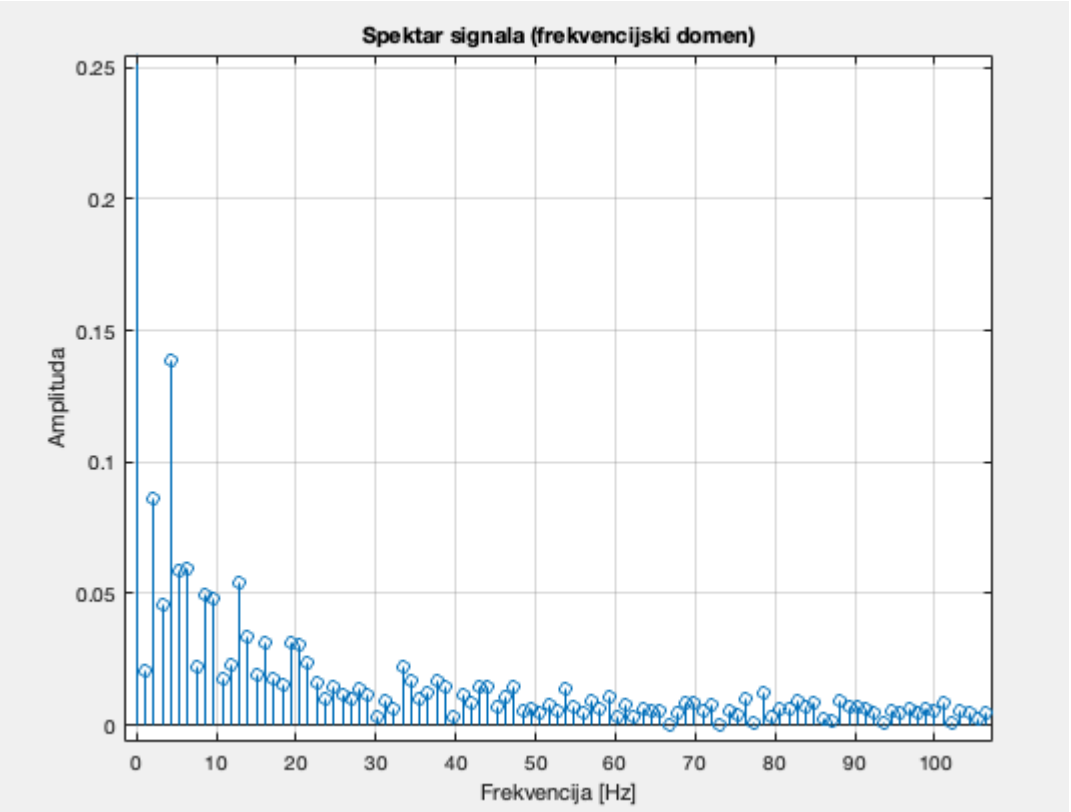
```python
import serial  # Uvoz biblioteke za rad sa serijskom komunikacijom

# Otvaranje serijskog porta (potrebno je uneti tačan port koji ESP32 koristi)
ser = serial.Serial('/dev/tty.usbserial-0001', 115200)  # Naziv porta i brzina

output_file = "adc_data.csv"  # Naziv CSV fajla u koji će se snimati podaci

# Otvaranje fajla u režimu pisanja
with open(output_file, "w") as f:
    print("Započeto snimanje podataka...")  # Informacija da je snimanje počelo
    try:
        while True:
            # Čitanje sa serijskog porta i uklanjanje nepotrebnih karaktera
            line = ser.readline().decode('utf-8', errors='ignore').strip()
            f.write(line + "\n")  # Upisivanje pročitanog reda u CSV fajl
            print(line)  # Prikaz pročitanog reda u terminalu
    except KeyboardInterrupt:
        # Prekid programa kada korisnik pritisne Ctrl+C
        print("Završeno snimanje.")
        ser.close()  # Zatvaranje serijskog porta
```

Slika x: Python kod za prenos podataka sa mikrokontrolera

4.1.2 Analiza signala u MATLAB-u

Da bismo analizirali i procesirali signal dobijen sa ESP32 mikrokontrolera, koristimo MATLAB kao alat za implementaciju i testiranje digitalnih filtera. Ovaj deo obrade signalnih podataka oslanja se na učitavanje signala, postavljanje osnovnih parametara, dizajniranje Butterworth filtra i primenu filtra na ulazni signal. Svaka komponenta koda ima specifičnu svrhu, pa ćemo detaljno analizirati njegovu funkcionalnost.

Pre nego što se dizajnira konkretan digitalni filtar, važno je izvršiti analizu signala u frekvencijskom domenu (Slika 8.) kako bi se identifikovao opseg frekvencija koje želimo da sačuvamo, kao i one koje treba eliminisati. Prikazom spektra signala pomoću brze Furijeove transformacije (FFT) moguće je vizuelno uočiti dominantne komponente – korisne informacije obično se nalaze u niskofrekventnom opsegu, dok su visoke frekvencije često posledica šuma. Na osnovu takvog prikaza određuje se granična frekvencija (cutoff frequency) filtra, odnosno tačka nakon koje filtriranje počinje da potiskuje neželjene komponente. U konkretnom slučaju, signal sa senzora ima značajan sadržaj do približno 30 Hz, dok se iznad te vrednosti dominantno javlja šum, zbog čega je kao cutoff frekvencija odabrana upravo ta vrednost. Ovakav pristup može se primeniti i na druge signale – jednostavnom analizom njihovog spektra možemo efikasno odabrati odgovarajuću vrstu i parametre filtra, što predstavlja osnovu za uspešnu obradu signala u digitalnim sistemima.



