NTNU

elsys
ELEKTRONISK SYSTEMDESIGN & INNOVASJON

TTT4280 – Lab 1

Tittel: Lab 1: Systemoppsett

Skrevet av: Siri Helene Wahl og Sunniva Rostad

Gruppe: 22

Dato: 23. april 2024

1 Sammendrag

Denne labrapporten beskriver implementasjonen av et målesystem bestående av en Raspberry Pi og fem analog-til-digital-omformere (ADC), som er designet for å samle inn, digitalisere og lagre analoge signaler. Dette systemet er svært anvendelig innen flere felt, blant annet sensorteknologi, som igjen byr på et mangfold av bruksmuligheter.

Dataen som samles inn av ADC-ene, overføres til en ekstern datamaskin for videre analyse. Ved testing med et 100 Hz sinussignal ble det observert konsistente resultater fra alle fem ADC-ene, som indikerer at målesystemet nøyaktig utfører samplingen og digitaliseringen. Videre ble det utført flere signalbehandlingsmetoder, inkludert Fast Fouriertransform (FFT), zero-padding og Hanning-vindu. FFT-transformen viste også her identiske resultater for alle ADC-ene, med en topp ved 100 Hz. Anvendelse av zero-padding og Hanning-vindu forbedret frekvensoppløsningen og reduserte støy, noe som resulterte i et mer presist frekvensspektrum.

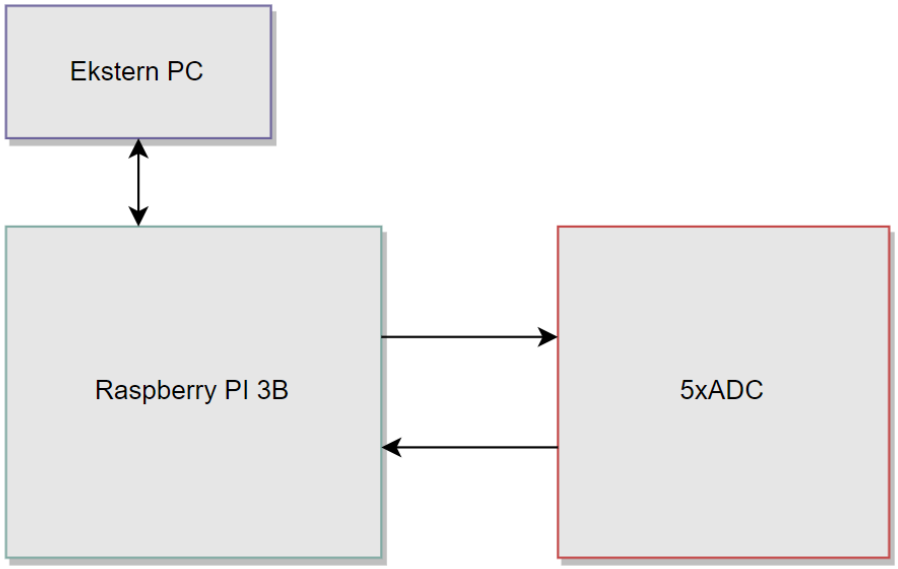
Det ble også fokusert på støyreduksjon ved spenningsforsyningen til RPi-en, og et Pi-Bridge lavpassfilter ble derfor implementert mellom ADC-ene og RPi-en. Resultatene av dette viste en målt knekkfrekvens som var litt høyere enn den teoretiske verden, men likevel hadde høy demping og god effekt for målesystemet. Det estimerte signal-støy-forholdet (SNR), basert på effektspekteret var noe dårligere enn den teoretiske maksimale SNR-verdien. Dette kan være et resultat av at systemet har blitt utsatt for flere ulike støykilder.

Samlet sett viser dette prosjektet hvordan integrert hardware og software kan brukes for å implementere et høytanvendelig målesystem. Systemet viser stor anvendelighet i sensorteknologi og andre områder som krever presis datainnsamling og signalbehandling.

Innhold

2 Introduksjon

Når man skal måle fysiske størrelser, bruker man ofte sensorer som konverterer det fysiske signalet til et analogt elektrisk signal slik at det videre kan behandles og analyseres. For å kunne anvende slik sensorteknologi, kreves et målesystem som kan sample det analoge signalet, digitalisere det, og til slutt lagre det i minnet for videre databehandling. I denne labrapporten skal det implementeres et slikt målesystem, bestående av en Raspberry Pi Single Board Computer (**RPi**) og fem analog-til-digital-omformere (**ADC**). RPi-en danner grunnlaget for systemet ved å gi et grensesnitt mellom hardware (ADC-ene) og software (ekstern PC). Målet med systemet er å fange opp og digitalisere analoge signaler for videre analyse. Et overordnet blokkdiagram av systemet er vist i figur 1.



Figur 1: Blokkdiagram som beskriver målesystemets komponenter, inkludert en Raspberry Pi, 5 ADC-er og en ekstern PC, og illustrerer hvordan dataflyten er organisert mellom disse enhetene.

3 Teori

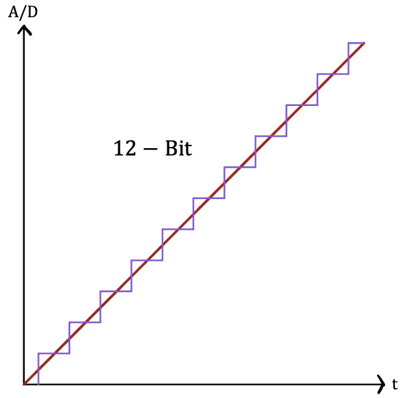
Denne delen av rapporten gir en oversikt over teori som er relevant for målesystemet. Videre er denne teorien organisert i underseksjoner.

3.1 ADC

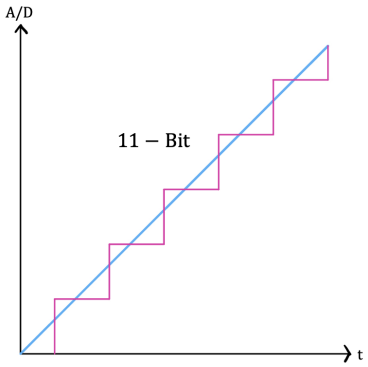
En analog-til-digital-omformer (ADC) er en elektronisk enhet som konverterer analoge signaler til digitale signaler, slik at signalene videre kan analyseres i digitale systemer, for eksempel datamaskiner [1]. ADC-ens nøyaktighet av denne konverteringen er avhengig av dens spesifikke tekniske spesifikasjoner, som for eksempel oppløsning og grensene for hvilken inngangs-

og utgangsspenning den opererer ideelt under.

Oppløsningen til en ADC definerer antall diskrete nivåer et analogt signal kan konverteres til, og er illustrert gjennom enhetens karakteristikkurve [2]. Disse nivåene, kjent som kvantiseringstrinnene, er den minste spenningsendringen ADC-en kan skille mellom. En ADC med 12-bits oppløsning, som vist i figur 2a, vil kunne representere et analogt signal med $2^{12} = 4096$ ulike verdier. I motsetning vil en ADC med 11-bits oppløsning, med karakteristikkurve vist i figur 2b, kunne representere et analogt signal med $2^{11} = 2048$ forskjellige verdier.



(a) Karakteristikkurve for en ADC med 12-bits oppløsning.



(b) Karakteristikkurve for en ADC med 11-bits oppløsning.

Figur 2: Karakteristikkurver som illustrerer oppløsningen og kvantiseringstrinnene til to ADC-er med 12- og 11-bits oppløsning, reprodusert fra figur 1-1 i [3].

Størrelsen på hvert kvantiseringstrinn, altså hvor stor spenningsendring ADC-en kan detektere, kan beregnes ved formelen gitt i 1,

$$\text{Kvantiseringstrinnstørrelse} = \frac{V_{ref}}{2^n}, \tag{1}$$

hvor n er antall bits oppløsning og V_{ref} er ADC-ens referansespenning. Størrelsen på kvantiseringstrinnene er tett knyttet til kvantiseringsfeilen, som oppstår dersom et kvantisert analogt signal har en verdi som avviker fra det faktiske signalet. Kvantiseringsfeilen er gitt ved formelen

$$\text{Maksimal kvantiseringsfeil} = \pm\frac{1}{2}\frac{V_{ref}}{2^n}, \tag{2}$$

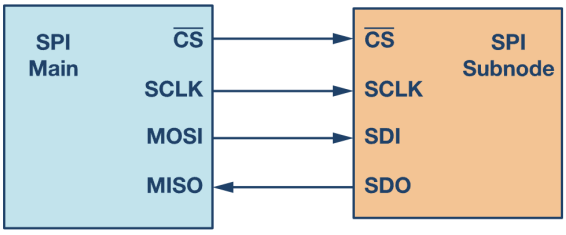
som tilsvarer halvparten av kvantiseringstrinnstørrelsen. Dette skyldes at ADC-en kan avrunde den analoge verdien til den nærmeste digitale verdien, enten opp eller ned.

Oppløsningen til ADC-en vil også være med å begrense hvor mange samples som kan foretas per sekund, altså samplingfrekvensen f_s. Dette kommer av at en ADC bruker et bestemt antall klokkesykluser k for å sample n bits. Klokkefrekvensen f_{clk} må derfor være k ganger høyere enn f_s. Dette gir formelen

$$\frac{f_{clk}}{f_s} = k. \tag{3}$$

3.2 Serial Peripheral Interface

Overføringen av data mellom RPi-en og ADC-ene kan utføres ved hjelp av kommunikasjonsprotokollen Serial Peripheral Interface (**SPI**). Denne kommunikasjonsprotokollen går ut på at data sendes over en seriell data-buss mellom en slave-node og en master-node. En illustrasjon av SPI-konfigurasjonen mellom en master og en slave er gitt i figur 3.



Figur 3: Illustrasjon av SPI-konfigurasjon, lånt fra [4].

Overføringen av data styres av kontrollsignalene SCLK (klokkesignal) og $\overline{\text{CS}}$ (Chip Select-signal). De to linjene, Master-In-Slave-Out (MISO) og Master-Out-Slave-In (MOSI), brukes til kommunikasjon mellom master- og slaveenheten, via hhv. Slave-Data-Out (SDO) og Slave-Data-In (SDI).

3.3 Direkte minnetilgang

Direkte minnetilgang (**DMA**), er en teknologi som gjør det mulig å overføre data fra en I/O-enhet til minnet uten at hovedprosessoren (**CPU**) er involvert.

