



# TTT4280 – Lab

**Tittel:** Labrapport 1

**Skrevet av:** Mads Mølbach og Lars Johan Solheim Henriksen

**Gruppe:** 32

**Dato:** 16. februar 2024

---

## Sammendrag

Denne rapporten analyserer kvaliteten på fem samplede analoge signaler med hensyn til støy. Samplingen blir utført ved å bruke en Raspberry Pi, et støy reduksjonsfilter og fem ADC er av typen MCP3201. Hovedmålet er å bekrefte ADC-enes presisjon og effektivitet, noe som oppnås gjennom ulike signalbehandlingsteknikker. Rapporten tar for seg hvordan forskjellige vindusfunksjoner påvirker frekvensspekteret, med en oppnådd SNR-verdi over 74dB for en sinus signal med frekvens lik 100 Hz. Resultatene understreker systemets evne til nøyaktig signalanalyse og effektiv støyfiltrering, som er essensielt for pålitelige målinger i sensor- og instrumenteringssystemer.

## Akronymer

**ADC** Analog-to-Digital Converter

**SNR** Signal-to-Noise Ratio

**FFT** Fast Fourier Transform

**DMA** Direct Memory Access

**DC** Direct Current

**GPIO** General-Purpose Input/Output

**SPI** Serial Peripheral Interface

**VDD** Digital Supply Voltage

**VSS** Ground (Voltage Source)

**CLK** Clock Signal

**Dout** Digital Output

**CS** Chip Select

**MISO** Master In Slave Out

**MOSI** Master Out Slave In

**Vref** Reference Voltage

---

# Innhold

<b>1 Innledning</b>	<b>1</b>
<b>2 Teori</b>	<b>2</b>
2.1 ADC . . . . .	2
2.1.1 Støykilder for ADC . . . . .	2
2.2 Støy reduksjons filter(PI-filter) . . . . .	3
2.3 FFT . . . . .	4
2.3.1 Vindusfunksjoner . . . . .	5
2.3.2 Zero-padding . . . . .	5
2.4 Direct Memory Acces . . . . .	6
<b>3 Metode</b>	<b>7</b>
3.1 Systemoppsett . . . . .	7
3.1.1 Raspberry PI . . . . .	9
3.2 ADC krets . . . . .	12
3.2.1 MCP3201 . . . . .	12
3.2.2 ADC-Krets . . . . .	13
3.3 Sampling av signal . . . . .	14
3.4 Signalbehandling . . . . .	14
3.5 Støy reduksjons filter . . . . .	14
<b>4 Resultater</b>	<b>16</b>
4.1 Sampling . . . . .	16
4.2 Frekvensspekter med ulike signalbehandlingsteknikker . . . . .	16
4.3 Oppkoblet filter . . . . .	19
<b>5 Diskusjon</b>	<b>21</b>
<b>6 Konklusjon</b>	<b>22</b>
<b>Referanser</b>	<b>23</b>
<b>A Vedlegg A</b>	<b>24</b>

---

## 1 Innledning

I dette notatet skal det ses på hvordan man kan sette opp et målesystem bestående av 5 ADC-er av type MCP3201 [1], og en Raspberry Pi. Notatet tar for seg oppkobling og generering av nødvendige styresignaler for ADC-ene, samt lagring av data. Data fra ADC-ene skal deretter behandles for å verifisere systemets funksjonabilitet. Det samplede signalet vil bli undersøkt i tidsdomenet, og frekvensdomenet. Dette gir oss informasjon om kvaliteten av det samplede signalet og potensielle forvengninger i signalet.

Bruken av ADC-er har gjennom det siste århundre blitt viktigere og viktigere. ADC-er lar oss konvertere analoge signaler til digitale signaler, slik at disse kan behandles digitalt. ADC-er brukes blant annet innenfor mange fagfelt, som blant annet audio, video/bilde og medisin. Behovet for ADC-er og velfungerende sensor- og instrumenteringssystemer er derfor essensielt for mange bruksområder, og fundamentalt for at mange teknologiske systemer skal fungere.

## 2 Teori

### 2.1 ADC

Ved bruk av ADC-er, er det flere ting å ta i betraktning, ettersom egenskapene til ADC-er varierer. Noen viktige aspekter å sette seg inn i ved bruk av en ADC, er blant annet antall klokkesykluser per sample ( $N_{CLK}$ ), samplingsfrekvens ( $F_s$ ), bitoppløsning ( $2^N$ ) og forventet SNR. I bitoppløsningen står  $N$  for antall bits tilgjengelig for datarepresentasjon i ADC-en. Ved bruk av ADC-er er det også viktig å ta hensyn til ADC-ens kapasiteter og egenskaper, som for eksempel maksimum og minimum spennin på koblingsbenene ved forskjellige forsyningsspenninger.

Samplingsfrekvensen,  $F_s$ , til ADC-en er gitt av forholdet mellom klokkefrekvensen,  $F_{CLK}$ , og antall klokkesykluser per sample,  $N_{CLK}$

$$F_s = \frac{F_{CLK}}{N_{CLK}}. \quad (1)$$

Den forventede SNR-en ved bruk av en ADC, er viktig å ta stilling til. Den forventede SNR-en tilknyttet en ADC, forteller noe om forholdet mellom det ønskede signalet, og støyen som oppstår ved bruk av en ADC under ideelle forhold. Den forventede SNR-en er avhengig av bitoppløsningen på ADC-en det er snakk om, og følger av ligningen

$$SNR_{forventet} = 6.02N + 1.76. \quad (2)$$

#### 2.1.1 Støykilder for ADC

Ved en analog-digitalomvandling vil kontinuerlige analoge spenningskilder  $e(n)$  bli gjort om til diskret digitale spenningsverdier  $e_q(n)$ . Siden ADC-en kun har en endelig mengde med bits vil dette konversjonen føre til informasjonstap. AD-omvandlingen blir dermed beskrevet av

$$e(n) \rightarrow e_q(n) \Rightarrow e_q(n) = e(n) + \epsilon_q(n), \quad (3)$$

hvor  $\epsilon_q(n)$  er kvantiseringsfeilen som oppstår når vi utfører en AD-prosess. Dette er en ikke-inversibel prosess som vil si at vi har misitet informasjon og det derfor ikke lenger er mulig å reproduksere det analoge signalet perfekt[2].

Den vanligste formen for AD-konverterer bruker en linear Pulse-Code Modulation for å approksimere en kontinuerlig verdi. Dette vil si at et amplitude-intervall for inngangssignalet deles opp i like store steg  $\Delta$ . Inngangssignalet  $e(n)$  blir dermed representert av et heltall  $N$  som er tallet for steget som er nærmest amplituden. [2].

$\Delta$  er definert som

$$\Delta = \frac{V_{dd}}{2^n} \quad (4)$$

hvor n er antall bits tilgjengelig for ADC-en og  $V_{dd}$  forsyningsspenningen.

Det er åpenbart at denne kvantifiseringen gir en feil og at maksfeilen er på  $\pm \frac{\Delta}{2}$  kvantiseringsfeilen kan dermed ligge mellom intervallet

$$\epsilon_q \in [-\Delta/2, \Delta/2]. \quad (5)$$

Så lenge det analog input signalet  $e(n)$  er tilstrekkelig større enn  $\Delta$ , og  $e(n)$  inneholder mer enn en sinustone kan det antas at alle verdier i (5) er like sannsynlig. Med andre ord vil det si at kvantiseringsfeilen følger en uniform sannsynlighetsfordeling [2]. Siden kvantiseringsfeilen har en uniform sannsynlighetsfordeling er middelverdien  $\mu_q = 0$ . Variansen  $\sigma_q^2$  for den uniforme fordelingen kjent som

$$\sigma_q^2 = \frac{\Delta^2}{12}. \quad (6)$$

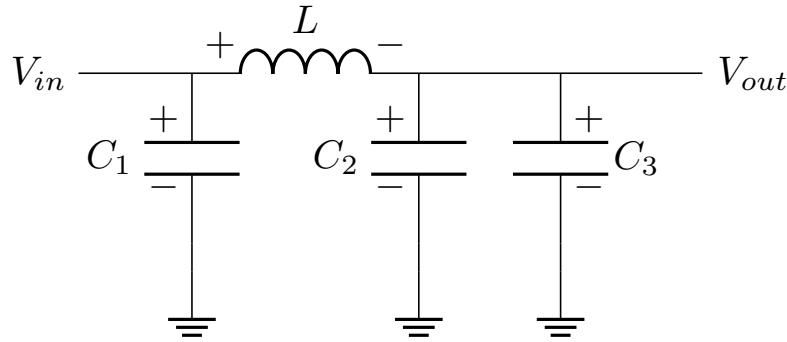
Om ønskelig er det mulig å linearisere en kvantiseringsprosess ved å bruke en såkalt *dither*. Det innebærer å legge til et stokastisk signal før kvantiseringen av signalet. Det finnes to ulike former for *dither*: nemlig *uniformdither*, som er en dither med uniform sannsynlighetsfordeling med et amplitudeområde innenfor  $[\pm \frac{\Delta}{2}]$ , eller ofte bedre, en *triangulardither*, som er stokastiske tall med en triangulært formet sannsynlighetsfordeling innenfor amplitudeområdet  $[\pm \Delta]$  [2].