Slika 8. Frekvencijski odziv signala

csvread('adc_data.csv'); omogućava učitavanje signala iz CSV fajla u MATLAB. Fajl adc_data.csv sadrži digitalizovane uzorke koje je ADC na ESP32 generisao i poslao preko serijske komunikacije. Ovaj signal predstavlja ulaz u naš proces obrade. Funkcija csvread učitava sve vrednosti u obliku vektora, gde svaki element odgovara jednom uzorku signala. Ključni korak ovde je osigurati da format podataka u fajlu odgovara očekivanju MATLAB-a – na primer, da su vrednosti uredno odvojene zarezima ili novim redovima.

```
signal = csvread('adc_data.csv'); % Lokacija csv fajla
Fs = 1000;  % Frekvencija odabiranja
Fc = 30;    % Frekvencija odsecanja (Cutoff freq)
t = (0:length(signal)-1) / Fs; % Vreme u sekundama

N = 4;  % Red filtra
Wn = Fc / (Fs / 2);  % Normalized cutoff frequency (0 to 1)
```

Slika 9. Definisane vrednosti filtra

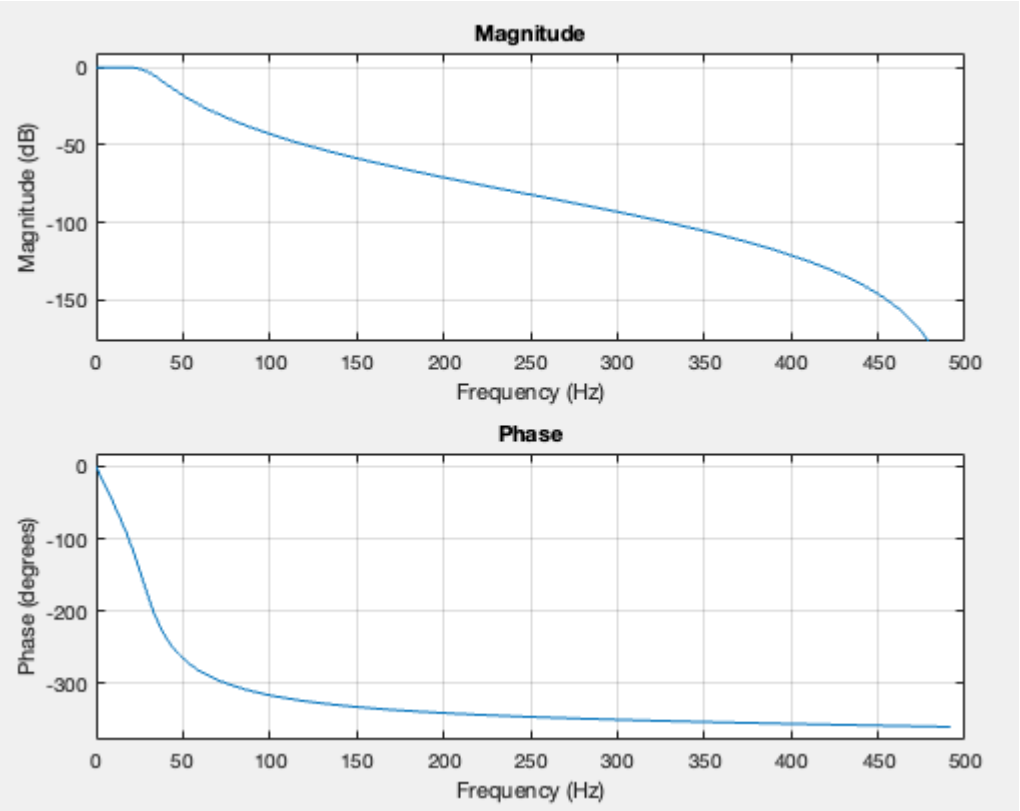
Definišemo frekvenciju uzorkovanja pomoću Fs = 1000;. Ova vrednost (1000 Hz) odražava brzinu uzorkovanja signala na ESP32. Preciznost ove vrednosti je od suštinskog značaja, jer ona direktno utiče na analizu signala, dizajn filtra i interpretaciju rezultata. Ako je frekvencija uzorkovanja pogrešno postavljena, rezultati filtriranja mogu biti netačni ili čak beskorisni. Sledeće, graničnu frekvenciju niskopropusnog filtra postavljamo pomoću Fc = 30;. Ova vrednost (30 Hz) je odabrana s obzirom na prirodu signala i ciljeve obrade – na primer, eliminisanje šumova visokih frekvencija ili izdvajanje korisnih podataka u niskofrekventnom opsegu. Granična frekvencija je ključni parametar koji definiše koliko frekvencija će filter propustiti. Vektor vremena t = (0:length(signal)-1) / Fs; konstruisan je kako bi omogućio vizualizaciju signala u vremenskom domenu. Ovaj vektor vremenskih oznaka se generiše tako što svaki uzorak signalnog vektora dodeljujemo odgovarajućem trenutku u sekundama. Pomoću ovog vektora možemo precizno pratiti promene signala tokom vremena.