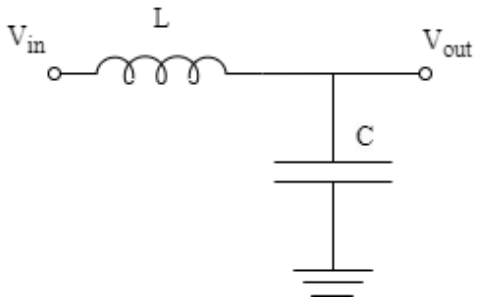
I stedet for at CPU-en skal bruke prosessorressurser på å håndtere denne dataoverføringen kan en DMA-kontroller ta av seg denne jobben. Dette er hjelpsomt da operativsystemet, **OS**, allerede har en rekke oppgaver med ulike prioriteringer som må håndteres. OS kan ikke alltid sette dataoverføringer øverst på prioriteringslisten da det er flere oppgaver som krever prosessorressurser. DMA-kontrolleren kommer dermed inn som en mellommann, og kan styre dataoverføringen mellom I/O-enheten og hovedminnet uten å belaste CPU-en. [5, s.23]

3.4 Lavpassfilter

Støy er en vanlig feilkilde i elektroniske systemer. Minimering av støy er spesielt viktig når det kommer til spenningsforsyningen til analoge kretser [6, s.14], og et lavpassfilter anvendes ofte for å løse dette. Et mulig design av et lavpassfilter er gitt i figur 4. Filteret består av en kondensator C og en spole L.

Formelen for knekkfrekvensen f_c til et lavpassfilter er gitt i likning 4,

$$f_c = \frac{1}{2\pi}\sqrt{\frac{1+\sqrt{2}}{LC}}. \tag{4}$$



Figur 4: Et mulig design av et lavpassfilter bestående av en spole L og en kondensator C. Inngangs- og utgangsspenningene er definert som henholdsvis V_{in} og V_{out}.

Denne formelen kan utledes ved å løse likningen

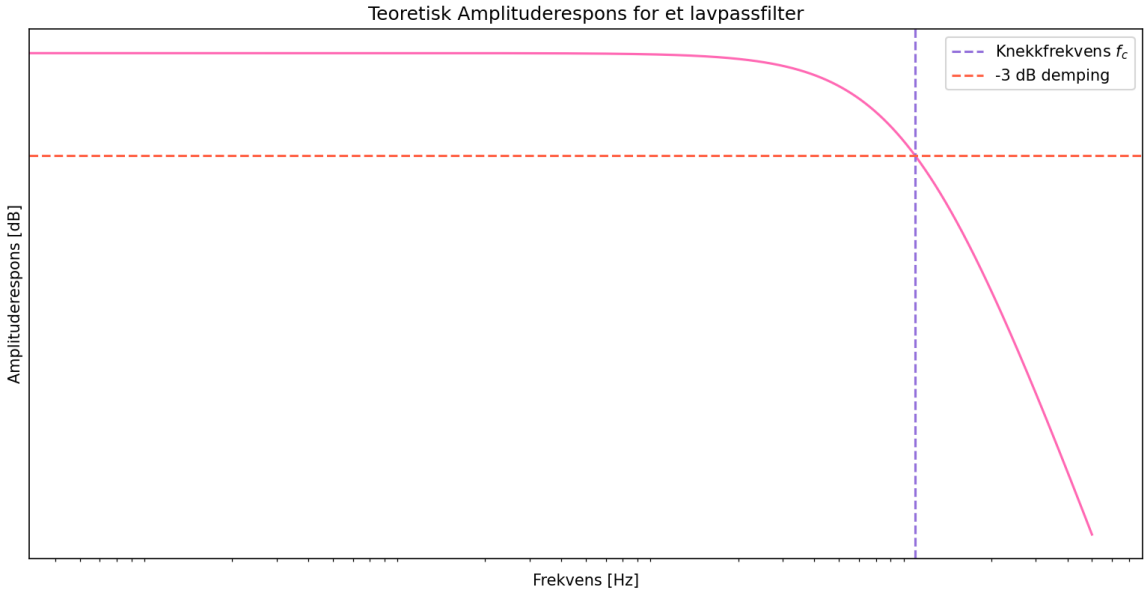
$$|H(f)| = \frac{1}{\sqrt{2}}, \tag{5}$$

altså

$$\frac{1}{|1 - (2\pi f_c)^2 LC|} = \frac{1}{\sqrt{2}}, \tag{6}$$

for en reell og positiv f_c.

Ved måling defineres knekkfrekvensen, f_c, som den frekvensen der dempingen ligger på 3dB. Dette er illustrert med stiplede linjer i figur 5.

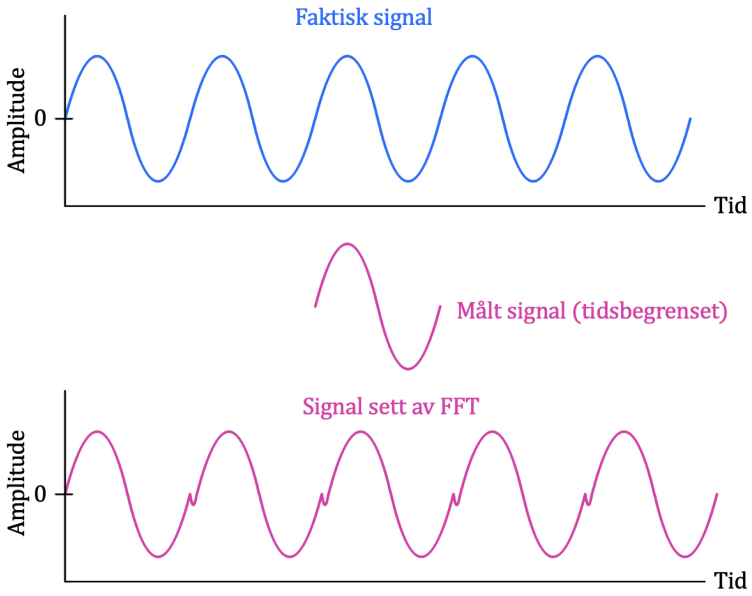


Figur 5: Illustrasjon av amplituderesponsen for et lavpassfilter, med knekkfrekvens f_c merket med stiplede linjer.

3.5 Signalbehandling

Fast Fourier Transform (**FFT**) er en algoritme som effektivt beregner fouriertransformen til et signal. Denne prosessen transformerer et signal som opprinnelig er uttrykt i tidsdomenet, til frekvensdomenet, hvor det er representert som en funksjon av vinkelfrekvens ω [7]. Dette gjør det enklere å utføre analyser og tolke frekvenskomponentene av et ønsket signal.

En utfordring med FFT er at den antar at et signal gjentar seg uendelig, når et signal i virkeligheten har en tidsbegrenset natur. FFT-algoritmen vil derfor repetere signalet utenfor måleperioden, noe som kan føre til ujevnheter ved enden av signalene, som vist i figur 6. Dette fenomenet kalles spektrallekkasje, og kan håndteres ved å anvende en Hanning-vindusfunksjon. Ved å multiplisere signalet med et Hanning-vindu før FFT utføres, kan vi redusere ujevnhetene ved enden av signalet og oppnå en jevnere spektralanalyse uten diskontinuiteter [8].



Figur 6: Illustrasjon av spektrallekkasje for FFT, reprodusert fra figur 1 i [9].

I tillegg til Hanning-vindusfunksjonen, er zero-padding en annen teknikk som kan forbedre oppløsningen av FFT. Zero-padding innebærer å utvide signalet med nuller for å øke antallet datapunkter som FFT beregnes på. Dette gjør det mulig å justere signalets lengde til en optimal verdi for FFT-prosessen, typisk den nærmeste potensen av to, som er området hvor FFT-algoritmen er mest effektiv [10]. Resultatet av å bruke zero-padding er at FFT-algoritmen lettere kan skille mellom tettliggende frekvenskomponenter, noe som gir en mer detaljert frekvensanalyse.

3.6 SNR

Signal-støy-forhold (**SNR**) er et mål på hvor mye støy et signal består av. Dette vurderes ved å se på effekten til det ønskede signalet i forhold til effekten til støyet. Altså

$$\text{SNR} = \frac{P_{signal}}{P_{støy}} [11, s.34]. \tag{7}$$

I dB blir dette

$$\text{SNR} = 10\log\left(\frac{P_{signal}}{P_{støy}}\right). \tag{8}$$

$$\text{SNR} = 10\log(P_{signal}) - 10\log(P_{støy}). \tag{9}$$

En tilnærmet verdi til effekten til et signal kan finnes ved å se på effektspekteret (PSD).

Et sinussignal som inneholder støy, vil ha et effektspekter bestående av en klar peak i sinusfrekvensen, fulgt av sidelober som oppstår som følge av støykomponentene. Da vil amplituden til hovedloben tilsvare effekten til sinussignalet, mens amplituden til den høyeste sideloben vil gi et estimat på den maksimale støyeffekten. Det er også mulig å tilnærme effekten over et bestemt frekvensintervall ved å se på gjennomsnittet av amplituden til den høyeste og laveste sideloben.