Det er derimot verdt å merke seg at om en velger å ta i bruk *dithering*, kan det ofte føre til en høyere SNR-verdi. Det blir derfor en avveining hvor brukeren må velge mellom en kvantiseringsfeil som er korrelert med inngangssignalet eller en helt stokastisk kvantiseringsfeil med litt høyere SNR.

## 2.2 Støy reduksjons filter(PI-filter)

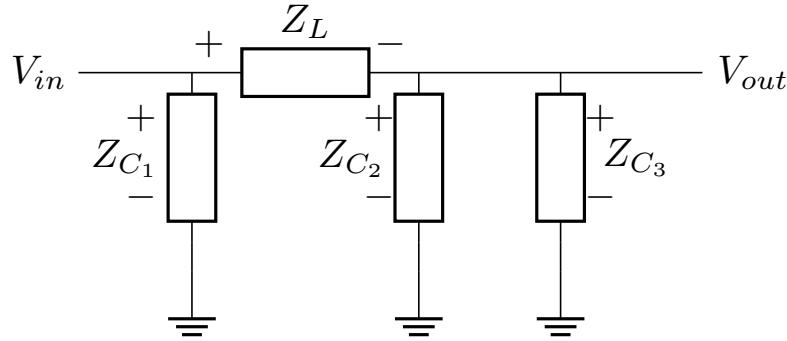
Støy på forsyningsspenningen er et velkjent problem for elektroingeniører. Spesielt når man skal arbeide med sensitive analoge komponenter som operasjonsforsterkere, ADC-er og mikrofoner, kan man spare seg selv for mye tid med feilsøking ved å benytte et støyreduksjonsfilter for forsyningslinjen på disse analoge komponentene.

For at ADC-ene skal fungere som forventet, er det derfor ønskelig å anvende et slikt filter. Et DC-filter, som vist i figur 1, brukes for å filtrere ut støyen og oppnå et mer stabilt forsyningsspenning på ADCen.



**Figur 1:** DC-filter for å redusere støy på forsyningsspenning til ADCene.

Ved å i bruk de respektive impedansene til spolene og kondensatorene kan DC-filteret tegnes som visst i figur 2.



**Figur 2:** DC-filter for å redusere støy med impedans verdier.

Amplituderesponsen til filteret kan beskrives som:

$$|H(\omega)| = \left| \frac{Z_{C_3}||Z_{C_2}}{Z_{C_3}||Z_{C_2} + Z_L} \right| = \left| \frac{\frac{-j}{\omega(C_2+C_3)}}{j\omega L - \frac{j}{\omega(C_2+C_3)}} \right| = \left| \frac{1}{1 - \omega^2 L(C_2 + C_3)} \right|. \quad (7)$$

Fra (7) kan en se at høyfrekvent støy blir filtrert bort fra forsyningsspenningen til Raspberry Pi, noe som gir et mer stabilt signal til ADC-ene som ønsket.

### 2.3 FFT

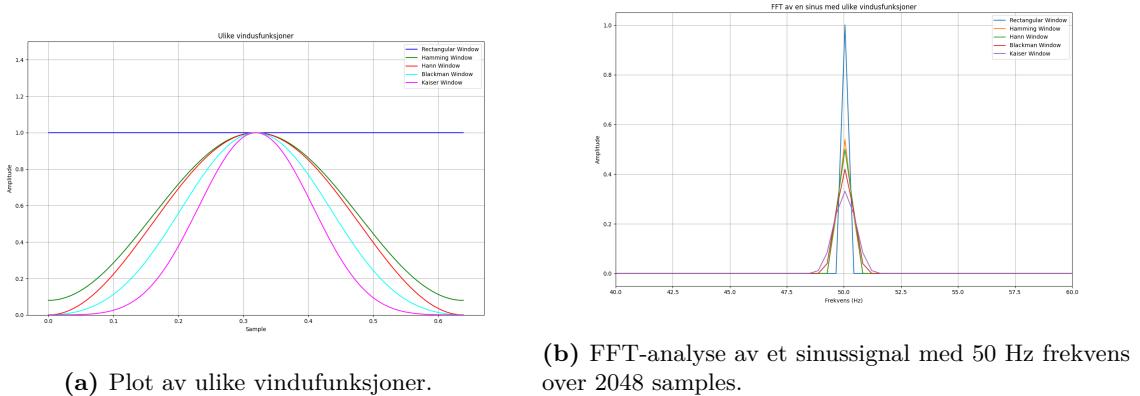
Innenfor flere fagfelt, er fourier transformasjon en essentiel operasjon. Fourier transformasjon brukes til å representere et signal som en sum av sinus-funksjoner. Innenfor signalbehandling er fourier-transformasjon spesielt viktig for å kunne analysere frekvensinnholdet i et signal. Dersom man, for eksempel, er interessert i støynivået til et signal, kan frekvensspekteret være nyttig for å analysere uønskede frekvenser i signalet.

Dersom man skal analysere fysiske signaler, må disse samples før behandling. Ved sampling blir man stående igjen med en samling av diskrete verdier. DFT (Discrete Fourier-Transform) er operasjonen for å ta fourier transformen av et diskret signal. FFT (Fast Fourier-Transform) er betegnelsen på algoritmer som reduserer beregningstiden til DFT-er. Ettersom signalbehandling hovedsakelig gjøres digitalt med datamaskiner, er behovet for gode FFT-algoritmer viktig for å begrense hvor mye beregningskapasitet som går til utregningene.

### 2.3.1 Vindusfunksjoner

Når det jobbes med tidsbegrensete signaler, som for eksempel ved sampling av et signal, kan vindusfunksjoner bidra til å begrense frekvenslekkasje. Vinudsfunksjoner kan også bidra til bedre oppløsning i frekvensdomene. Signalet multipliseres med vindusfunksjonen før Fouriertransformasjonen, hvor vindusfunksjonen som regel er en glatt funksjon som begrenser signalet i tid, og ofte reduserer signalets amplitude mot kantene av tidsbegrensningen. Den avtagende amplituden, sørger for mindre energilekkasje til høyere frekvenser[3].

Det finnes mange ulike vindudsfunksjoner. Figur 3a viser noen av de mest populære vindusfunksjonene og 3b hvordan en FFT av et sinussignal ser ut med og uten disse.



(a) Plot av ulike vindufunksjoner.

(b) FFT-analyse av et sinussignal med 50 Hz frekvens over 2048 samples.

**Figur 3:** Sammenligning av ulike vindufunksjoner og FFT-analyse av et sinussignal.

### 2.3.2 Zero-padding

Zero-padding er en teknikk brukt for å forbedre oppløsningen i frekvensssdomene. Ved zero-padding, legges det til nuller i signalet slik at antall punkter analysert i DFT-en øker uten at det legges til informasjon til signalet. Dette fører til en bedre oppløsning i frekvensssdomene. Den høyere oppløsningen tilfører ingen informasjon til signalet, men gjør det mulig å analysere frekvenser med høyere nøyaktighet.