Zatim, red filtra definišemo sa N = 4;, što označava da koristimo Butterworth filter četvrtog reda. Viši redovi filtra omogućavaju bolju selekciju frekvencija, ali povećavaju složenost i mogu uvoditi fazne pomake. Četvrti red je kompromis između efikasnosti i jednostavnosti implementacije, posebno na mikrokontrolerima. Normalizovana granična frekvencija računa se sa Wn = Fc / (Fs / 2);. Ova normalizacija je neophodna jer MATLAB zahteva da se granične frekvencije izraze kao proporcija Nyquistove frekvencije (Fs / 2). Nyquistova frekvencija je teoretski maksimum koji signal može imati bez aliasinga, pa je korektna normalizacija ključna za dizajn tačnog filtra.

```

% Primenjujemo butterworth filtar
% i dodajemo koeficijente za low pass filtar
[b, a] = butter(N, Wn, 'low');

% uzimamo koeficijente i formatiramo za c:
% definisemo niz i upisujemo prvu vrednost
fprintf('float b[] = {%.6f', b(1));
fprintf(', %.6f', b(2:end)); % vrednosti duzine b
fprintf('};\n'); % zatvorena zagrada na kraju
% isto za a niz
fprintf('float a[] = {%.6f', a(1));
fprintf(', %.6f', a(2:end));
fprintf('};\n');
```

Slika 10. primena filtra i formatiranje dobijenih koeficijenata

Dizajn filtra se vrši pomoću butter funkcije: [b, a] = butter(N, Wn, 'low');. Ova funkcija vraća dva vektora, b i a, koji predstavljaju koeficijente digitalnog filtra. Ovi koeficijenti određuju matematičku funkciju filtra, tj. kako će signal biti procesuiran. U ovom slučaju, dizajniran je niskopropusni filtar koji propušta frekvencije ispod 30 Hz, dok značajno smanjuje amplitude frekvencija iznad tog praga. Kada odredimo koeficijente, možemo ih formatirati i prekopirati u kod za mikrokontroler. Za to su zadužene funkcije fprintf.



Slika 11. prikaz faznog i amplitudskog odziva za kreirani filtar

Na osnovu dobijenih koeficijenata moguće je prikazati frekvencijski odziv dizajniranog digitalnog filtra, uključujući kako amplitudsku, tako i faznu karakteristiku (Slika 11.). Ovakva vizuelizacija omogućava precizan uvid u ponašanje filtra kroz sve ključne opsege – propusni (pass-band), prelazni (transition-band) i zaustavni (stop-band). Na osnovu prikazanog odziva moguće je izvršiti dodatna podešavanja parametara filtra, poput granične frekvencije i reda filtra, kako bi se postigli optimalni rezultati za konkretan slučaj. Ukoliko se na osnovu grafa uoči da prelaz između propusnog i zaustavnog opsega nije dovoljno strm, može se razmotriti i upotreba druge vrste filtra, poput elliptic filtra, koji omogućava oštriji pad u prelaznom opsegu uz manji red filtra.

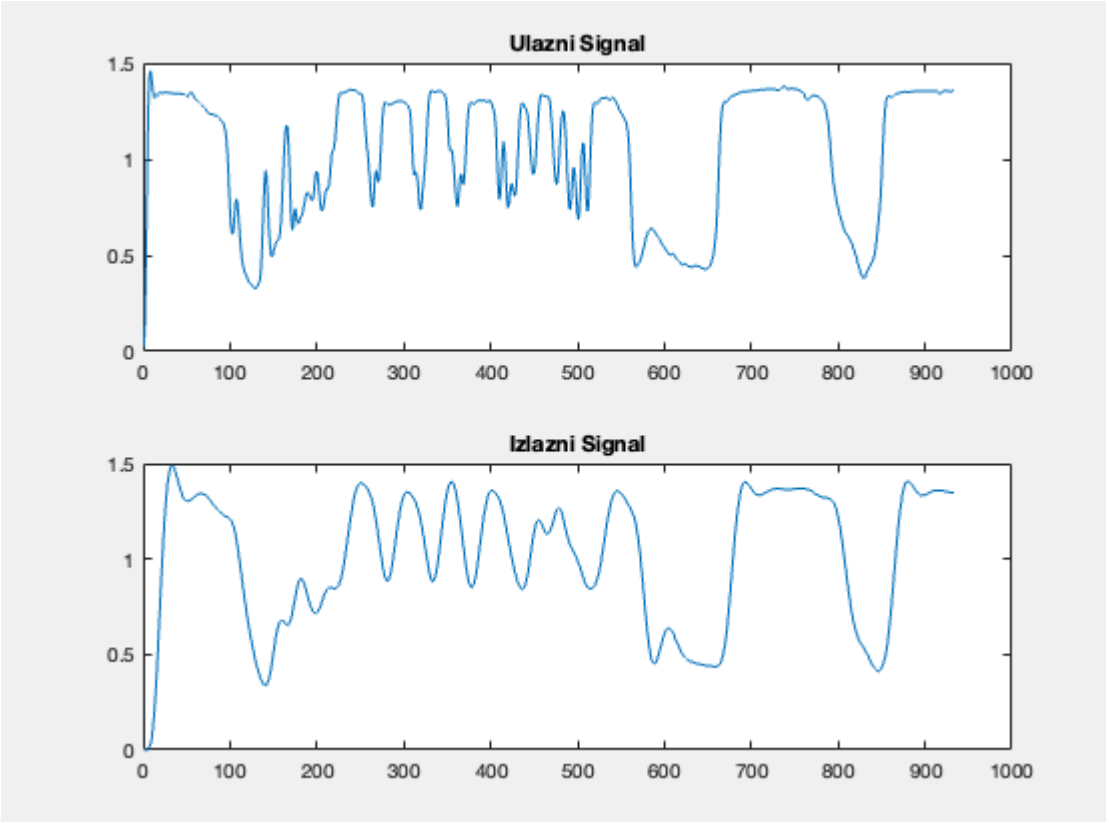
Do sadasnja funkcionalnost je dovoljna za implementaciju filtra na mikrokontroleru, ali bi njegovo testiranje na taj način bilo neefikasno. Zbog toga ćemo najpre testirati filtar i analizirati njegov odziv unutar MATLAB-a, što će nam omogućiti brzo i jednostavno prilagođavanje parametara, ukoliko bude potrebno, pre nego što koeficijente prebacimo u kod za ESP32.