Dersom en ser på et kvantisert sinussignal med antakelse om at støyet kun er et resultat av en uniformt fordelt kvantiseringsfeil kan den maksimale SNR-verdien skrives som

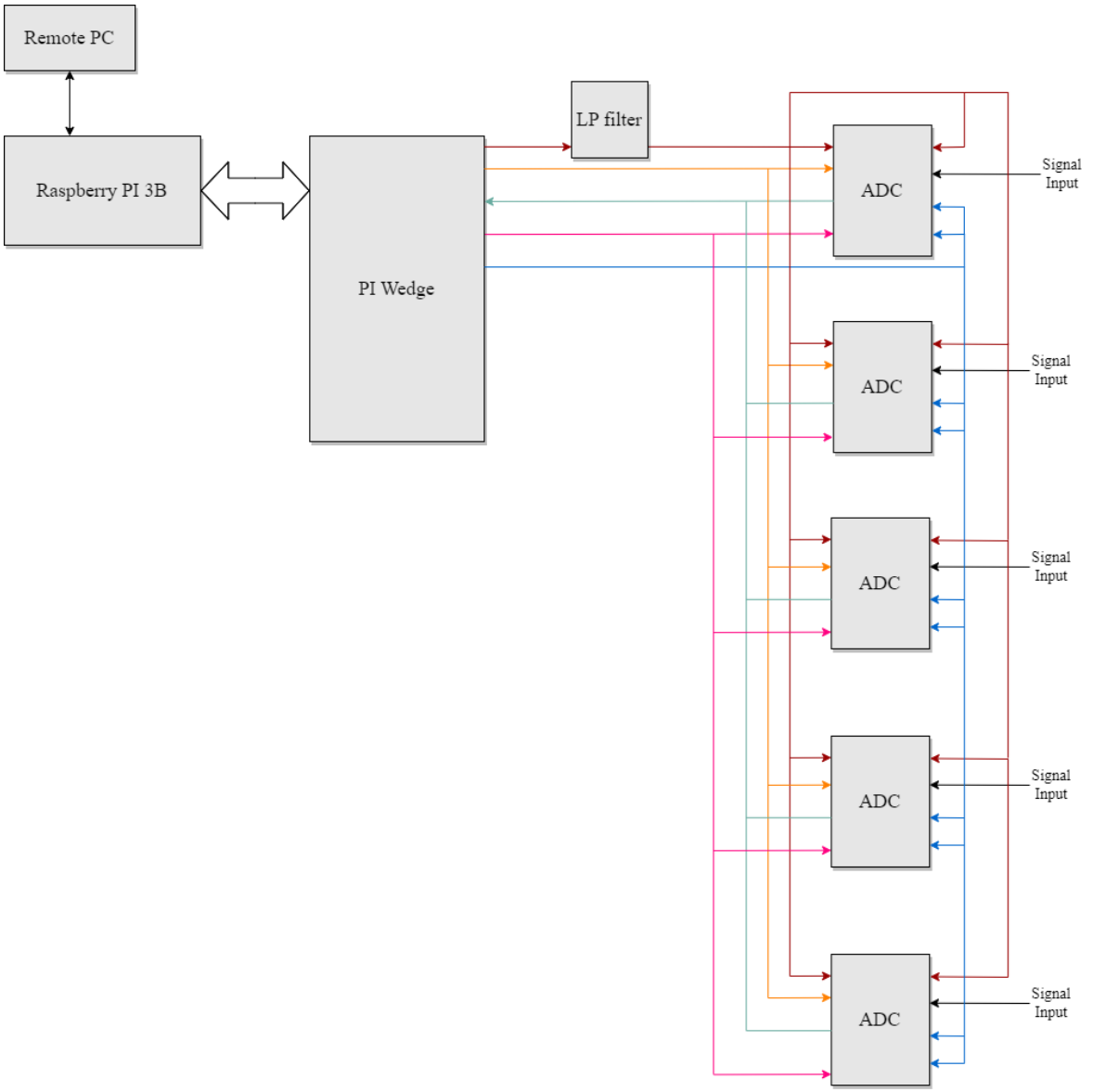
$$\text{SNR}_{\text{maks}} = 10\log\left(\frac{12(2^{N_{bits}} \cdot \Delta)^2}{8\Delta^2}\right) \text{dB}, \tag{10}$$

altså

$$\text{SNR}_{\text{maks}} \approx 1.76 + 6.02N_{bits}\text{dB} \ [12, s.7]. \tag{11}$$

4 Metode

Som forklart i seksjon 2 skal det settes opp et målesystem som bruker en Raspberry Pi (RPi) 3B+ Single Board Computer til å lese av fem ADC-er. Et fullstendig blokkdiagram er gitt i figur 7.

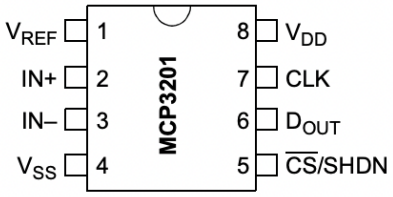


Figur 7: Blokkdiagram for målesystem basert på en RPi 3B+ og fem ADC-er. ADC-ene kobles til RPi-en med en PI wedge. Et lavpassfilter (LP-filter), er inkludert mellom PI-Wedgen og de fem ADC-ene.

Blokkdiagrammet i figur 7 viser sammensetningen og tilkoblingene i målesystemet. RPi-en er koblet sammen med de fem ADC-ene ved hjelp av en PI-Wedge. Det er også inkludert et lavpassfilter mellom PI-Wedgen og ADC-ene for å minimere støy på spenningsforsyningen fra RPi-en. Systemet vil bli beskrevet nærmere i de kommende underseksjonene.

4.1 ADC og sampling

ADC-en som brukes i prosjektet, MCP3201, har pinouts som vist i figur 8.



Figur 8: Pin-outs MCP3201 12 bit ADC, lånt fra [13].

Denne modellen har en oppløsning på 12 bit, som vil si at den kan representere et analogt signal med $2^{12} = 4096$ ulike digitale nivåer. I ADC-ens datablad er det spesifisert at inngangs- og utgangsspenninger, i forhold til V_{SS}, skal ligge i området $[-0.6\text{V}, V_{DD} + 0.6\text{V}]$. Vi benytter oss av RPi-ens forsyningsspenning $V_{DD} = 3.3\text{V}$. Fra likning 1 får vi da en kvantiseringstrinnstørrelse på

$$\text{Kvantiseringstrinnstørrelse} = \frac{3.3\text{V}}{2^{12}} = 0.8\text{mV} \tag{12}$$

og fra likning 2 en kvantiseringsfeil mellom

$$\text{Maksimal kvantiseringsfeil} = \pm\frac{0.8\text{mV}}{2} = \pm 0.4\text{mV}. \tag{13}$$

ADC-ene sampler analoge signaler og overfører dataen til RPi ved bruk av programmet **adc_sampler.c**. I denne koden definerte vi hvilke pins vi benyttet oss av, som vist i tabell 2. Denne konfigurasjonen er vist i kode 1.

```
/* RPi PINS */
#define MISO1 4
#define MISO2 5
#define MISO3 6
#define MISO4 12
#define MISO5 13

#define MOSI 10     // GPIO for SPI MOSI
#define SPI_SS 24   // GPIO for CS
#define CLK 23    // GPIO for SPI CLK
```

Kode 1: Utdrag fra **adc_sampler.c**, som viser konfigurasjonen av pins benyttet til hver ADC.

Overføringen av data fra ADC-ene til RPi gjennomføres ved kommunikasjonsprotokollen SPI, som er diskutert i seksjon 3.2. I dette målesystemet overføres data fra ADC-ene til RPi via hhv. Slave-Data-Out (SDO) og Master-In-Slave-Out (MISO).

I MCP3201s datablad [13] kan det tolkes fra figur 5.1 at for å ta én punktprøve og å overføre denne dataen til RPi, trengs det minst 16 klokkesykluser. I **adc_sampler.c** er det oppgitt en klokkefrekvens $f_{clk} = 500\text{kHz}$, og fra likning 3 gir dette en samplingsfrekvens $f_s = 31250\text{Hz}$.

For å kunne analysere og bruke dataen RPi har overført fra ADC-ene etablerer vi en tilkobling mellom datamaskinen vår og RPi sin server ved hjelp av Secure Shell (**SSH**). Videre benytter vi oss av SSH File Transfer Protocol (**SFTP**) for å overføre filer mellom RPi og vår datamaskin på en sikker og kryptert måte. For at RPi og ADC-ene skal kunne kommunisere ved bruk av DMA, som vi har belyst fordelene med i seksjon 3.3, bruker vi et utdelt C-bibliotek kalt **pigpio**.

Dataen innhentet av ADC-ene lagres i binærfiler, som direkte representerer de digitale verdiene generert gjennom konverteringsprosessen. Videre anvender vi et python-script, vist i kode 7. Denne koden importerer og leser innholdet i den binære filen og oppretter et array som inneholder all innsamlet data fra ADC-ene, slik at vi enklere kan analysere og behandle denne dataen.

4.2 Signalbehandling

For å kunne analysere de digitale signalene som ble overført fra ADC-ene til RPi, benyttet vi oss av flere signalbehandlingsmetoder. Den sentrale teknikken var FFT, nevnt i seksjon 3.5, som vi benyttet oss av for å transformere de digitale signalene til analoge signaler og videre analysere dem. Dette gjennomførte vi ved bruk av koden vist i listing 6.

Videre, for å forbedre resultatene av FFT-en, benyttet vi oss av Hanningvindu og zero-padding, som nevnt i seksjon 3.5, for å dempe signalendene og å tydeligere skille de ulike frekvenskomponentene . Vi implementerte Hanningvindu og zero-padding ved bruk av kodelinjene vist under i listing 2.

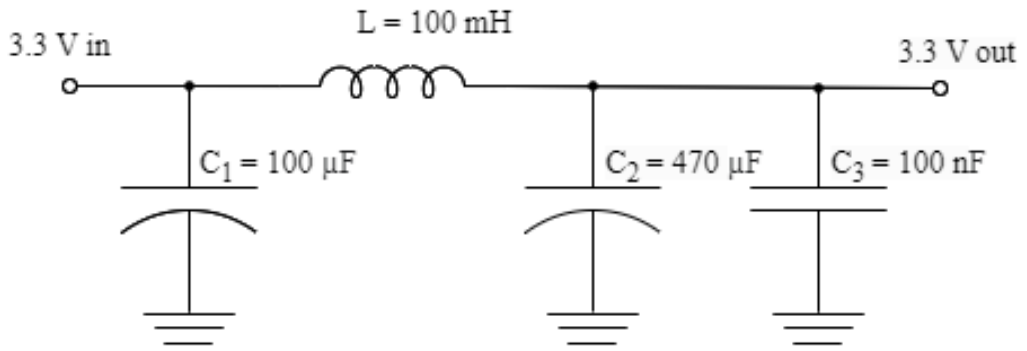
```
from scipy.signal import windows

# Hanningvindu
    window = windows.hann(len(channel_data))
    windowed_data = window * channel_data

# Zero-padding
    zero_padding_faktor = 2

    n_padded = len(windowed_data) * zero_padding_faktor
    windowed_data_padded = np.pad(windowed_data, (0, n_padded - len(
    windowed_data)), 'constant')
    channel_data_padded = np.pad(channel_data, (0, n_padded - len(channel_data
    )), 'constant')
```

Kode 2: Pythonkode for implementasjon av Hanningvindu og zero-padding.