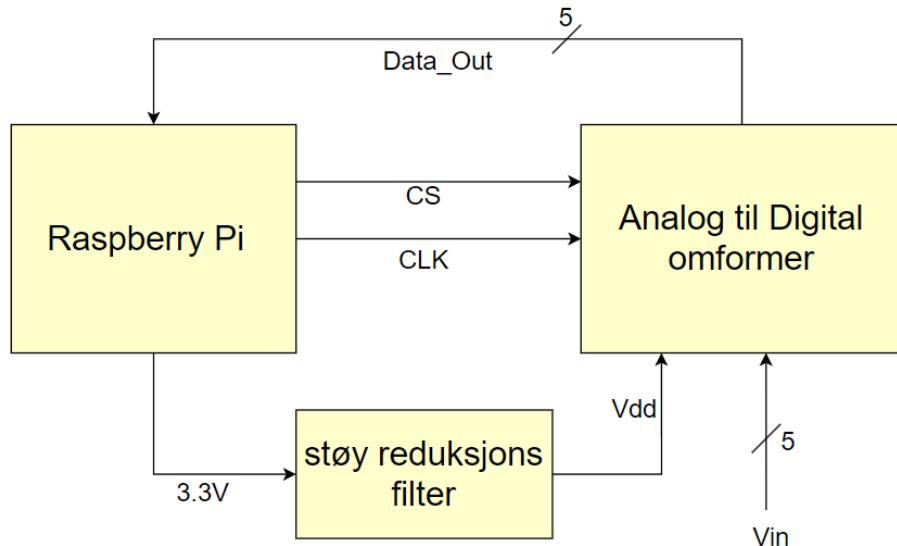
## 2.4 Direct Memory Acces

Raspberry Pi behandler de samplede signaler fra ADC-ene. Dette vil bli gjort via "“Direct Memory Access”(DMA). DMA gjør det mulig for periferiutstyr å overføre data til og fra systemminnet uten å involvere CPU-en. Dette er ønskelig da CPU-en blir mindre belastet, og kan dermed ta seg viktigere oppgaver. I tillegg tillater dette at CPU-en og DMA-en kan arbeide parallelt, noe som betyr at CPU-en kan fortsette med å utføre andre oppgaver mens DMA-en tar seg av henting av data fra periferiutstyret. Dette er ønskelig da det kan forekomme informasjonstap om CPUen må vente noen klokkesykluser ved høyfrekvent sampling.

### 3 Metode

#### 3.1 Systemoppsett

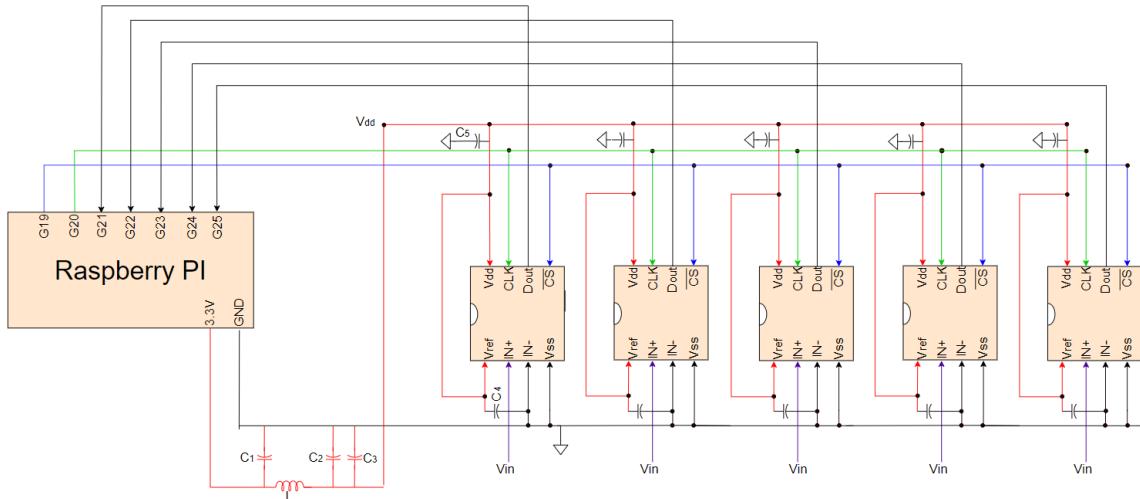
Figur 4 illustrerer et blokkskjema av kretsen og dataoverføringen mellom de ulike delsystemene.



**Figur 4:** Blokkskjema for systemet.

Som illustrert i blokkskjemaet blir det 3.3 V likespenningsignal fra Raspberry Pi, behandlet med et filter for støyreduksjon. Formålet med dette er å redusere støyen i forsyningsspenningen til de analog til digital omformerne, som er diskutert i grundigere detalje i kapittel 2. Resultatene fra eksperimenter med og uten bruken av dette filteret vil bli fremstilt i kapittel 4.3.

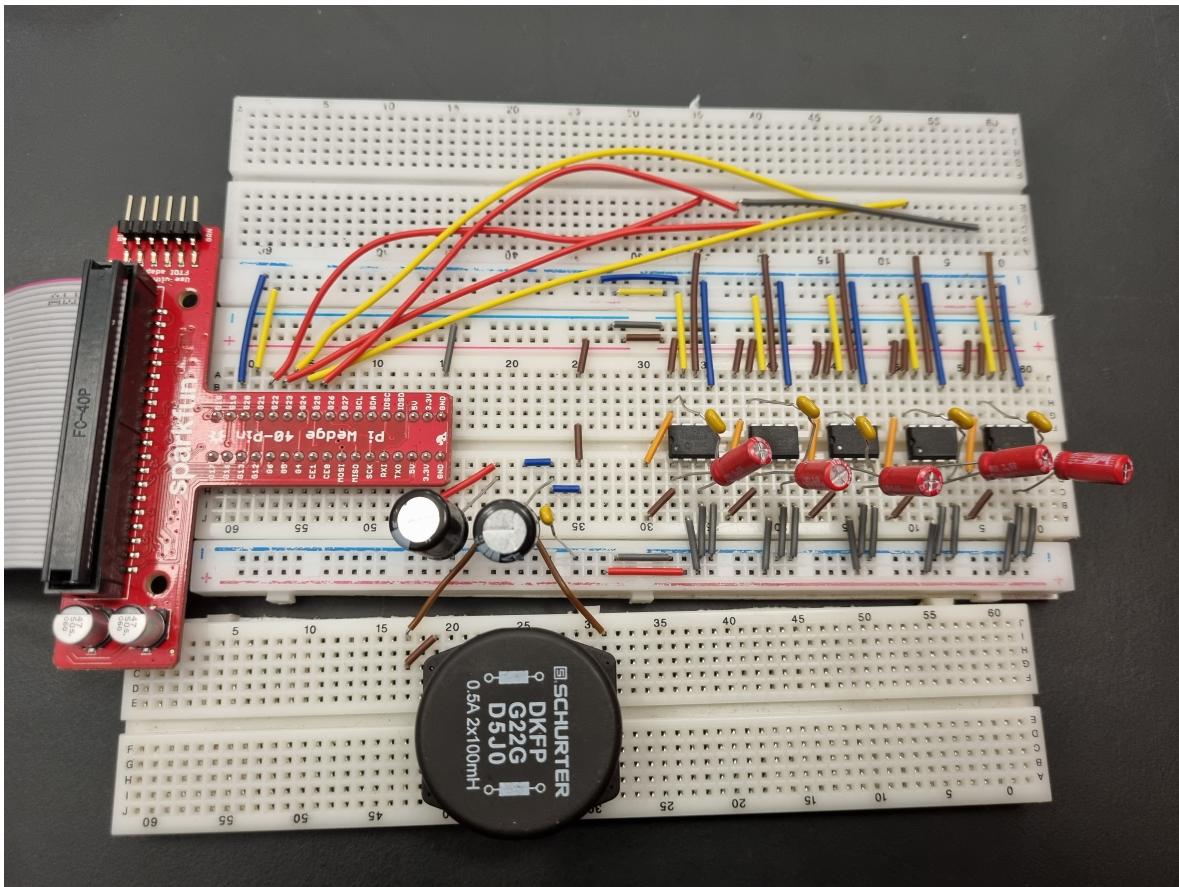
Figur 5 fremviser et kretskjema av kretsen med data overføring mellom komponentene og oppsett av koblingsben.



**Figur 5:** Kretsskjema for systemet.

Som figur 5 viser ble det valgt å bruke to ekstra kondensatorer  $C_4$  og  $C_5$ . Kondensator  $C_5$  er koblet mellom forsyningspenningen til ADC-ene  $V_{dd}$  og jord, og  $C_4$  er koblet mellom  $V_{ref}$  og jord.  $C_5$  og  $C_4$  fungerer som dekobliseringskondensatorer, og hindrer derfor støy.

Kretsen ble koblet opp som visst i 6.



Figur 6: Bilde av realisert krets med en PI wedge.

### 3.1.1 Raspberry PI

I målesystemet er det benyttet en raspberry pi for å generere styresignaler, samt spenningsforsyning til ADC-ene i systemet. Raspberry pi-en tar også imot signalene fra utgangen  $D_{out}$ , på ADC-ene. Raspberry pi-en står for lesing av data fra adc-ene, og lagre disse verdiene i en binær fil, som videre brukes til behandling av data.