Linija filtered_signal = filter(b, a, signal); primenjuje dizajnirani filtar na ulazni signal. Funkcija filter koristi koeficijente b i a kako bi izračunala izlazni signal, odnosno filtriranu verziju ulaznog signala. Ovde signal prolazi kroz proces koji eliminiše komponente visokih frekvencija, ostavljajući samo korisne podatke.

```
22
23      % Primena dizajniranog digitalnog filtra na ulazni signal
24      filtered_signal = filter(b, a, signal);
25      % Otvaranje novog prozora za prikaz grafa
26      figure;
27
28      % Prva slika u okviru subplot-a (gornji grafikon)
29      subplot(2, 1, 1); % Podela figure na 2 reda i 1 kolonu
30      plot(signal); % Prikazivanje originalnog (nefiltriranog) signala
31      title('Ulazni Signal'); % Naslov grafikona: "Ulazni Signal"
32
33      % Druga slika u okviru subplot-a (donji grafikon)
34      subplot(2, 1, 2); % Aktivna je druga (donja) sekcija
35      plot(filtered_signal); % Prikazivanje filtriranog signala
36      title('Izlazni Signal'); % Naslov grafikona: "Izlazni Signal"
```

Slika x: Primena filtra na ulazni signal i prikaz oba signala

Za vizuelizaciju rezultata koristimo funkciju plot. Prvi grafikon prikazuje originalni signal, dok drugi prikazuje filtrirani signal.



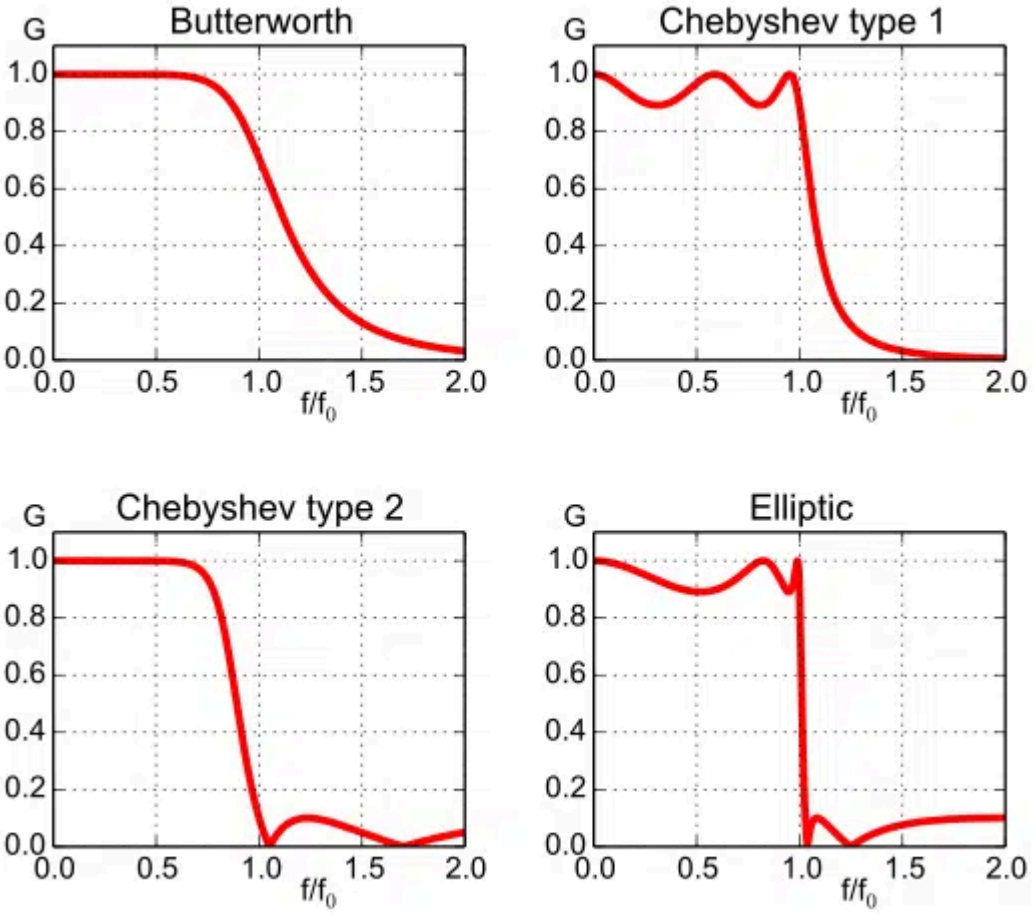
Slika x: Prilaz ulaznog i obrađenog signala

Ova vizualna poređenja su korisna za procenu efikasnosti filtra i uočavanje razlika između originalnog i obrađenog signala. Mada ova linija nije kritična za implementaciju na mikrokontroleru, ona pomaže u analizi rezultata i finom podešavanju parametara filtra.