4.3 Pi-bridge lavpassfilter

Som forklart i underseksjon 3.4 er minimering av støy på spenningsforsyningen til analoge kretser svært viktig for å unngå feil i systemet. Dette gjøres ved å designe lavpassfilteret i figur 7 som et Pi-Bridge lavpassfilter. Designet av et slikt filter er vist i figur 9.

Filteret i figur 9 består av to store stabiliserende kondensatorer C_1 og C_2, en spole L for å



Figur 9: Kretstegning for et Pi-bridge lavpassfilter. Filteret består av tre kondensatorer med C_1, C_2 og C_3 med verdier på henholdvis 100μF, 470μF og 100nF og en spole L på 100mH.

dempe høye frekvenser og en liten kondensator C_3 for å kortslutte høye frekvenser [6, s.14]. Tallverdiene til hver kretskomponent er gitt i tabell 1. En forsyningspenning på 3.3V sendes gjennom filteret, og det er ønskelig at spenningen skal forbli uendret i DC-verdi, men at eventuell signalstøy skal minimeres.

Tabell 1: Tallverdier for kretskomponentene brukt i Pi-bridge lavpassfilteret i figur 9.

Kretskomponent	Tallverdi
C_1	100 μF
C_2	470 μF
C_3	100 nF
L	100 mH

Formelen for knekkfrekvensen f_c til et lavpassfilter, gitt i likning 4, ble introdusert i seksjon 3.4. Her settes

$$C = C_2 + C_3. \tag{14}$$

Gitt dette, blir utrykket for knekkfrekvensen

$$f_c = \frac{1}{2\pi}\sqrt{\frac{1+\sqrt{2}}{L \cdot (C_2 + C_3)}}. \tag{15}$$

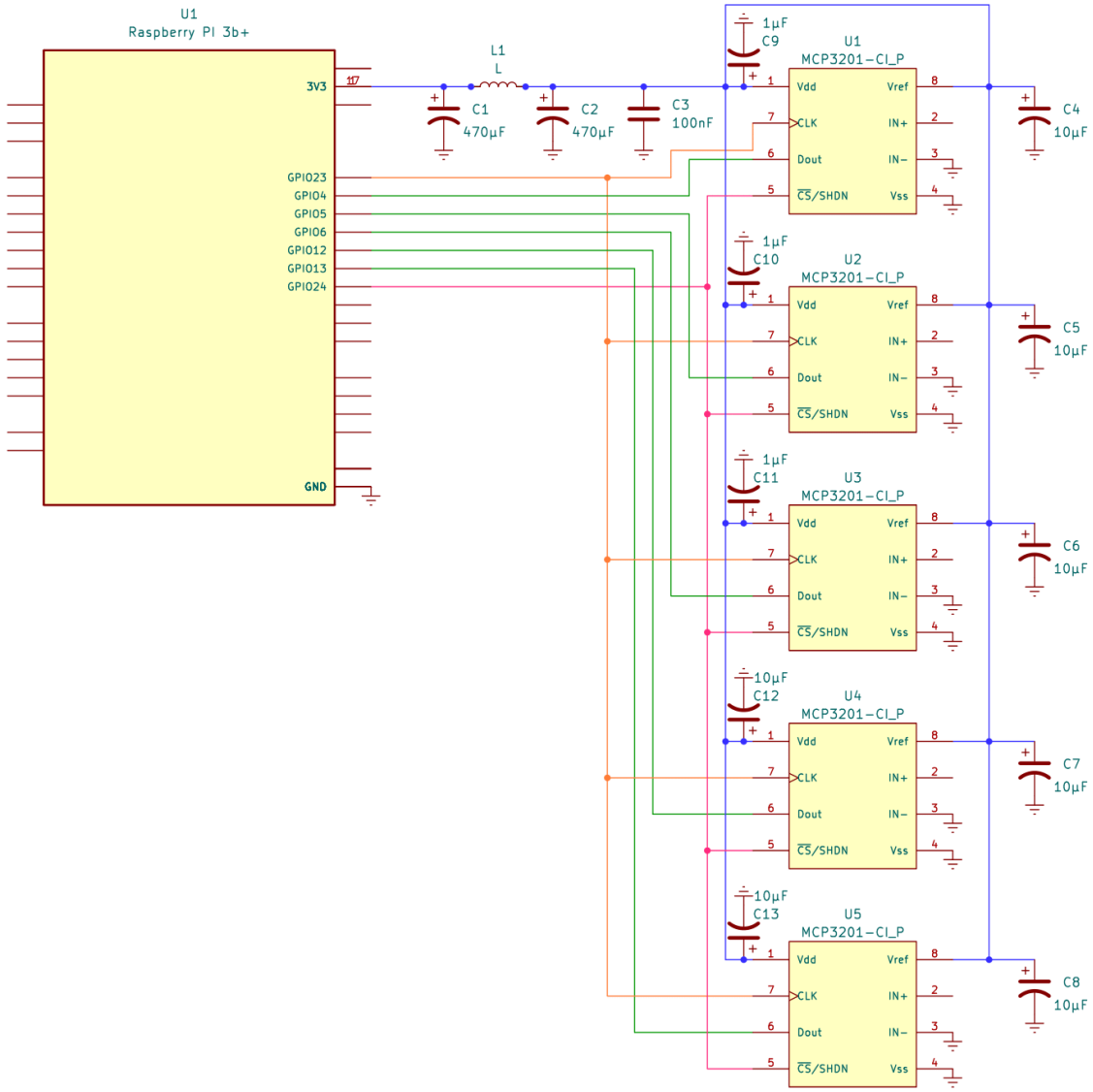
Med verdier gitt i tabell 1, blir den teoretiske knekkfrekvensen

$$f_c \approx 36.07\text{Hz}. \tag{16}$$

Ved testing av lavpassfilteret fjernes C_1, da denne kun er ment for å stabilisere forsyningspenningen på inngangen. For å analysere frekvensresponsen brukes en Analog Discovery 2 og WaveForms sin network-funksjon. Dette skal gi et bodeplot med tilsvarende karakteristikk som bodeplottet gitt i figur 5. Den målte knekkfrekvensen må også sammenliknes med den teoretiske knekkfrekvensen på 36.07 Hz.

4.4 Krets

Kretsdiagrammet i figur 10 viser hvordan ADC-ene er koblet til Raspberry Pi-en, samt lavpassfilteret og avkoblingskondensatorer.



Figur 10: Kretsdiagram for oppkobling av Raspberry Pi og fem ADC-er. Et Pi-bridge lavpassfilter er plassert etter forsyningsspenningen fra Rpi-en. To avkoblingskondensatorer er koblet til V_{dd} og V_{ref} på alle ADC-ene.

En oversikt over koblingen mellom GPIO-pinsene på RPi-en og ADC-ene er gitt i tabell 2.

Som forklart i seksjon 4.3 er et Pi- Bridge lavpassfilter inkludert for å minimere støy. Dette filteret er plassert etter spenningsforsyningen ($V_{dd} = 3.3$V) til RPi-en. I tillegg er to avkoblingskondensatorer plassert på mellom V_{dd} og GND, og V_{ref} og GND på alle ADC-ene.

Tabell 2: Oversikt og koblingen mellom Raspberry Pien og ADC-ene.

RPi	ADC
3.3V	V_{dd} og V_{ref}
GPIO 23	CLK
GPIO 4, 5, 6, 12, 13	D_{out}
GPIO 24	$\overline{CS}$
GND	IN- og V_{ss}

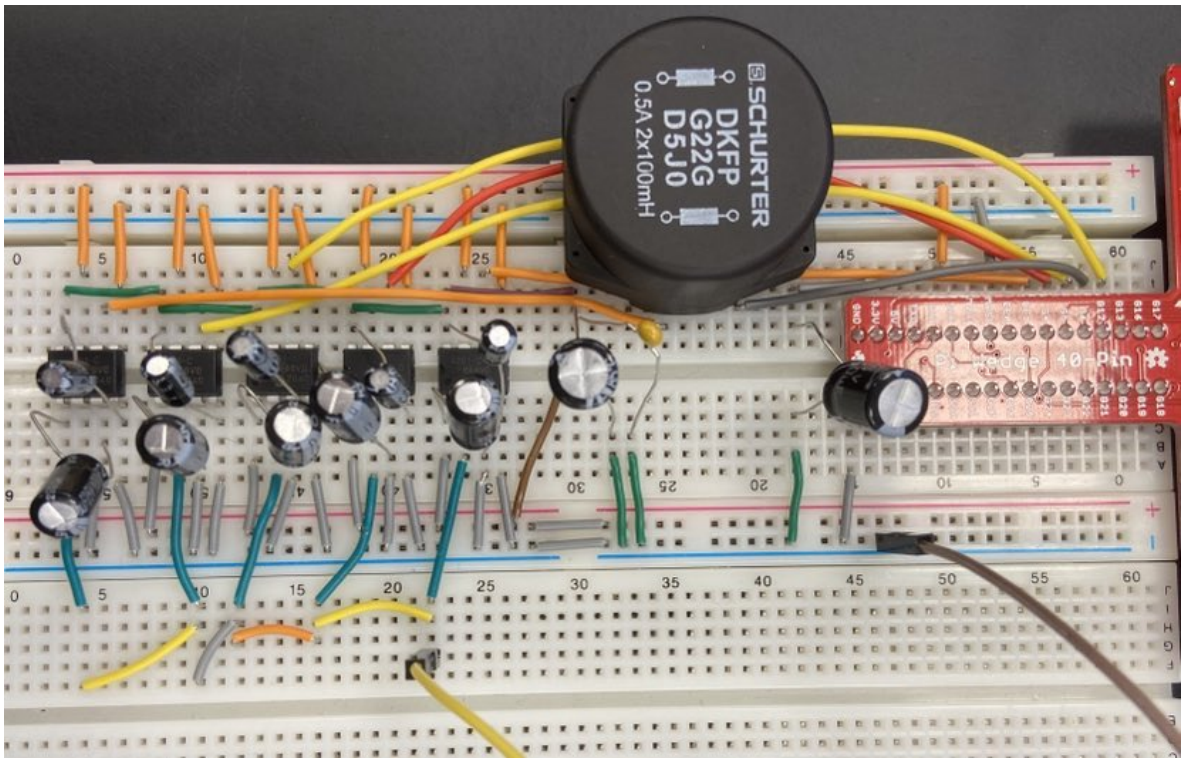
Avkoblingskondensatorene har verdier på henholdsvis 1 μF og 10 μF og brukes for redusere støy. Denne sammenkoblingen er vist med mørkeblå ledninger i figur 10.