Tabell 1 under viser hendholsvis hvilke koblingsben som ble brukt ved oppkobling av målesystemet demonstrert i dette notatet. Den følgende koden, er første del av koden i en C-fil som må opprettes på Raspberry pi-en. Denne koden genererer styresignalene og forsyningsspenningen for ADC-ene. Den står også for parallel innhenting av data fra de fem ADC-ene ved bruk av DMA. Koden lagt ved under, inneholder den delen av koden som må endres med hensyn til egne valg av koblingsbein og antall ADC-er, og samsvarer med oppkoblingen i demonstrert dette notatet. Koden i sin helhet er lagt ved i A. For å kunne kjøre denne koden, må et “C-bibliotek” kalt “pigpio” lastes inn på Raspberry Pi-en. Hvordan denne opplastningen gjøres og mer om oppsettet av Raspberry Pi-en blir beskrevet i [4].

**Tabell 1:** Tabell med koblingsben mellom ADC og Raspberry Pi

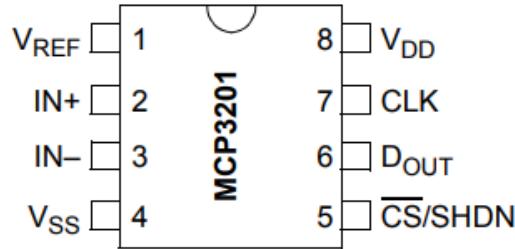
ADC	Raspberry Pi
$ADC_1 D_{out}$	GPIO 21
$ADC_2 D_{out}$	GPIO 22
$ADC_3 D_{out}$	GPIO 23
$ADC_4 D_{out}$	GPIO 24
$ADC_5 D_{out}$	GPIO 25
CLK	GPIO 20
$\overline{CS}$	GPIO 19
VDD	3.3V