4.1.3 Različite vrste filtara

Butterworth filtar je jedan od najpoznatijih i najčešće korišćenih tipova filtara u oblasti obrade signala, prvenstveno zbog svoje karakteristike maksimalno ravne amplitudske frekvencijske karakteristike u propusnom opsegu. Ovaj filtar je prvi put predstavio Stephen Butterworth 1930. godine, sa idejom da kreira filtar koji ne bi imao oscilacije u propusnom opsegu, što ga čini idealnim za aplikacije gde je potrebno zadržati nepromenjen intenzitet signala unutar definisanog opsega frekvencija. Butterworth filtar ima takozvanu monotono opadajuću karakteristiku u zaustavnom opsegu, ali prelaz između propusnog i zaustavnog opsega nije nagao već postepen, što je ključna razlika u odnosu na druge vrste filtara. Matematički, funkcija prenosa Butterworth filtra projektovana je tako da se polovi nalaze simetrično raspoređeni na kružnici u kompleksnoj ravni, pri čemu njihov položaj zavisi od reda filtra. Veći red omogućava oštriji prelaz između opsega, ali takođe zahteva veću kompleksnost u implementaciji.

Pored Butterworth filtra, postoje i drugi poznati filtri koji se koriste u zavisnosti od zahteva aplikacije. Chebyshev filtar, na primer, nudi strmiji prelaz između propusnog i zaustavnog opsega u poređenju sa Butterworth filtrom, ali dolazi s kompromisom – oscilacije u amplitudskoj karakteristici. Postoje dva tipa Chebyshev filtra: Tip I, koji ima oscilacije u propusnom opsegu, i Tip II, kod kojeg su oscilacije prisutne u zaustavnom opsegu. Elliptic filtar predstavlja još ekstremniju varijantu jer ima oscilacije u oba opsega, ali zauzvrat pruža najstrmiji prelaz između propusnog i zaustavnog opsega. Na drugom kraju spektra nalazi se Bessel filtar, koji je specifičan po svojoj linearnosti faze, što ga čini idealnim za aplikacije gde je očuvanje oblika talasnog signala od suštinskog značaja, poput sistema za prenos govora i muzike.



Slika 12. najčešći filtri koji se koriste i njihovi odzivi u frekvencijskom domenu

U ovom projektu odabran je Butterworth filtar zbog njegove jednostavnosti implementacije i predvidivog ponašanja, što ga čini pogodnim za uklanjanje visokofrekvencijskog šuma bez značajnog uticaja na osnovni signal. Njegova monotona amplitudska karakteristika osigurava da nema oscilacija koje bi mogle uneti dodatne nepreciznosti u sistem. Ipak, važno je napomenuti da izbor filtra uvek zavisi od specifičnih potreba sistema. Na primer, u situacijama gde je potrebna velika preciznost u zaustavljanju neželjenih frekvencija, Chebyshev ili Elliptic filtri mogu biti bolji izbor, dok je Bessel filtar idealan kada je ključno očuvanje vremenskog rasporeda signala. Razumevanje prednosti i nedostataka različitih filtara omogućava inženjerima da u svakom trenutku odaberu optimalno rešenje za konkretne zadatke.

4.1.4 Implementacija filtra unutar mikrokontrolera

Implementacija filtra unutar ESP32 predstavlja ključni korak u praktičnoj realizaciji digitalne obrade signala, gde se teorijski modeli primenjuju na stvarne podatke u realnom vremenu. Nakon što smo kroz prethodne analize odredili parametre filtra i dobili odgovarajuće koeficijente, sada je potrebno te vrednosti preneti u kod i omogućiti mikrokontroleru da neprekidno obrađuje ulazni signal. Korišćenjem ESP-IDF okruženja i PlatformIO alata, omogućena je efikasna implementacija IIR filtra koji će filtrirati signal očitan sa ADC modula ESP32. Ovaj proces podrazumeva

konfiguraciju ADC jedinice, pravilno uzorkovanje signala, skladištenje prethodnih uzoraka radi rekurzivne obrade i implementaciju same matematičke operacije filtra. Cilj ove implementacije je da se signal prečisti od neželjenih frekvencija ili šuma, bez značajnog kašnjenja u obradi, što je ključno za aplikacije koje zahtevaju brze i precizne odgovore sistema. Kako bi se osigurala tačnost rezultata, važno je da uzorkovanje bude konzistentno i da algoritam filtra bude pravilno optimizovan za rad u ograničenom okruženju mikrokontrolera. U narednom delu detaljno će biti objašnjena implementacija ovog sistema, uključujući konkretan kod koji realizuje obradu signala u realnom vremenu.

Kod implementacije IIR (Infinite Impulse Response) filtra na ESP32 mikrokontroleru demonstrira proces obrade analognog signala u realnom vremenu koristeći ESP-IDF framework. Prvo je neophodno konfigurisati ADC (Analog-to-Digital Converter) na ESP32 kako bi analogni signal bio pravilno uzorkovan i konvertovan u digitalni format. U ovom slučaju, ADC je postavljen na 12-bitnu rezoluciju, što omogućava očitavanje vrednosti u opsegu od 0 do 4095. Izabrani kanal je GPIO34, dok je atenuacija postavljena na 11 dB, čime se proširuje dinamički opseg ulaznog signala. Frekvencija uzorkovanja je postavljena na 1000 Hz kako bi se obezbedila dovoljna gustina uzoraka za kvalitetnu obradu signala. Ova konfiguracija je ključna jer svako odstupanje u frekvenciji uzorkovanja može značajno uticati na performanse filtra i na tačnost uklanjanja neželjenih komponenti.