De digitale signalene omgjort av ADC-ene overføres til en ekstern PC av en Raspberry Pi ved bruk av Serial Peripheral Interface (SPI) og Master In Slave Out (MISO) konfigurasjon, som er introdusert i seksjon 3.2. I dette tilfellet opptrer RPi-en som en master og ADC-ene som slaver. Dataen sendes fra D_{out} utgangen på de fem ADC-ene til den tilhørende GPIO pinnen på RPi-en. Databussene er vist i grønt i figur 10.

ADC-enhetene trenger også et klokkesignal for å synkronisere dataoverføringen til RPi-en. CLK-inngangen på ADC-ene kobles til GPIO 23 på RPi-en. Dette er illustrert med oransje ledninger i figur 10. Klokkefrekvensen blir satt til 500kHz. Gitt av likning 3 blir samplingfrekvensen f_s lik 31250 Hz.

Som forklart i seksjon 3.2 er det også behov for et Chip Select-signal, $\overline{CS}$. Denne kontrollinjen er vist i rosa i figur 10.

Det realiserte systemet er vist i figur 11.

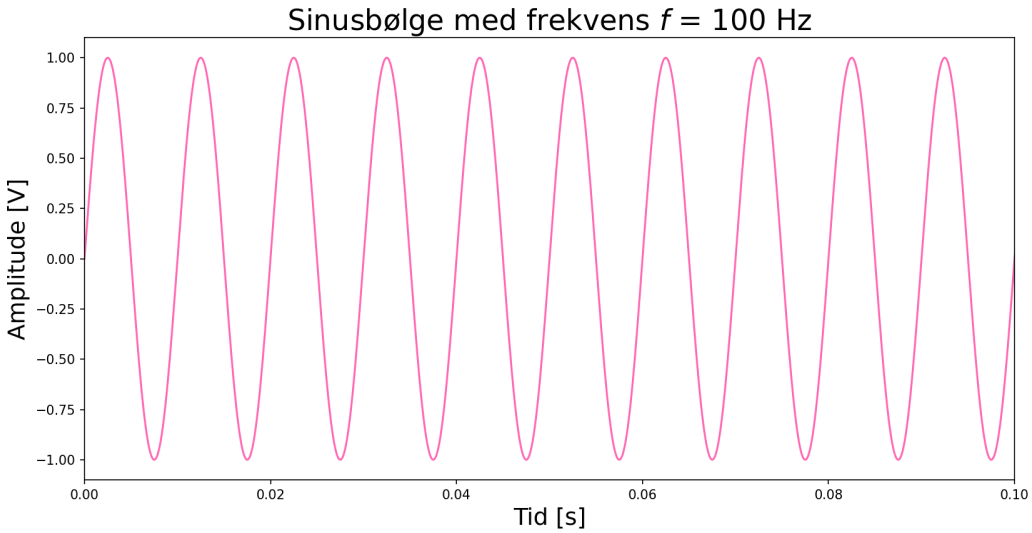


Figur 11: Fysisk oppkobling av målesystemet.

4.5 Testing av målesystemet

Ved testing av målesystemet sendes et sinussignal inn på IN+-inngangen til ønsket ADC. Siden IN- er koblet til jord vil det ikke være mulig å måle det negative signalet, og det vil derfor være optimalt å legge til en DC-offset på testsignalet.

Det brukes et sinussignal med en frekvens $f = 100$ Hz, amplitude $A = 1$ V og offset på 1V for å teste målesystemet. Dette testsignalet er illustrert i figur 12.



Figur 12: Illustrasjon av testsignal. Testsignalet er en sinusbølge med frekvens $f = 100$ Hz. Amplitude i Volt [V] er plottet mot tid i sekunder [s].

Frekvensspekteret til et slikt signal skal kun inneholde en topp i sinusfrekvensen $f = 100$ Hz, ved et rent teoretisk tilfelle.

Når det samplede signalet skal plottes må dataene multipliseres med kvantiseringstrinnsstørrelsen fra likning 1. Dette er vist i listing 3.

```
delta = 0.8e-3
channel_data = data[:,3] #ADC4
channel_data *= delta
t = np.arange(0,1, 1/31250)
```

Kode 3: Kodesnutt som viser fremgangsmåten for å justere data med kvantiseringstrinnsstørrelsen.

4.6 SNR

Som forklart i seksjon 3.6 kan signal-støy-forholdet brukes til å vurdere kvaliteten av et digitalisert signal. Dersom en kun vurderer effekten av kvantiseringsstøy vil den maksimale SNR verdien bli

$$\text{SNR}_{\text{maks}} \approx 1.76 + 6.02 \cdot 12\text{dB}, \tag{17}$$

altså

$$\text{SNR}_{\text{maks}} \approx 74. \tag{18}$$

Det vil si at det målte signalet ikke kan ha en høyere SNR-verdi enn 74. For å finne et estimat på den reelle SNR-verdien plottes effektspekteret til signalet. Kode for hvordan dette gjennomføres er gitt i listing 4.

```
import numpy as np

    psd_channel_data = np.abs(fft_data_channel)**2
```

```
4
5       psd_db = 20*np.log10(psd_channel_data) #dB konvertering
6       psd_norm = psd_db - np.max(psd_db) #Normalisert data
```

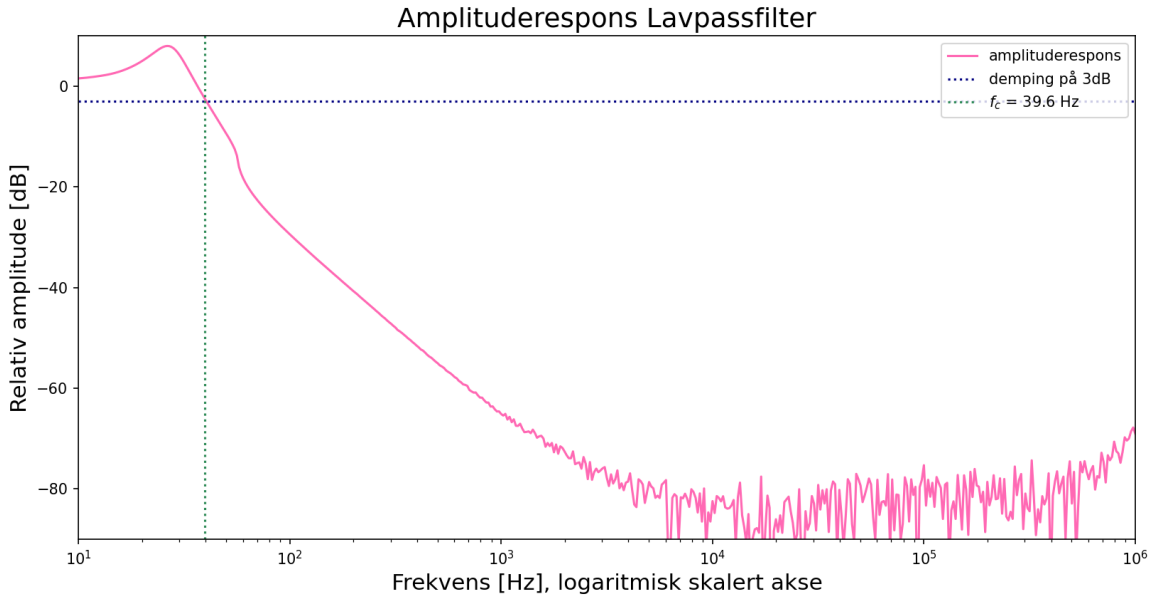
Kode 4: Pythonkode for plotting av effektspekter for å finne estimatet av den reelle SNR-verdien.

5 Resultater

Denne seksjonen presenterer resultatene fra testingen av målesystemet beskrevet i seksjon 4. Seksjonen vil i hovedsak ha fokus på kvaliteten av det samplede signalet, både når det kommer til frekvensnøyaktighet og støy. Videre diskusjon av resultatene kommer i seksjon 6.

5.1 Pi-Bridge Lavpassfilter

Som beskrevet i seksjon 4.3 blir Pi-Bridge lavpassfilteret analysert gjennom en spektralanalyse. Det resulterende bodeplottet er gitt i figur 13



Figur 13: Amplituderespons for Pi-Bridge Lavpassfilteret. Bodeplottet viser filterets frekvensrespons, med knekkfrekvensen $f_c = 39.6$ Hz markert ved 3dB demping, indikert som skjæringspunktet mellom grønn og blå stiplet linje.

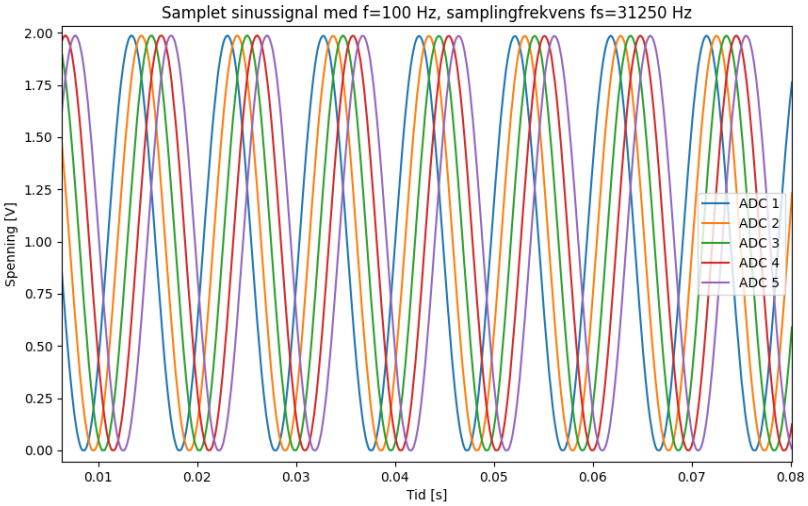
I figuren markerer den blå stiplede linjen en demping på 3dB, mens den grønne representerer den observerte knekkfrekvensen f_c på 39.6 Hz. Fra likning 16, ser vi at den målte knekkfrekvensen er noe høyere enn den teoretiske gitt ved 36.07 Hz.