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 #include <pigpio.h>
6 #include <math.h>
7 #include <time.h>
8 ////////////// USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO THEIR
      SETUP //////////
9 #define ADCS 5          // Number of connected MCP3201.
10
11 #define OUTPUT_DATA argv[2] // path and filename to dump
      buffered ADC data
12
13 /* RPi PIN ASSIGNMENTS */
14 #define MISO1 21          // ADC 1 MISO (GPIO pin number)
15 #define MISO2 22          // ADC 2 MISO (GPIO pin number)
16 #define MISO3 23          // ADC 3 MISO (GPIO pin number)
17 #define MISO4 24          // ADC 4 MISO (GPIO pin number)
18 #define MISO5 25          // ADC 5 MISO (GPIO pin number)
19
20 #define MOSI 10           // GPIO for SPI MOSI (GPIO 10 aka SPI_MOSI).
      MOSI not in use here due to single ch. ADCs, but must be
      defined anyway.
21 #define SPI_SS 19          // GPIO for slave select (GPIO 15).
22 #define CLK 20           // GPIO for SPI clock (GPIO 16).
23 /* END RPi PIN ASSIGNMENTS */
24
25 #define BITS 12            // Bits per sample.
26 #define BX 4                // Bit position of data bit B11. (3
      first are t_sample + null bit)
27 #define B0 (BX + BITS - 1) // Bit position of data bit B0.
28
29 #define NUM_SAMPLES_IN_BUFFER 300 // Generally make this buffer
      as large as possible in order to cope with reschedule.
30
31 #define REPEAT_MICROS 32 // Reading every x microseconds. Must
      be no less than 2xB0 defined above
32
33 #define DEFAULT_NUM_SAMPLES 31250 // Default number of samples
      for printing in the example. Should give 1sec of data at Tp
      =32us.
34
35 int MISO[ADCS]={MISO1, MISO2, MISO3, MISO4, MISO5}; // Must be
      updated if you change number of ADCs/MISOs above
36 ////////////// END USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO
      THEIR SETUP //////////
```

### 3.2 ADC krets

#### 3.2.1 MCP3201

I dette målesystemet er det brukt 5 ADC-er av type MCP3201. I databladet for MCP3201 [1], finner man relevant informasjon for bruk av komponenten. Figur 7, viser komponenten og dets respektive inn- og utganger  $V_{ref}$ ,  $IN+$ ,  $IN-$ ,  $V_{SS}$ ,  $V_{DD}$ ,  $CLK$ ,  $D_{OUT}$  og  $\overline{CS}$ . Her er  $D_{OUT}$  datautgangen til ADC-en,  $IN+$ ,  $IN-$ ,  $CLK$  og  $\overline{CS}$  er styresignaler, og  $V_{DD}$ ,  $V_{REF}$  og  $V_{SS}$  er spenningsforsyninger til ADC-en.



Figur 7: ADC MCP3201, hentet fra (KILDE)

I databladet til MCP3201 oppgis informasjon om komponenten, som sammen med teorien i seksjon 2, gir oss det vi trenger for å gjennomføre målinger. Minimum og maksimum spenninger på koblingsben for MCP3201 er

$$V_{min} = V_{SS} - 0.6V \quad (8)$$

$$V_{maks} = V_{DD} + 0.6V \quad (9)$$

MCP3201 er en 12-bits ADC. Fra dette kan vi ved ligning 2 finne hva forventet SNR for MCP3201 er under ideelle forhold. Ettersom dette er under ideelle forhold, vil ikke denne verdien nødvendigvis gjenspeiles i resultater ved målinger, men gi et inntrykk av hvilket verdiområde, den faktiske SNR-en burde ligge rundt. Den forventede SNR-en blir dermed

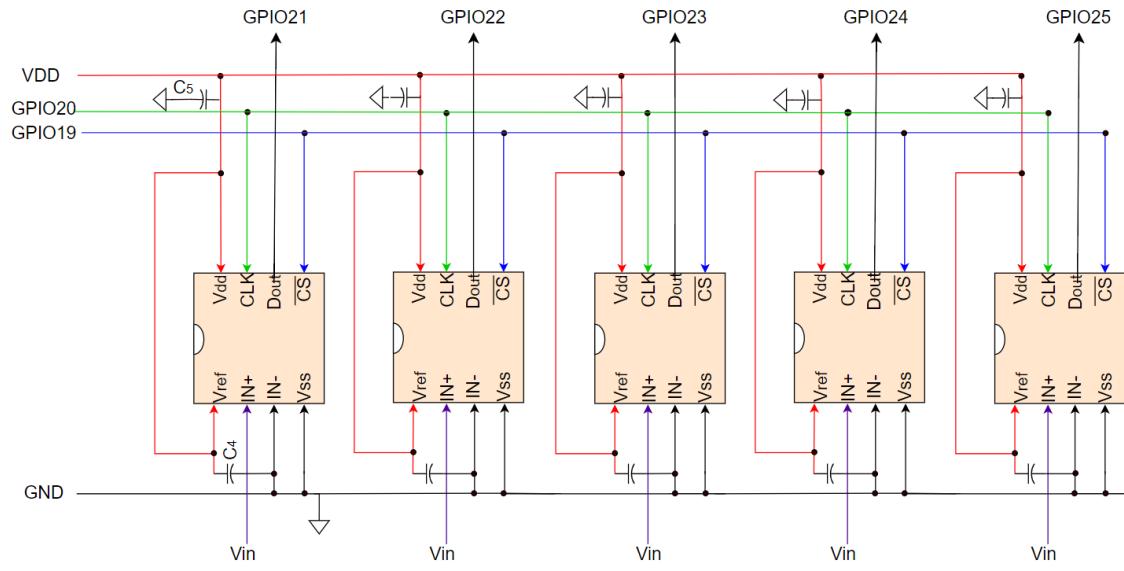
$$SNR_{forventet} = 6.02 * 12 + 1.76 = 74dB. \quad (10)$$

I dette målesystemet blir det brukt en klokkefrekvens på  $F_{CLK} = 500kHz$ , og en sample tar 16 klokkesykluser. Ved bruk av ligning 1, sitter man igjen med en samplefrekvens på

$$F_S = \frac{500kHz}{16} = 31250Hz. \quad (11)$$

### 3.2.2 ADC-Krets

Figur 8 viser et kretsskjema over hvordan ADC-ene er koblet isolert fra resten av systemet. ADC-ene deler styresignaler og spenningsforsyninger.  $IN+/V_{in}$  er inngangen for signalet vi ønsker å sample, mens  $D_{OUT}$  er det tilsvarende samplede signalet digitalisert.



**Figur 8:** Kretsskjema for ADC-krets

### 3.3 Sampling av signal

Målesystemet ble testet med et  $f_1(t) = 1.5 + \sin(2\pi \cdot 1000 \cdot t)$  signal og et  $f_2(t) = 1.5 + \sin(2\pi \cdot 500 \cdot t)$  signal. Disse signalene genereres og sendes inn på  $V_{in}$ . Den innlagte DC-offseten i signalet, kommet av at ADC-ene representerer, i denne oppkoblingen, verdier mellom 0V og 3.3V. Så for å unngå klipping samlingen, vi er avhengig av at signalet ligger mellom 0V og 3.3V

### 3.4 Signalbehandling

For å verifisere systemet, ble flere aspekter ved det samplede signalet analysert. De samplede dataene fra de fem ADC-ene, ble sammenlignet i tidsdomene. Dette ble gjort for å bekrefte at det samplede signalet, visuelt lignet inngangssignalet til systemet. Selv om det visuelle kan bekrefte signalet til en viss grad, er det hensiktsmessig å se på frekvensinnholdet i signalet. Det ble derfor tatt fourier transform av det samplede signalet, for å kunne analysere frekvensspekteret til det samplede signalet. For enkelhets skyld, er kun resultatene fra signalet på 1000Hz fra én av ADC-ene plottet i frekvensdomenet. I frekvensdomenet, er det plottet med og uten vindusfunksjoner. Det ble valgt å se på de to vindusfunksjonene "blackman window" og "rectangular window", som begge er velkjente vindusfunksjoner. Ettersom det ofte er interessant å se nærmere på innholdet til signalet rundt en bestemt frekvens, er det demonstrert hvordan et område rundt 1000 Hz ser ut med og uten "zero-padding". SNR til signalet ble også undersøkt, men kun visuelt av frekvensspekteret. Dette ble da målt som differansen i amplitude (dB), fra den ønskede frekvensen, ned til den høyeste verdien i støygulvet [5].