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/adc.h"
#include "esp_log.h"


// Definisanje ADC kanala; GPIO34 na ESP32 je povezan na ADC1_CHANNEL_6
#define ADC_CHANNEL ADC1_CHANNEL_6
// Frekvencija uzorkovanja signala u Hz (10 uzoraka u sekundi)
#define SAMPLE_RATE 10

// Red filtra – mora odgovarati onom iz MATLAB simulacije
#define FILTER_ORDER 3

// Koeficijenti niza (b[]) iz MATLAB dizajna – ulazni uzorci
float b[] = {0.018099, 0.054297, 0.054297, 0.018099};
// težinski koeficijenti za izlazne uzorke (feedback)
float a[] = {1.000000, -1.760042, 1.182893, -0.278060};

// Bafer za ulazne uzorke (x[0] je najnoviji, x[1], x[2]... su stariji)
float x[FILTER_ORDER + 1] = {0};  // Inicijalizacija ulaznog niza na nule
// Bafer za izlazne uzorke (y[0] je najnoviji izlaz, y[1], y[2]... su prethodni)
float y[FILTER_ORDER + 1] = {0};  // Inicijalizacija izlaznog niza na nule
```

Slika 13. Kod koji definiše neophodne konstante i promenljive

Definicija konstante FILTER_ORDER postavlja red filtra na 3, što znači da će izlazni signal zavisiti od trenutnog i tri prethodna uzorka ulaza i izlaza. Koeficijenti b[] i a[] određuju karakteristike filtra i izračunati su u MATLAB-u, pri čemu b[] sadrži koeficijente koji množe ulazne uzorke, dok a[] sadrži koeficijente za prethodne

izlazne uzorke. Prvi element a[] je uvek 1, što služi za normalizaciju i osigurava numeričku stabilnost. Dodatni nizovi x[] i y[] čuvaju ulazne i izlazne uzorke signala, sa veličinom FILTER_ORDER + 1, jer je potrebno sačuvati trenutni i prethodne uzorke. Prilikom svakog novog uzorkovanja, najnoviji uzorak se dodaje na početak niza, dok se stariji pomeraju unazad, omogućavajući efikasnu primenu rekurzivne formule IIR filtra. Ova struktura osigurava da trenutni izlaz bude kombinacija trenutnog i prethodnih ulaza, kao i prethodnih izlaza, čime se postiže željeni frekvencijski odziv sistema i vrši selektivna filtracija signala u realnom vremenu.

```
25
26  // Funkcija koja prima novi uzorak i vraća filtriranu vrednost
27  float filter_sample(float new_sample) {
28
29      // Pomeranje prethodnih uzoraka unazad za jedno mesto
30      for (int i = FILTER_ORDER; i > 0; i--) {
31          x[i] = x[i - 1];  // Pomeranje ulaznih uzoraka udesno
32          y[i] = y[i - 1];  // Pomeranje izlaznih uzoraka udesno
33      }
34
35      // Dodavanje novog ulaznog uzorka na početak niza
36      x[0] = new_sample;
37      // Inicijalizacija nove izlazne vrednosti
38      y[0] = 0;
39
40      // Filtrirana vrednost (y[0]) se računa kao zbir ulaznih
41      // uzoraka (x[]) množenih sa b[] koeficijentima
42      // i prethodnih izlaza (y[]) množenih sa a[] koeficijentima (bez a[0]).
43      for (int i = 0; i <= FILTER_ORDER; i++) {
44          // Sabiranje vrednosti trenutnog i prethodnih ulaza
45          y[0] += b[i] * x[i];
46
47          if (i > 0) {
48              // Oduzimanje vrednosti prethodnih izlaza (bez a[0])
49              y[0] -= a[i] * y[i];
50          }
51      }
52
53      // Vraćanje novog filtriranog izlaza
54      return y[0];
55  }
```

Slika 14. Funkcija koja filtrira dobijenu vrednost

Funkcija filter_sample implementira IIR filter tako što obrađuje novi ulazni uzorak i izračunava odgovarajući filtrirani izlaz koristeći unapred definisane koeficijente. Prvi deo funkcije sadrži petlju koja pomera sve vrednosti nizova x[] (ulazni uzorci) i y[] (izlazni uzorci) za jedno mesto unazad. Ovo se radi zato što se IIR filter oslanja na prethodne vrednosti signala kako bi izračunao novi izlaz. Konkretno, svaka vrednost iz niza x se premešta u sledeću poziciju, a isto se dešava i sa nizom y, kako bi najnoviji uzorak mogao da zauzme prvo mesto u nizu, dok se stari uzorci ne gube već se pomeraju niže u nizu. Nakon toga, novi uzorak new_sample se postavlja na početak niza x[], što znači da će on biti uzet u obzir pri računanju trenutnog izlaza.

Drugi deo funkcije računa filtrirani izlaz y[0] primenom razlike između dve sume – jedne koja koristi koeficijente b[] i ulazne vrednosti x[], i druge koja koristi koeficijente a[] i prethodne izlazne vrednosti y[]. Prvo se trenutni izlaz inicijalizuje množenjem najnovijeg ulaza sa prvim koeficijentom b[0], nakon čega sledi petlja koja dodaje doprinose ostalih ulaznih uzoraka skaliranih sa koeficijentima b[i], čime se postiže proporcionalni deo filtra. Zatim se od rezultata oduzima suma prethodnih izlaznih uzoraka pomnoženih sa odgovarajućim koeficijentima a[i], čime se uvodi povratna veza karakteristična za IIR filtere. Ova struktura omogućava filtriranje signala tako da se određene frekvencije prigušuju ili propuštaju u zavisnosti od izračunatih koeficijenata. Na kraju, izračunata vrednost filtriranog signala se vraća iz funkcije, omogućavajući njenu dalju obradu ili prikaz u realnom vremenu.