Fra figur 13 er det tydelig at filteret likner et teoretisk lavpassfilter fremstilt i figur 4. Filteret har en bratt reduksjon i forsterkning etter knekkfrekvensen, og alle frekvenser over omtrent

100 Hz har en demping på minimum 30dB. Siden filteret består av en spole og en kondensator vil vi få en resonnerende effekt før grafen synker. Dette vil føre til en liten forsterkning av frekvenser rundt resonnansfrekvensen.

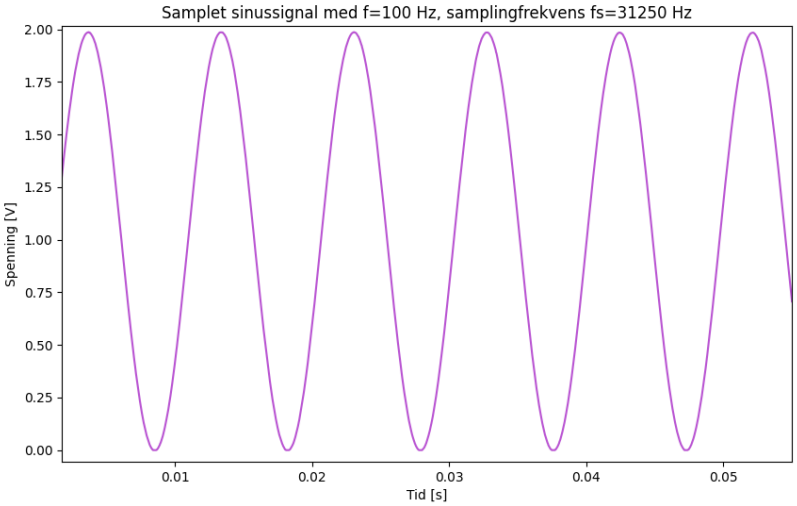
5.2 Sampling av sinussignal

Resultatet av samplingprosessen forklart i seksjon 4.1 er presentert i figur 14. Her vises det digitaliserte signalet fra alle ADC-ene med en faseforskyvning for å synliggjøre alle signalene. Det er tydelig fra figuren at alle ADC-ene ga et tilnærmet likt resultat.



Figur 14: Sinussignal med $f = 100$ Hz samplet og digitalisert av de fem ADC-ene ved samplingsfrekvens $f_s = 31250$ Hz.

Et plot av det digitaliserte signalet fra ADC 1 alene er gitt i figur 15.

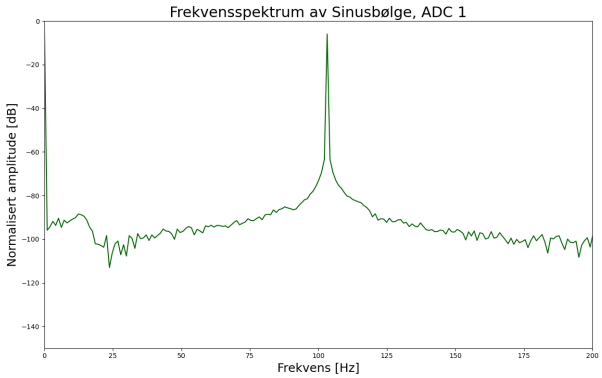


Figur 15: Sinussignal med $f = 100$ Hz samplet og digitalisert av ADC 1 ved samplingsfrekvens $f_s = 31250$ Hz.

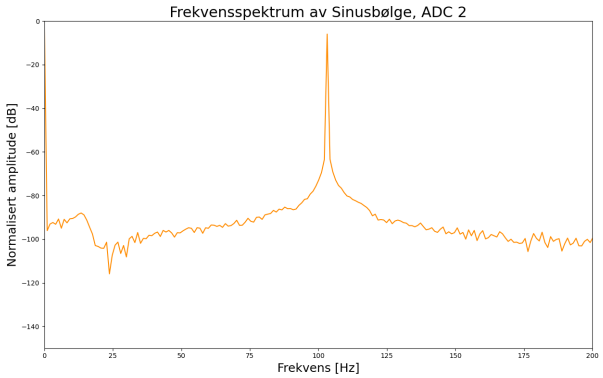
Det samplede signalet er tilsynelatende identisk inngangssignalet, vist i figur 12. For å analysere signalet videre er det nødvendig å gå over til frekvensdomenet.

5.3 Signalbehandling

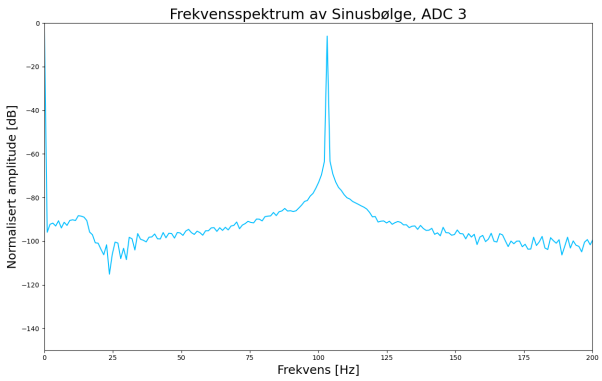
Som beskrevet i seksjon 4.2, brukte vi FFT for å konvertere fra digitale til analoge signaler. Resultatene av denne transformen for alle ADC-ene er vist i figur 16. Fra disse resultatene ser vi at amplituderesponsene for de fem ADC-ene er identiske, alle med en topp i nærheten av $f = 100$ Hz. En mer presis måling gir at spektertoppen ligger i f =103 Hz, altså 3 Hz fra teoretisk verdi.



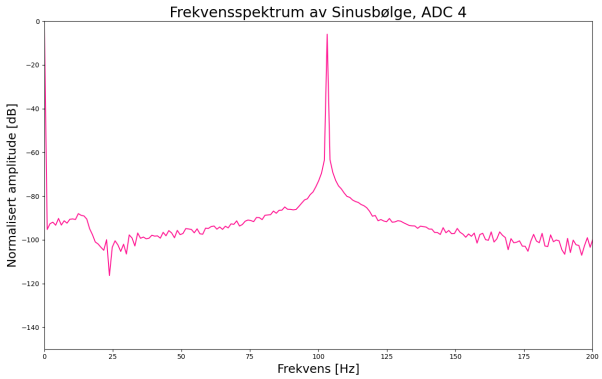
(a) Frekvensspektrum av sinussignal samplet av ADC 1.



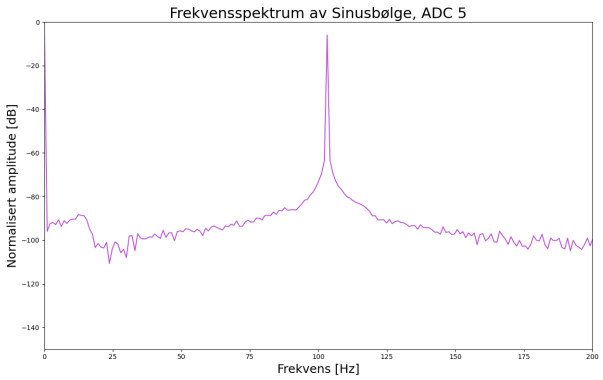
(b) Frekvensspektrum av sinussignal samplet av ADC 2.



(c) Frekvensspektrum av sinussignal samplet av ADC 3.



(d) Frekvensspektrum av sinussignal samplet av ADC 4.

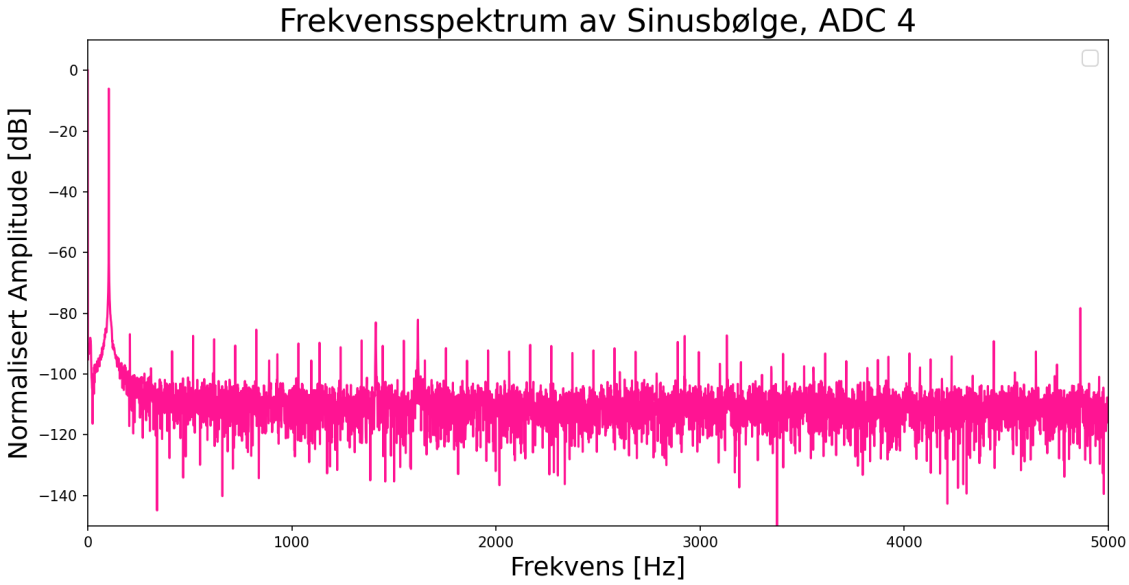


(e) Frekvensspektrum av sinussignal samplet av ADC 5.

Figur 16: Frekvensspektrum til det digitaliserte sinussignalet med frekvens $f = 100$ Hz samplet av hhv. ADC 1-5.