### 3.5 Støy reduksjons filter

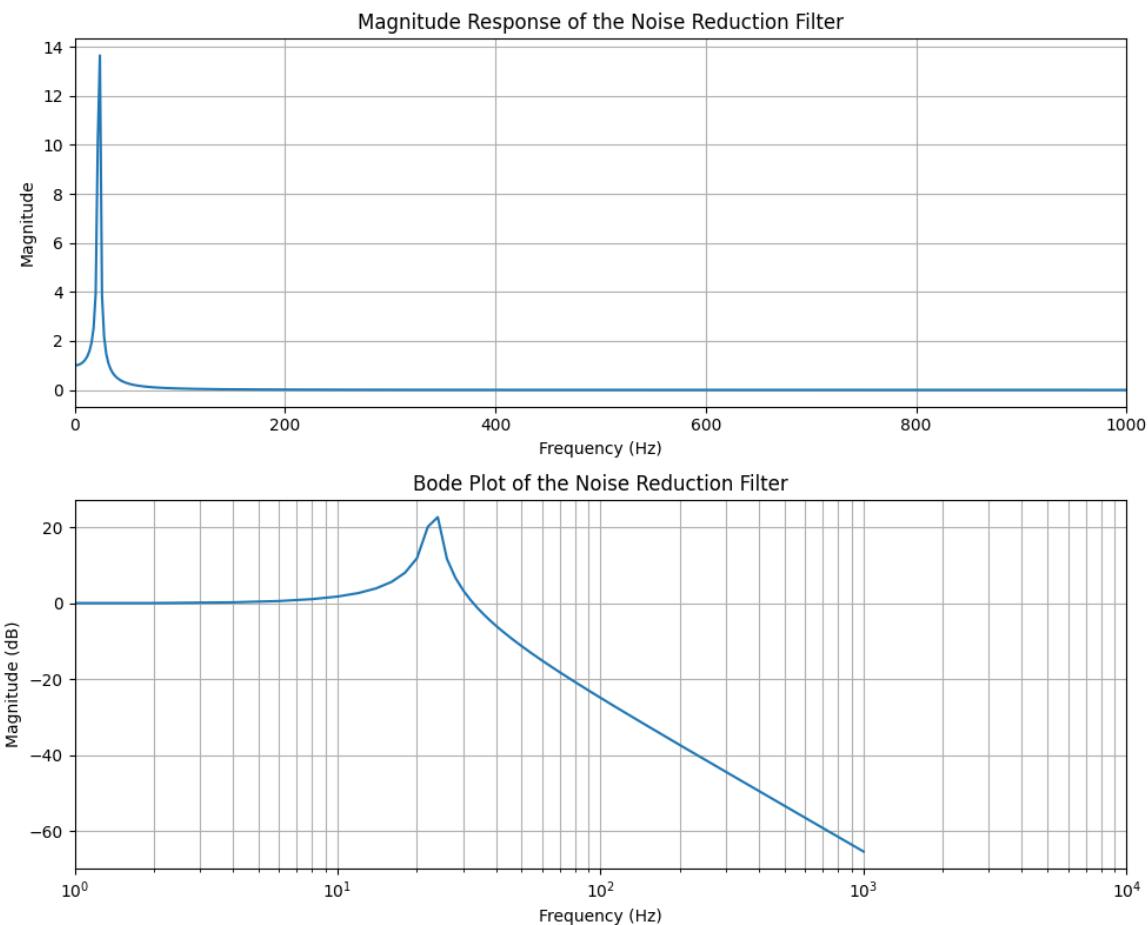
Det blir tatt i bruk et filter for å minimere støyen til forsyningsspenningen til ADC-ene, som forklart i kapittel 2.

Filteret blir koblet opp med verdier visst i tabell 2.

**Tabell 2:** Tabell av nominell verdier for komponenter bruk i DC-filter.

Komponent	Nominell verdi
$C_1$	470 $\mu\text{F}$
$C_2$	470 $\mu\text{F}$
$C_3$	100 nF
$L$	100 mH

. Figur 9 viser den teoretiske bodeplottet og magnituderesponsen til filteret.



**Figur 9:** Bodeplot og amplituderespons for filteret.

I kapittel 4.3 vil bodeplottet til det realiserte filteret bli presentert.

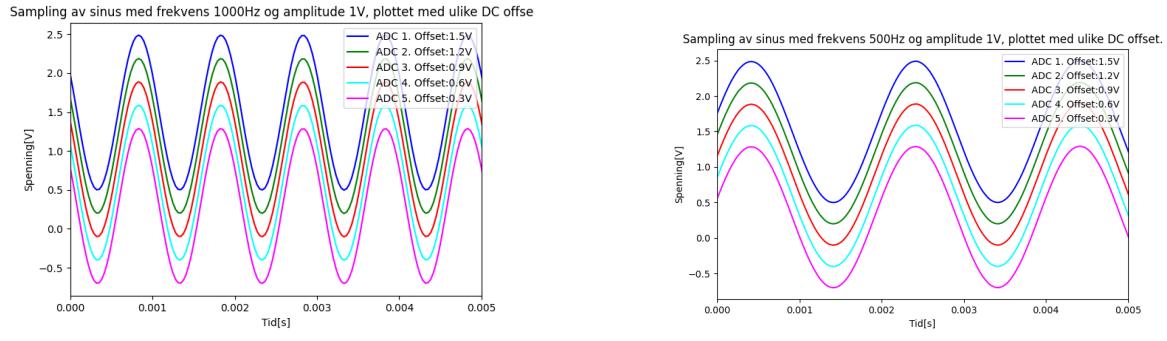
## 4 Resultater

Denne seksjonen presenterer de målte resultatene basert på de innsamlede prøvene. Seksjonen vil omfatte resultatene knyttet til det samplede inngangssignalet, dets frekvenspekter, og systemets SNR. Videre illustreres hvordan forskjellige signalbehandlingsmetoder påvirker systemets frekvenspekter.

### 4.1 Sampling

Tre plots med ulike frekvenser. SNR. Sjekk om en får riktig spenningsutsving uten klipping og om det eventuelt dukker opp uønskede signalkomponenter i spekteret (støy/interferens).

For å bekrefte funksjonaliteten til ADC-ene, ble det utført en test der en sinusbølge med en frekvens på 1000Hz og 500Hz( $V_{in}$ ) ble sendt inn som inngangssignal til alle ADC-ene. Under datainnsamlingen ble det lagt til en offset for å gjøre det enklere å atskille signalene fra de forskjellige ADC-ene. Analyse av spektrumet for de ulike ADC-ene med inngangssignal på en sinusbølge med frekvenser på henholdsvis 1000 Hz og 500 Hz er presentert i figur 10a og figur 10b.



(a) Spekteret i tidsplan til de fem ADC-ene med  $V_{in}$  som en sinus med frekvens på 1000 Hz.

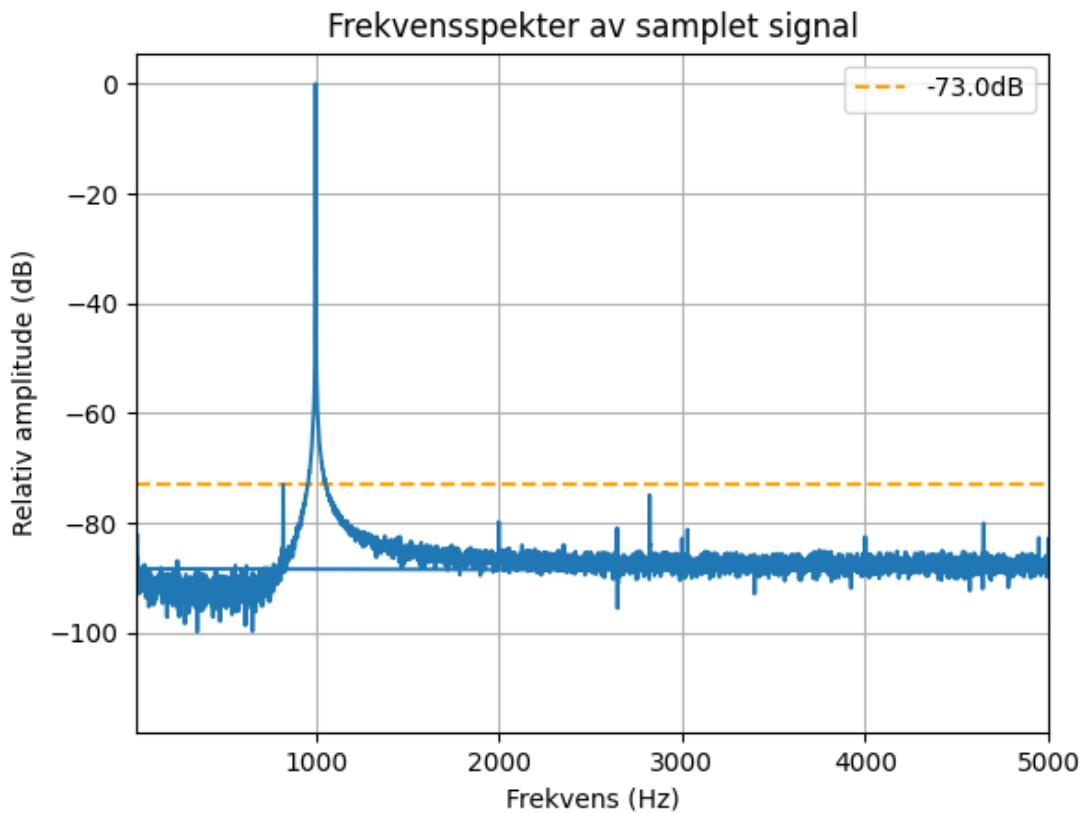
(b) Spekteret i tidsplan til de fem ADC-ene med  $V_{in}$  som en sinus med frekvens på 500 Hz.

**Figur 10:** Spekteret i tidsplan til de fem ADC-ene med  $V_{in}$  som en sinus med to ulike frekvenser. Det er lagt til en offset for å gjøre det enklere å adskille ADC-ene.

Som figur 10 viser fungerer samplingen av signalet som forventet.

### 4.2 Frekvenspekter med ulike signalbehandlingsteknikker

For å bekrefte at signalene fra ADC-ene opererer som antatt og at signalstøyen er innenfor akseptable grenser, plottes et frekvenspekter til en sinusbølge med frekvens på 1000 Hz ved bruk av FFT. Resultatet er visst i figur 11.

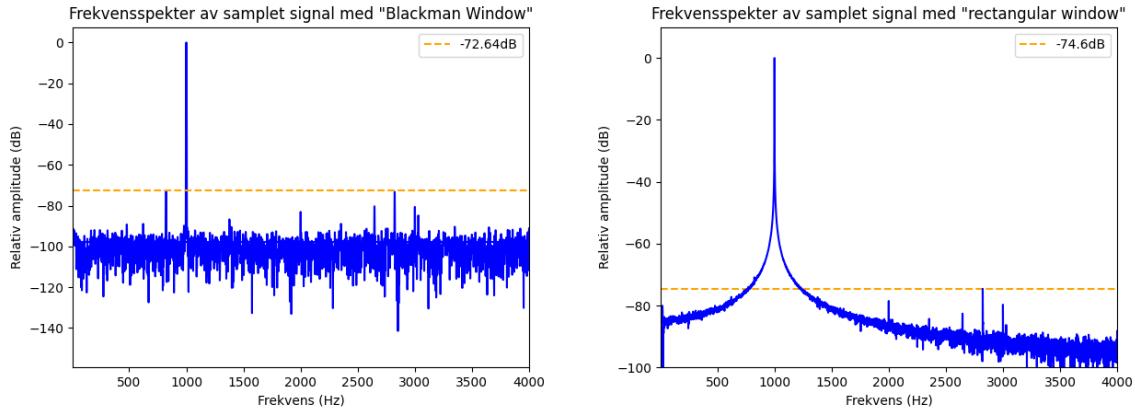


**Figur 11:** Frekvensspekteret til en ADC når den mottar en sinusformet inngangssignal med en frekvens på 1000 Hz. Den oransje stripla linjen indikerer den høyeste støytopenn som tilsvarer SNR til systemet.

For å redusere støynivået og sideblobbene til signalet blir det valgt å teste signalet med ulike vindsfunksjoner. Figur 12a og 12b framstiller frekvensspekteret til signalet plottet med vindsfunksjoner "blackman" og "rectangular" i respektiv rekkefølge.

## Oppkoblet filter

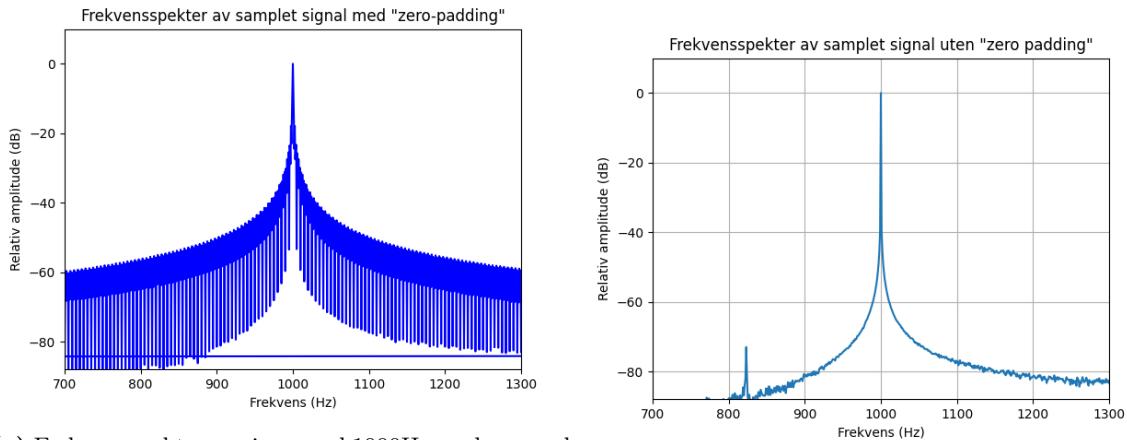
---



(a) Frekvensspekter av sinus med 1000Hz med vindusfunksjon Blackman.  
(b) Frekvensspekter av sinus med 1000Hz med vindusfunksjon Rectangular.

**Figur 12:** Frekvensspekter av sinus med 1000Hz med forskjellige vindusfunksjoner. Den oransje strikkete linjen viser SNR-nivået til systemet.

Frekvensspekteret til signalet med og uten zero-padding er vist i figur 13a og 13b.



(a) Frekvensspekter av sinus med 1000Hz med zeropadding. Det er lagt til 18 750 nuller som tilsvarer 1.6 gang orginale lengden til signalet.  
(b) Frekvensspekter av sinus med 1000Hz uten zeropadding.

**Figur 13:** Frekvensspekter av sinus med 1000Hz med og uten bruk av zeropadding

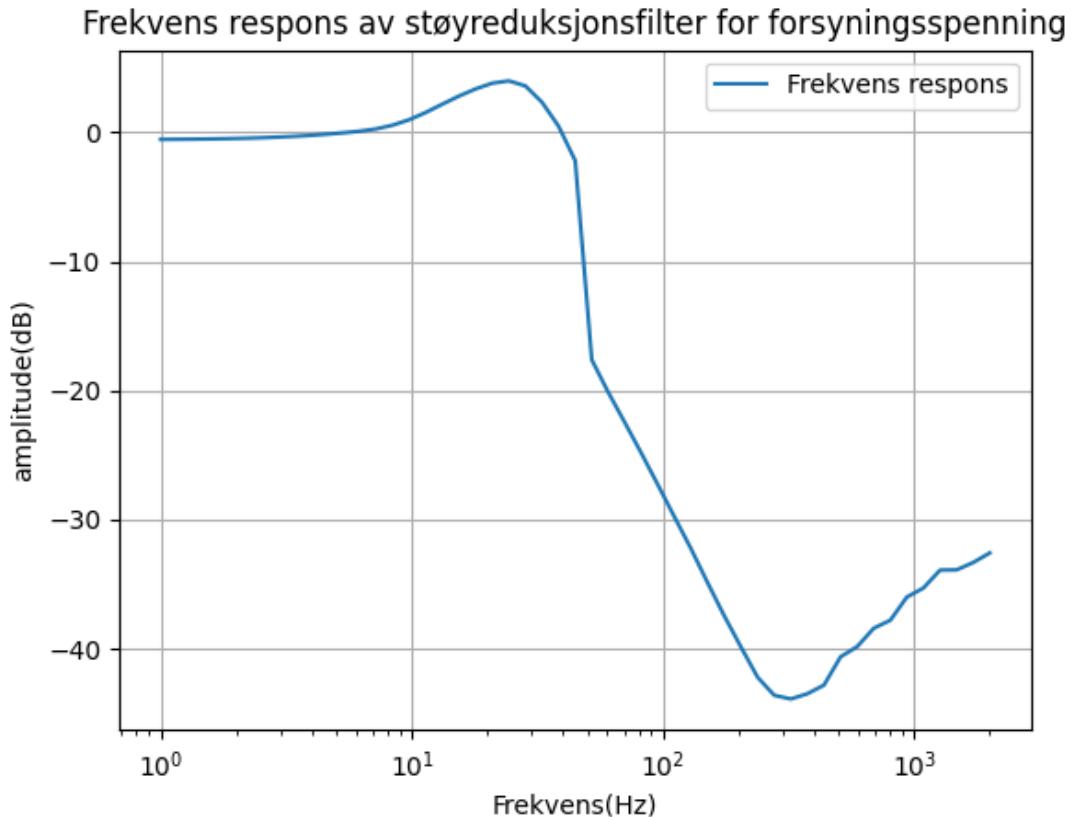
SNR til signalet for de ulike tilfellene er vist i tabell 3

**Tabell 3:** Tabell over SNR-verdier for ulike vindusfunksjoner.

Vindusfunksjon	SNR-verdi
Blackman	72.64 dB
Rectangular	74.6 dB
Uten vindu	73.0 dB

#### 4.3 Oppkoblet filter

For å verifisere at filteret fungerer som forventet velges det å plotte bode-plottet til filteret. Det realiserte bodeplottet til filteret er visst i figur 14.



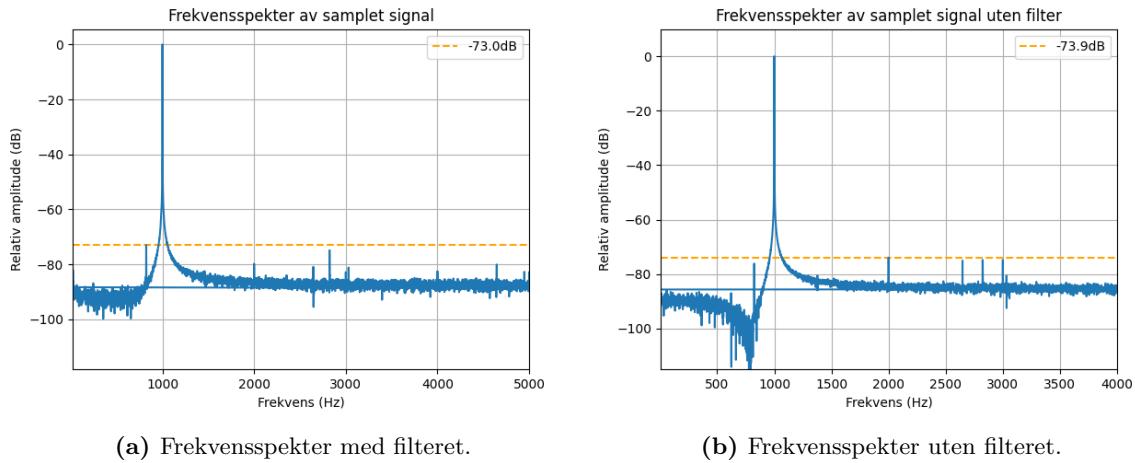
**Figur 14:** Bodeplot til realisert filter.

Som vist i figur 14, fungerer filteret stort sett som forventet. Derimot opplever filteret en viss økning i desibel ved høyere frekvenser, noe som gjør det mindre optimalt i disse områdene. Dette er derimot ikke et stort problem for filterets formål med å redusere støy.