```
57  void app_main() {
58      // Konfigurisanje ADC na 12 bita (vrednosti od 0 do 4095)
59      adc1_config_width(ADC_WIDTH_BIT_12);
60
61      // Postavljanje atenuacije za izabrani ADC kanal (11 dB ~ 3.3V)
62      adc1_config_channel_atten(ADC_CHANNEL, ADC_ATTEN_DB_11);
63
64      // Beskonačna petlja - očitavanje vrednosti se izvršava neprekidno
65      while (1) {
66          // Čitanje sirove ADC vrednosti sa definisanog kanala (0–4095)
67          int adc_raw = adc1_get_raw(ADC_CHANNEL);
68
69          // Primena digitalnog filtra na očitanu vrednost
70          float filtered_value = filter_sample(adc_raw);
71
72          // Ispis filtrirane vrednosti na serijski terminal
73          printf("%.3f\n", filtered_value);
74
75          // Kašnjenje od 10 milisekundi (učestalost uzorkovanja ~ 100 Hz)
76          vTaskDelay(pdMS_TO_TICKS(10));
77      }
78  }
```

Slika 15. Glavna funkcija za očitavanje i filtriranje signala

Glavna funkcija app_main povezuje sve ključne delove koda i omogućava kontinuiranu obradu signala. Prvi korak u ovoj funkciji je očitavanje sirovih vrednosti sa ADC-a pomoću funkcije adc1_get_raw. Ova sirova vrednost se zatim prosleđuje funkciji filter_sample radi obrade, a filtrirana vrednost se šalje putem serijskog porta pomoću funkcije printf. Na ovaj način, korisnik može u realnom vremenu pratiti rezultate filtracije i analizirati efikasnost implementacije. Frekvencija uzorkovanja je pažljivo kontrolisana pomoću funkcije vTaskDelay, koja uvodi precizne vremenske pauze između uzorkovanja. Ovaj korak je od suštinskog značaja jer nepravilno uzorkovanje može dovesti do izobličenja signala i grešaka u radu filtra. Kontrolisana učestalost uzorkovanja osigurava da svaki uzorak signalizira tačno stanje ulaznog signala u tom trenutku, što je neophodno za pravilno funkcionisanje IIR algoritma.

Pored tehničkih detalja implementacije, ovaj kod ilustruje i važnost pažljivog projektovanja i testiranja filtra pre njegove implementacije na mikrokontroler.

Koeficijenti filtra su unapred definisani i optimizovani u MATLAB-u, gde je moguće simulirati različite scenarije i proveriti odziv filtra na različite vrste signala. Ovaj proces značajno smanjuje mogućnost grešaka tokom rada mikrokontrolera, jer omogućava korisniku da unapred prilagodi parametre filtra, poput frekvencije odsecanja ili reda filtra. Nakon što su koeficijenti potvrđeni, njihova implementacija u kod za mikrokontroler postaje jednostavna, jer su svi potrebni parametri već poznati. Ovaj pristup takođe omogućava optimizaciju resursa mikrokontrolera, jer se koristi samo onoliko memorije i procesorske snage koliko je zaista potrebno za efikasnu obradu signala.

Implementacija ovog filtra na ESP32 pokazuje kako se teoretski koncepti obrade signala mogu primeniti u praksi na uređajima sa ograničenim resursima. IIR filtri, iako numerički zahtevniji od FIR filtara, omogućavaju visoku preciznost i efikasnost u obradi signala uz minimalno korišćenje memorije i procesorske snage. Kroz pažljivu konfiguraciju ADC-a, precizno definisanje koeficijenata i pravilno upravljanje učestalošću uzorkovanja, moguće je postići pouzdanu i tačnu obradu signala čak i u stvarnom vremenu. Rezultati ovog procesa mogu se koristiti u različitim aplikacijama, od analize biomedicinskih signala do industrijske automatizacije, čime se dodatno naglašava značaj digitalne obrade signala u savremenim mikrokontrolerskim sistemima.

5. Načini za unapredjenje projekta

5.1.1 Korišćenje drugog mikrokontrolera

Iako je ESP32 popularan mikrokontroler sa brojnim prednostima, jedno od njegovih ograničenja u ovom projektu je kvalitet ADC konverzije. ADC u ESP32 nije visokih performansi i može uvesti šum i nelinearnosti u očitane vrednosti, što može negativno uticati na preciznost filtriranja signala. Nasuprot tome, mikrokontroleri poput STM32 nude ADC konvertere sa većom rezolucijom i boljim performansama, često sa dodatnim mogućnostima kao što su višekanalno uzorkovanje i interni hardverski filtri. Implementacija ovog projekta na STM32 platformi mogla bi doneti preciznije rezultate i omogućiti pouzdaniju obradu signala, posebno u aplikacijama gde je potrebna visoka tačnost merenja.

5.1.2 Korišćenje već postojeće DSP - biblioteke

Drugi potencijalni način unapređenja ovog sistema je korišćenje ESP-DSP biblioteke, koja nudi optimizovane DSP funkcije za ESP32. Ova biblioteka je deo ESP-IDF okruženja i omogućava korišćenje unapred optimizovanih algoritama za digitalnu obradu signala, uključujući FFT, IIR, FIR i vektorske matematičke operacije. Funkcije u ovoj biblioteci su često pisane u asembleru, čime se postiže veća brzina izvršavanja u odnosu na standardne C implementacije. Takođe, omogućena je podrška za operacije u pokretnom zarezu (32-bitni float) i sa 16-bitnim celobrojnim vrednostima, što može poboljšati performanse filtra i preciznost obrade signala. Integracija ove biblioteke u postojeći kod mogla bi značajno ubrzati obradu podataka i smanjiti opterećenje procesora.