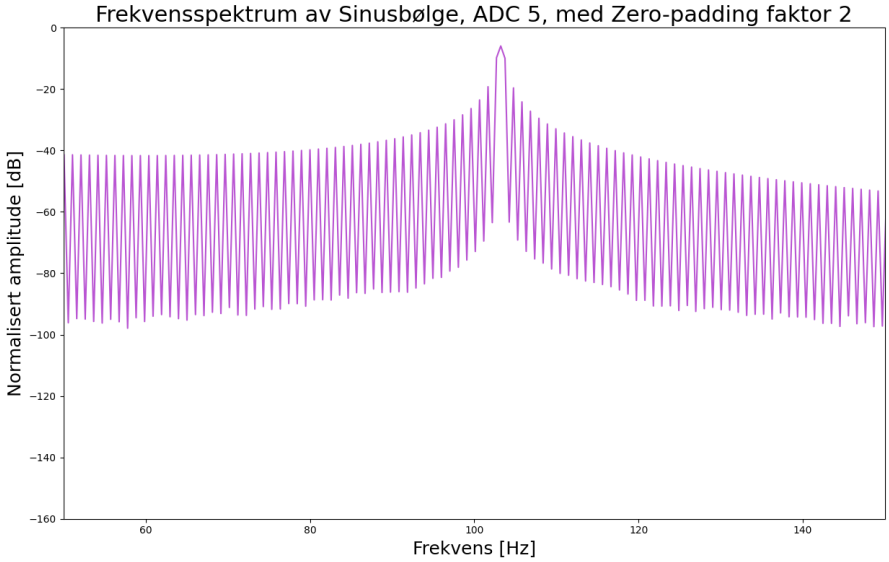
I kontrast med et teoretisk sinussignal inneholder spekteret noe støy. Dette støyet vises tydeligere i plottet i figur 17 med en utvidet frekvensakse. Det er tydelig at spekteret har frekvenskomponenter utenom 100 Hz som et resultat av støy. Basert på en en visuell vurdering

av spekteret har det et støygulv på omtrent -100 dB.

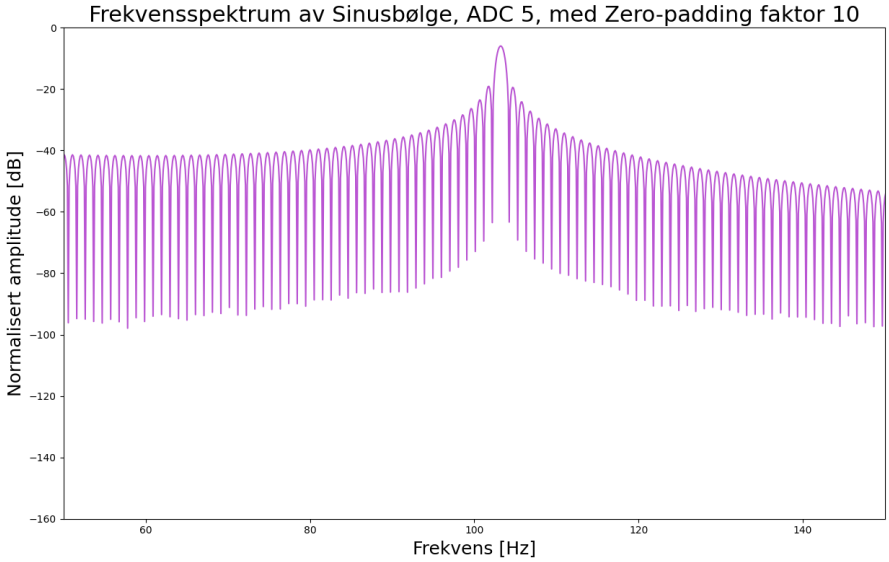


Figur 17: Frekvensspekter tilhørende ADC 4.

Videre anvendte vi zero-padding og Hanning-vindu på signalet beregnet av FFT-en. Vi utførte zero-padding med to ulike faktorer. Resultatet av zero-padding med faktor = 2 og faktor = 10 er vist i hhv. figur 18a og 18b. Resultatet etter bruk av Hanning-vindu er vist i figur 19. Det er kun inkludert plot for ADC 5, da alle plot er identiske, som vist i plotet av frekvensspektrene.



(a) Frekvensspektrum av sinussignal $f = 100$ Hz, etter anvendelse av zero-padding med zero-padding faktor = 2.

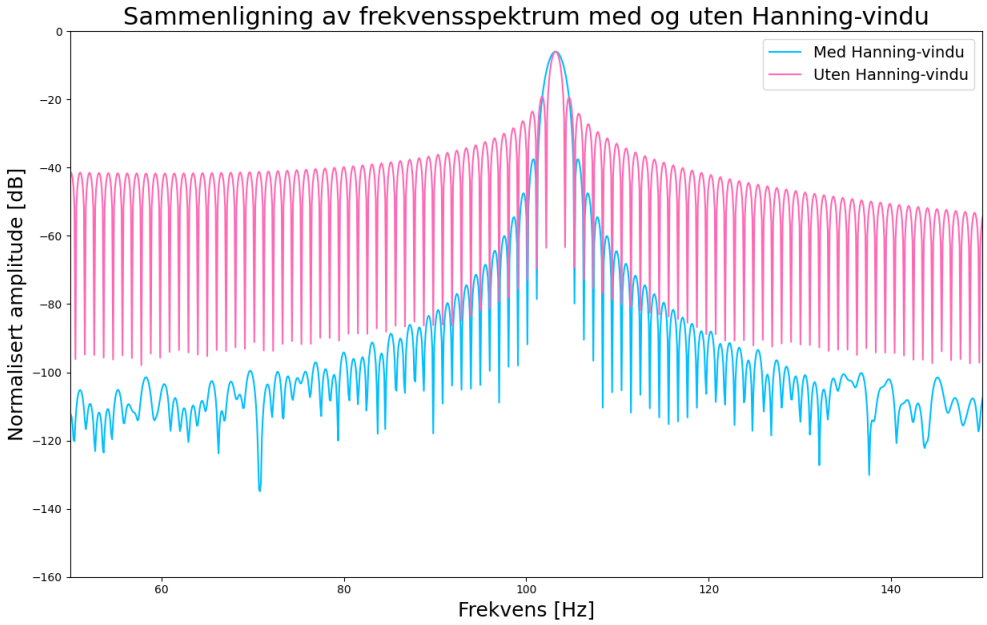


(b) Frekvensspektrum av sinussignal $f = 100$ Hz, etter anvendelse av zero-padding med zero-padding faktor = 10.

Figur 18: Frekvensspektrum av sinussignal $f = 100$ Hz, samplet av ADC 5, etter anvendelse av zero-padding med ulike zero-padding faktorer.

Vi ser fra figur 18 at signalet viser en tydelig forbedring i frekvensoppløsningen sammenliknet med frekvensspekteret av signalet uten zero-padding i figur 16a til 16e.

Det er også tydelig at en økning i zero-padding faktor også bidrar med å øke kvaliteten på spekteret. Med en faktor på 2 er sidelobene spisse, og når faktoren økes til 10 blir sidelobene rundere slik som teoretisk forventet.

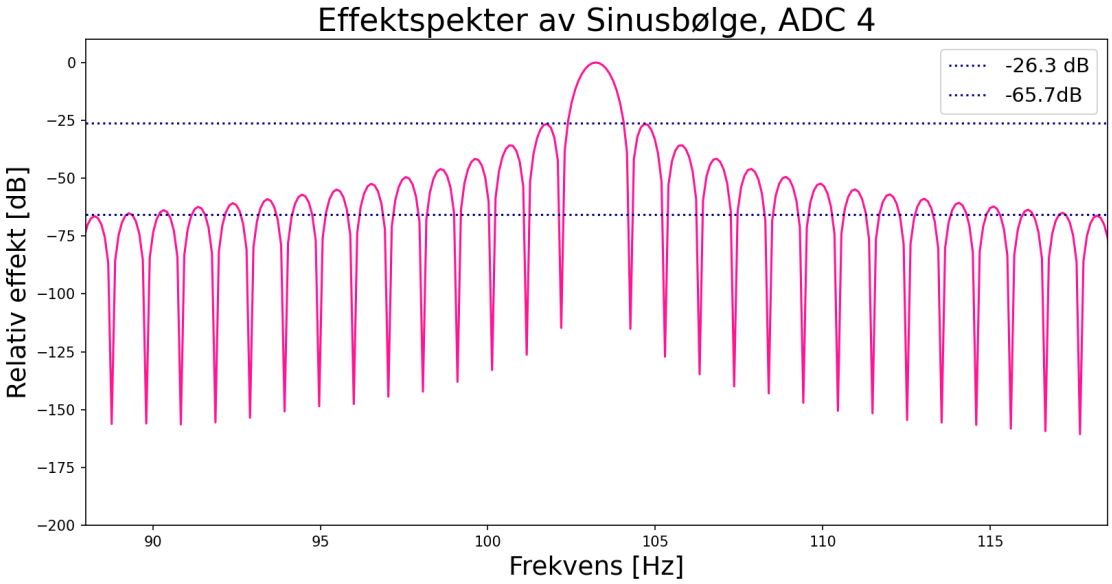


Figur 19: Sammenligning av frekvensspektrum til et sinussignal med $f = 100$ Hz, med og uten Hanning-vindu. Det er også benyttet zero-padding med en faktor på 2.

Figur 19 viser at ved bruk av Hanning-vindu ser vi tydeligere toppen ved $f = 100$ Hz, og at støykomponentene er betydelig dempet. Helhetlig observerer vi at ved bruk av Hanning-vindu får vi et frekvensspektrum som er mer fokusert rundt den ønskede frekvensen, dette samsvarer bedre med det teoretiske spekteret til sinussignalet ADC-ene samplet. Det er også tydelig at hanning-vinduet påvirker bredden på hovedloben. Denne er blitt bredere enn tidligere.

5.4 SNR

Som forklart i seksjon 3.6 kan signal-støy forholdet regnes ut ved å se på effekten til det faktiske signalet i forhold til effekten til støysignalet. Effektspekteret til det digitaliserte signalet hentet fra ADC 4 er gitt i figur 20. Her er den høyeste og laveste sideloben markert med stiplede blå linjer.



Figur 20: Effektspekter av samplet sinussbølge med frekvens f lik 100 Hz. De stiplede linjene markerer dB-verdier brukt til å estimere SNR.

Basert på dette plottet og likning 9 er den minimale SNR verdien på 26.3. Dersom en ser på et større intervall rundt 100 Hz vil SNR-verdien raskt bli bedre. Intervallet fra omtrent 88Hz til 118Hz gir

$$\text{SNR} = \frac{26.3 + 65.7}{2}. \tag{19}$$

Altså

$$\text{SNR} = 46. \tag{20}$$

Som forventet er begge verdiene mindre enn den maksimale SNR-verdien lagt frem i seksjon 3.6.

6 Diskusjon

Basert på resultatene presentert i seksjon 5 klarer målesystemet å sample og overføre data fra ADCene til en ekstern PC ved hjelp av en RPi. Etter å ha anvendt ulike signalbehandlingsmetoder og analysert signalene, er det tydelig at det målte signalet er påvirket av noe støy.