Frekvensspekteret til en sinus med og uten filter er visst i figur 15a og 15b.

## Oppkoblet filter

---



**Figur 15:** Frekvensspekter av sinus på 1000Hz med og uten bruk av støy reduksjons filteret.

Det er tydelig at bruk av filteret ikke medfører betydelige endringer i frekvensspekteret ved høyere frekvenser. Derimot så ser vi at lavere frekvenser har blitt filtrert til en stor grad som ønsket.

Tabell 4 viser målt SNR-verdi med og uten filter

**Tabell 4:** Målt SNR-verdi med og uten filter.

	SNR-verdi
Med Filter	73.0dB
Uten Filter	73.9dB

## 5 Diskusjon

Systemet i sin helhet fungerer som forventet. Alle fem ADC-ene sampler signalet som forventet, slik at det kan rekonstrueres gjennom behandling. Målesystemet klarer å ta opp signalene fra de fem ADC-ene parallelt og lagre disse som binær-filer på Raspberry Pi-en. Visuelt kan man se i tidsdomene, at alle ADC-ene representerer signalet uten store avvik.

Frekvensspekteret til det samplede signalet har en tydelig amplitud i den ønskede frekvensen. Samtidig ligger amplituden til støygulvet betraktelig lavere enn signalet. Dette tyder på at signalet blir representert uten store forvrengninger. Dette er et viktig aspekt når det kommer til målesystemer, for å forsikre seg om at målingene er til å stole på.

Fra målingene våre av SNR, selv om disse kun er gjort visuelt og ikke matematisk, viser at støynivået er relativt lavt. Vi kan også sammenligne målingene SNR med den forventede SNR-en fra seksjon, og ser at den målte verdien ligger rundt den forventede verdien. En utregning av den faktiske SNR kan gjøres dersom man ønsker et mer nøyaktig mål på SNR. Nødvendig SNR varierer med tanke på bruksområde og hvor nøyaktige målinger man har behov for.

## 6 Konklusjon

Måleystemet sampler inngangssignalet, og representerer det til den grad at det ikke viker fra inngangssignalet ved visuell verifikasjon. Ved undersøkelse av frekvensspekteret, vises det støy i signalet, men støyen er tilstrekkelig lav til at systemet kan kvalifiseres som funksjonabelt. Målinger for SNR er gjort visuelt, og vil derav avvike fra den faktiske SNR-en.

## Referanser

- [1] Microchip: *Datasheet for component MCP3201.* <https://ww1.microchip.com/downloads/en/DeviceDoc/21290F.pdf>, 2011. Accessed: February 16, 2024.
- [2] U. Peter Svensson, Egil Eide, Lise Lyngnes Randeberg: *Instrumentering*, kapittel 3, sider 79–84. Institutt for elektroniske systemer — NTNU, 2023.
- [3] U. Peter Svensson, Egil Eide, Lise Lyngnes Randeberg: *Instrumentering*, kapittel 2, sider 37–40. Institutt for elektroniske systemer — NTNU, 2023.
- [4] NTNU, Faculty of Information Technology and Electrical Engineering: *User Guide for the Laboratory*, 2024.
- [5] Cisco Meraki: *Signal-to-Noise Ratio (SNR) and Wireless Signal Strength.* [https://documentation.meraki.com/MR/Wi-Fi\\_Basics\\_and\\_Best\\_Practices/Signal-to-Noise\\_Ratio\\_\(SNR\)\\_and\\_Wireless\\_Signal\\_Strength](https://documentation.meraki.com/MR/Wi-Fi_Basics_and_Best_Practices/Signal-to-Noise_Ratio_(SNR)_and_Wireless_Signal_Strength), 2023. Accessed: 16-02-2024.

## A Vedlegg A

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 #include <pigpio.h>
6 #include <math.h>
7 #include <time.h>
8 ////////////// USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO THEIR
9 // SETUP //////////
10 #define ADCS 5           // Number of connected MCP3201.
11
12 #define OUTPUT_DATA argv[2] // path and filename to dump
13             buffered ADC data
14
15 /* RPi PIN ASSIGNMENTS */
16 #define MISO1 21          // ADC 1 MISO (GPIO pin number)
17 #define MISO2 22          // ADC 2 MISO (GPIO pin number)
18 #define MISO3 23          // ADC 3 MISO (GPIO pin number)
19 #define MISO4 24          // ADC 4 MISO (GPIO pin number)
20 #define MISO5 25          // ADC 5 MISO (GPIO pin number)
21
22 #define MOSI 10          // GPIO for SPI MOSI (GPIO 10 aka SPI_MOSI).
23             MOSI not in use here due to single ch. ADCs, but must be
24             defined anyway.
25 #define SPI_SS 19         // GPIO for slave select (GPIO 15).
26 #define CLK 20            // GPIO for SPI clock (GPIO 16).
27 /* END RPi PIN ASSIGNMENTS */
28
29 #define BITS 12           // Bits per sample.
30 #define BX 4               // Bit position of data bit B11. (3
31             first are t_sample + null bit)
32 #define B0 (BX + BITS - 1) // Bit position of data bit B0.
33
34 #define NUM_SAMPLES_IN_BUFFER 300 // Generally make this buffer
35             as large as possible in order to cope with reschedule.
36
37 #define REPEAT_MICROS 32 // Reading every x microseconds. Must
38             be no less than 2xB0 defined above
39
40 #define DEFAULT_NUM_SAMPLES 31250 // Default number of samples
41             for printing in the example. Should give 1sec of data at Tp
42             =32us.
```