5.1.3 Vizuelni prikaz signala u realnom vremenu

Za vizuelizaciju procesuiranog signala koristio sam Python skriptu koja očitava filtrirane vrednosti preko serijske komunikacije i prikazuje ih u realnom vremenu pomoću matplotlib biblioteke. Ova metoda omogućava jednostavan uvid u performanse filtra i pomaže pri analizi signala bez potrebe za dodatnom opremom. Podaci se učitavaju direktno iz mikrokontrolera i ažuriraju na grafiku, čime se dobija jasan prikaz kako filtar utiče na signal.

Skripta koristi deque strukturu za čuvanje ograničenog broja poslednjih uzoraka signala, što omogućava kontinuirano ažuriranje grafikona bez usporavanja programa. Signal se očitava preko serijskog porta, parsira u numeričku vrednost i dodaje u niz podataka koji se prikazuju na ekranu. Ovaj način vizuelizacije pomaže u podešavanju parametara filtra i identifikaciji eventualnih problema, poput kašnjenja signala ili nedovoljno prigušenog šuma.

Ovakav pristup je potreban samo ako želimo da signal vidimo u realnom vremenu, što nije potrebno u većini situacija. Za većinu slučajeva može se koristiti funkcija plot u matlabu, kao što je već prikazano u projektu.

6. Zaključak

U ovom radu prikazan je proces očitavanja analognih signala i njihove obrade u digitalnom obliku korišćenjem ESP32 mikrokontrolera. Kroz teorijski deo obrađene su osnovne karakteristike digitalne obrade signala, uključujući pojmove kao što su DFT, FFT, pojave aliasinga i kvantizacije, kao i osnovne razlike između FIR i IIR filtara. Na praktičnom nivou, implementiran je IIR filtar čiji su koeficijenti prethodno proračunati u MATLAB-u, a zatim iskorišćeni u C kodu za obradu signala u realnom vremenu na ESP32. Posebna pažnja posvećena je efikasnoj organizaciji memorije pomoću nizova za ulazne i izlazne uzorke, kao i tačnoj primeni rekurzivne formule sa povratnom spregom. Na ovaj način dobijen je funkcionalan sistem koji uspešno izdvaja korisne komponente signala i eliminiše šum. Vizuelizacija filtriranih podataka omogućena je jednostavnom Python skriptom koja u realnom vremenu prikazuje tok izlaznog signala, što je omogućilo direktnu proveru rezultata i bolju kontrolu procesa.

Iako se ESP32 pokazao kao korisna i dostupna platforma za osnovnu obradu signala, tokom rada je uočeno nekoliko potencijalnih ograničenja. Pre svega, njegov interni ADC nije idealan za precizna merenja, zbog čega bi se u budućim verzijama projekta mogla razmotriti primena eksternih ADC čipova sa boljim performansama. Takođe, u jednom od poglavlja pomenuta je mogućnost upotrebe DMA mehanizma za uzorkovanje preko I2S interfejsa, što bi značajno rasteretilo CPU i poboljšalo pouzdanost u radu sa većim količinama podataka. Pored toga, Espressif DSP biblioteka nudi dodatne optimizovane funkcije koje bi se mogle iskoristiti za proširenje sistema, kao što su brže FFT operacije, FIR/IIR filtriranje i Kalman filtri. U budućnosti, sistem se može dodatno razviti i primeniti u konkretnim IoT aplikacijama koje zahtevaju obradu senzorskih podataka u realnom vremenu, čime bi se povezao svet digitalne obrade signala sa stvarnim problemima i primenom u industriji ili istraživanjima.

7. Reference

1. Jezzamon, *Intro to Fourier Transformation* (https://www.jezzamon.com/fourier/)
2. Kalid Azad, *An Interactive Guide to the Fourier Transform* (https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/)
3. Wikipedia, *Furijeova transformacija* (https://sr.wikipedia.org/sr-el/Фуријеова_трансформација)
4. Luis Llamas, *ESP32 ADC analog input* (https://www.luisllamas.es/en/esp32-adc/)
5. Monolithic Power Systems, *Detailed Analysis of SAR ADCs*(https://www.monolithicpower.com/en/learning/mpscholar/analog-to-digital-converters/detailed-analysis-of-adc-architectures/sar-adcs)
6. Espressif Systems, *ESP-IDF ADC API Reference (v4.4.3)* (https://docs.espressif.com/projects/esp-idf/en/v4.4.3/esp32/api-reference/peripherals/adc.html)
7. GitHub, *ESP32 Incorrect Analog Values Issue #92* (https://github.com/espressif/arduino-esp32/issues/92)
8. Espressif Systems, *ESP32 Datasheet*(https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)
9. Espressif Systems, *ESP32-WROOM-32 Datasheet*(https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf)
10. Acoustiblok, *What Are the Different Types of Noise?* (https://www.acoustiblok.co.uk/what-are-the-different-types-of-noise/)
11. Unison Audio, *Low Pass Filters Explained* (https://unison.audio/low-pass-filters/)