Den estimerte SNR-verdien til signalet er basert på effektspekteret og avhenger av hvilket frekvensintervall som blir vurdert. Dette kan gjøre det vanskelig å si om SNR-verdien er bedre enn den teoretiske maksimale SNR-verdien SNR_{maks}. Dersom man ser på et stort frekvensintervall går SNR-verdien nærmere SNR_{maks}. Hvorvidt dette er tilstrekkelig avhenger av hvilket frekvensintervall som er kritisk for den spesifikke applikasjonen systemet skal brukes i.

Det at SNR-verdien fortsatt er et stykke unna den maksimale SNR-verdien SNR_{maks} på 76 kan komme av at signalet også kan være påvirket av andre støykilder enn kvantiseringsstøy. For å ta stilling til mulige forbedringer vil det være behov for å vite hvilke andre støykilder som påvirker signalet, og vil avhenge av applikasjonen målesystemet skal brukes i. En fellesfaktor for mange støykilder er at de genererer stokastisk støy, og effekten av dette støyet kan minimeres ved bruk av et Wienerfilter.

Fra frekvensspekteret gitt i figur 16 ser vi at frekvensen f er forskjøvet med omtrent 3 Hz. Dette kan blant annet komme av aliasing. Dersom det er fare for at støyfrekvensene er større enn Nyquist-grensen, $\frac{f_s}{2}$, kan det være optimalt å inkludere et anti-alias filter på inngangene til ADC-ene.

7 Konklusjon

I denne rapporten er det blitt presentert et målesystem som benytter seg av en Raspberry Pi 3b og fem ADC-er. Dataoverføringen mellom ADC-ene og Rpi-en ble utført ved bruk av en SPI-protokoll, og for å maksimere ytelsen av RPi-en ble også direkte minnetilgang tatt i bruk.

For å teste systemet ble det sendt inn et sinussignal med frekvens f =100 Hz. Resultatet av denne målingen viste at systemet var i god stand til å ta målinger av et analogt signal og overføre de til en ekstern PC for videre analyse i frekvensdomenet. Anvendelse av FFT på signalet viste en peak i f =103 Hz, altså 3 Hz fra den teoretiske verdien. Signalet hadde også frekvenskomponenter ved andre frekvenser, som et resultat av støy. Den estimerte SNR-verdien, basert på effektspekteret, ble målt til omtrent 26.3. Om dette er et tilstrekkelig resultat avhenger av hva målesystemet skal brukes til, men i sammenlikning med den maksimale SNR-verdien på 76, er den ganske lav. På en annen side vil en mindre SNR-verdi kunne estimeres ved å se på et større frekvensintervall.

Signalbehandlingsteknikker som zero-padding og Hanning-vindu har også blitt undersøkt. Dette har bidratt til å forbedre frekvensoppløsningen og redusere spektrallekkasje. Zero-paddingen økte frekvensoppløsningen til spekteret og Hanning-vinduet dempet frekvenskomponentene som oppsto på grunn av støy.

Referanser

[1] Larsen, B. B.: *analog-digital-omformer [Internett].* 4.april 2022 [hentet februar 2024]. https://snl.no/analog-digital-omformer.

[2] Arar, S.: *ADC Resolution vs. Accuracy—Sub-range ADCs, Two-step ADCs, and TUE [Internett].* 8.januar 2023 [hentet februar 2024]. https://www.allaboutcircuits.com/technical-articles/adc-resolution-vs-adc-accuracy-subrange-adc-two-step-adc-and-total-unadjusted-error/.

[3] EEWeb: *AVR121: Enhancing ADC resolution by oversampling [Internett].* 23. november 2010 [hentet februar 2024]. https://www.eeweb.com/wp-content/uploads/articles-app-notes-files-enhancing-adc-resolution-1290444120.pdf.

[4] Dhaker, P.: *Introduction to SPI Interface [Internett].* September 2018 [hentet februar 2024]. https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf.

[5] Eide, Egil: *Labforelesning1: Målesystem.* 15. januar 2024 [hentet januar 2024].

[6] NTNU: *User Guide for the Laboratory.* 8. januar 2024 [hentet februar 2024].

[7] Andersen, P. B.: *FFT [Internett].* 21. januar 2023 [hentet februar 2024]. https://snl.no/FFT.

[8] Ena, A.: *Fast Hann Function implementation [Internett].* 17. september 2022 [hentet februar 2024]. https://medium.com/@lalesena/what-you-should-know-about-hann-function-790270460a75.

[9] Viswanathan, M.: *FFFT and Spectral Leakage [Internett].* 20. januar 2011 [hentet februar 2024]. https://www.gaussianwaves.com/2011/01/fft-and-spectral-leakage-2/.

[10] Hilbert, S.: *FFT Zero Padding [Internett].* 22. april 2013 [hentet februar 2024]. https://www.bitweenie.com/listings/fft-zero-padding/.

[11] Eide, Egil: *Støy og interferens 1.* 12. februar 2024 [hentet februar 2024].

[12] Eide, Egil: *Forelesning 5 Måleusikkerhet.* 2. februar 2024 [hentet februar 2024].

[13] Microchip Technology Inc: *MCP3201 [Internett]*, September 1998 [oppdatert august 2017; hentet februar 2024]. https://ww1.microchip.com/downloads/en/DeviceDoc/21290F.pdf.

A Kode

```

import numpy as np
import sys
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq, fftshift
from scipy.signal import windows, detrend

#---------------------------------------#

def raspi_import(path, channels=5):

    with open(path, 'r') as fid:
        sample_period = np.fromfile(fid, count=1, dtype=float)[0]
        data = np.fromfile(fid, dtype='uint16').astype('float64')
        data = data.reshape((-1, channels))

    sample_period *= 1e-6
    return sample_period, data

if __name__ == "__main__":

    sample_period, data = raspi_import(sys.argv[1] or 'foo.bin')

    delta = 0.8e-3
    channel_data = data[:,3] #ADC4
    channel_data *= delta
    t = np.arange(0,1, 1/31250)

    plt.plot(t, channel_data, color="deeppink")
    plt.title('Samplet sinusbolge', fontsize = 22)
    plt.xlabel('Tid [s]',fontsize=18)
    plt.ylabel('Amplitude [V]',fontsize=18)
    plt.xlim(0, 0.1)
    plt.legend(fontsize = 14)
    plt.show()
```

Kode 5: Python kode for plot av samplet sinussignal.

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import windows

def raspi_import(path, channels=5):

    with open(path, 'r') as fid:
        sample_period = np.fromfile(fid, count=1, dtype=float)[0]
        data = np.fromfile(fid, dtype='uint16').astype('float64')
        data = data.reshape((-1, channels))

    sample_period *= 1e-6
    return sample_period, data

if __name__ == "__main__":
```

```
    sample_period, data = raspi_import(sys.argv[1] or 'foo.bin')

    channel_data = data[:,0] #ADC 1

    # FFT
    fft_data = fft(channel_data)

# Frekvensakse
    n = len(channel_data)
    frekvenser = fftfreq(n, d=sample_period)

# Amplitude
    magnitude = np.abs(fft_data)

    #sx = np.square(magnitude)
# i dB
    magnitude_db = 20 * np.log10(magnitude)
# Normalisert amplitude
    magnitude_db_normalisert = magnitude_db - np.max(magnitude_db)

# Plot
    plt.figure(figsize=(10, 6))
    plt.plot(frekvenser[:n//2], magnitude_db_normalisert[:n//2])
    plt.xlabel('Frekvens (Hz)')
    plt.ylabel('Amplitude')
    plt.xlim(0,200)
    #plt.ylim(0,400)
    plt.title('Frekvensspektrum av sinusbolge')
    plt.show()
```

Kode 6: Python kode for plot av frekvensspektrum.

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from scipy.signal import windows

#--------------------------------------#

def raspi_import(path, channels=5):

    with open(path, 'r') as fid:
        sample_period = np.fromfile(fid, count=1, dtype=float)[0]
        data = np.fromfile(fid, dtype='uint16').astype('float64')
        data = data.reshape((-1, channels))

    sample_period *= 1e-6
    return sample_period, data

if __name__ == "__main__":

    sample_period, data = raspi_import(sys.argv[1] or 'foo.bin')

    channel_data = data[:,4] # ADC 5

    #Hanning window
    window = windows.hann(len(channel_data))
```

```
27      windowed_data = window * channel_data
28
29      #Zero-padding
30      zero_padding_factor = 2
31
32      n_padded = len(windowed_data) * zero_padding_factor
33
34      windowed_data_padded = np.pad(windowed_data, (0, n_padded - len(
        windowed_data)), 'constant')
35
36      channel_data_padded = np.pad(channel_data, (0, n_padded - len(channel_data
        )), 'constant')
37
38      #FFT
39      fft_data_window = fft(windowed_data_padded)
40      fft_data_channel = fft(channel_data_padded)
41
42      #Frekvensakse
43      frequencies = fftfreq(n_padded, d=sample_period)
44
45      #Normalisert amplitude window-data
46      magnitude_window_data = np.abs(fft_data_window)
47      magnitude_db_window = 20 * np.log10(magnitude_window_data)
48      magnitude_db_normalisert = magnitude_db_window - np.max(
        magnitude_db_window)
49
50      #Normalisert amplitude channel-data
51      magnitude_channel_data = np.abs(fft_data_channel)
52      magnitude_db_channel_data = 20 * np.log10(magnitude_channel_data)
53      magnitude_db_normalisert_channel_data = magnitude_db_channel_data - np.max
        (magnitude_db_channel_data)
54
55
56      #Plot
57      #n_padded//2 for aa faa med kun den positive delen av frekvensspekteret
58
59      plt.plot(frequencies[:n_padded//2], magnitude_db_normalisert[:n_padded
        //2], color='deepskyblue', label='Med Hanning-vindu')
60      plt.plot(frequencies[:n_padded//2], magnitude_db_normalisert_channel_data
        [:n_padded//2], color='hotpink',label='Uten Hanning-vindu')
61
62      plt.title('Sammenligning av frekvensspektrum med og uten Hanning-vindu')
63      plt.xlabel('Frekvens [Hz]')
64      plt.ylabel('Normalisert amplitude [dB]')
65      plt.xlim(50,150)
66      plt.legend()
67      plt.show()
```

Kode 7: Python kode for plot av frekvensspektrum med og uten Hanning-vindu.