34

```
35 int MISO[ADCS]={MISO1, MISO2, MISO3, MISO4, MISO5}; // Must be
   updated if you change number of ADCs/MISOs above
36 ////////////// END USER SHOULD MAKE SURE THESE DEFINES CORRESPOND TO
   THEIR SETUP //////////
37
38 /**
39 * This function extracts the MISO bits for each ADC and
40 * collates them into a reading per ADC.
41 *
42 * \param adcs Number of attached ADCs
43 * \param MISO The GPIO connected to the ADCs data out
44 * \param bytes Bytes between readings
45 * \param bits Bits per reading
46 * \param buf Output buffer
47 */
48 void getReading(int adcs, int *MISO, int OOL, int bytes, int
   bits, char *buf)
49 {
50     int p = OOL;
51     int i, a;
52
53     for (i=0; i < bits; i++) {
54         uint32_t level = rawWaveGetOut(p);
55         for (a=0; a < adcs; a++) {
56             putBitInBytes(i, buf+(bytes*a), level & (1<<MISO[a]));
57         }
58         p--;
59     }
60 }
61
62
63 int main(int argc, char *argv[])
64 {
65     // Parse command line arguments
66     long num_samples = 0;
67     if (argc <= 1) {
68         fprintf(stderr, "Usage: %s NUM_SAMPLES\n\n"
69                 "Example: %s %d\n", argv[0], argv[0],
70                 DEFAULT_NUM_SAMPLES);
71         exit(1);
72     }
73     sscanf(argv[1], "%ld", &num_samples);
74
75     // Array over sampled values, into which data will be saved
76     uint16_t *val = (uint16_t*)malloc(sizeof(uint16_t)*
       num_samples*ADCS);
```

```
77     // SPI transfer settings, time resolution 1us (1MHz system
78     // clock is used)
79     rawSPI_t rawSPI =
80     {
81         .clk      = CLK,    // Defined before
82         .mosi     = MOSI,   // Defined before
83         .ss_pol   = 1,     // Slave select resting level.
84         .ss_us    = 1,     // Wait 1 micro after asserting slave
85         select.
86         .clk_pol  = 0,     // Clock resting level.
87         .clk_pha  = 0,     // 0 sample on first edge, 1 sample on
88         second edge.
89         .clk_us   = 1,     // 2 clocks needed per bit so 500 kbps.
90     };
91
92     // Change timer to use PWM clock instead of PCM clock.
93     // Default is PCM
94     // clock, but playing sound on the system (e.g. espeak at
95     // boot) will start
96     // sound systems that will take over the PCM timer and make
97     adc_sampler.c
98     // sample at far lower samplerates than what we desire.
99     // Changing to PWM should fix this problem.
100    gpioCfgClock(5, 0, 0);
101
102    // Initialize the pigpio library
103    if (gpioInitialise() < 0) {
104        return 1;
105    }
106
107    // Set the selected CLK, MOSI and SPI_SS pins as output pins
108    gpioSetMode(rawSPI.clk, PI_OUTPUT);
109    gpioSetMode(rawSPI.mosi, PI_OUTPUT);
110    gpioSetMode(SPI_SS, PI_OUTPUT);
111
112    // Flush any old unused wave data.
113    gpioWaveAddNew();
114
115    // Construct bit-banged SPI reads. Each ADC reading is
116    // stored separately
117    // along a buffer of DMA commands (control blocks). When the
118    // DMA engine
119    // reaches the end of the buffer, it restarts on the start
120    // of the buffer
121    int offset = 0;
122    int i;
123    char buf[2];
```

```
115     for (i=0; i < NUM_SAMPLES_IN_BUFFER; i++) {
116         buf[0] = 0xC0; // Start bit, single ended, channel 0.
117
118         rawWaveAddSPI(&rawSPI, offset, SPI_SS, buf, 2, BX, B0,
119                     B0);
120         offset += REPEAT_MICROS;
121     }
122
123     // Force the same delay after the last command in the buffer
124     gpioPulse_t final[2];
125     final[0].gpioOn = 0;
126     final[0].gpioOff = 0;
127     final[0].usDelay = offset;
128
129     final[1].gpioOn = 0; // Need a dummy to force the final
130     delay.
131     final[1].gpioOff = 0;
132     final[1].usDelay = 0;
133
134     gpioWaveAddGeneric(2, final);
135
136     // Construct the wave from added data.
137     int wid = gpioWaveCreate();
138     if (wid < 0) {
139         fprintf(stderr, "Can't create wave, buffer size %d too
large?\n", NUM_SAMPLES_IN_BUFFER);
140         return 1;
141     }
142
143     // Obtain addresses for the top and bottom control blocks (CB)
144     // in the DMA
145     // output buffer. As the wave is being transmitted, the
146     // current CB will be
147     // between botCB and topCB inclusive.
148     rawWaveInfo_t rwi = rawWaveInfo(wid);
149     int botCB = rwi.botCB;
150     int topOOL = rwi.topOOL;
151     float cbs_per_reading = (float)rwi.numCB / (float)
152     NUM_SAMPLES_IN_BUFFER;
153
154     float expected_sample_freq_khz = 1000.0/(1.0*REPEAT_MICROS);
155
156     printf("# Starting sampling: %ld samples (expected Tp = %d
157 us, expected Fs = %.3f kHz).\n",
158           num_samples, REPEAT_MICROS, expected_sample_freq_khz);
159
160     // Start DMA engine and start sending ADC reading commands
```

```
155     gpioWaveTxSend(wid, PI_WAVE_MODE_REPEAT);  
156  
157     // Read back the samples  
158     double start_time = time_time();  
159     int reading = 0;  
160     int sample = 0;  
161  
162     while (sample < num_samples) {  
163         // Get position along DMA control block buffer  
corresponding to the current output command.  
164         int cb = rawWaveCB() - botCB;  
165         int now_reading = (float) cb / cbs_per_reading;  
166  
167         while ((now_reading != reading) && (sample < num_samples)) {  
168             // Read samples from DMA input buffer up until the  
current output command  
169  
170             // OOL are allocated from the top down. There are  
BITS bits for each ADC  
171             // reading and NUM_SAMPLES_IN_BUFFER ADC readings.  
The readings will be  
172             // stored in topOOL - 1 to topOOL - (BITS *  
NUM_SAMPLES_IN_BUFFER).  
173             // Position of each reading's OOL are calculated  
relative to the wave's top  
174             // OOL.  
175             int reading_address = topOOL - ((reading %  
NUM_SAMPLES_IN_BUFFER)*BITS) - 1;  
176  
177             char rx[8];  
178             getReading(ADCS, MISO, reading_address, 2, BITS, rx)  
;  
179  
180             // Convert and save to output array  
181             for (i=0; i < ADCS; i++) {  
182                 val[sample*ADCS+i] = (rx[i*2]<<4) + (rx[(i*2)  
+1]>>4);  
183             }  
184  
185             ++sample;  
186  
187             if (++reading >= NUM_SAMPLES_IN_BUFFER) {  
188                 reading = 0;  
189             }  
190         }  
191         usleep(1000);
```

```
192     }
193
194     double end_time = time_time();
195
196     double nominal_period_us = 1.0*(end_time-start_time)/(1.0*
197     num_samples)*1.0e06;
198     double nominal_sample_freq_khz = 1000.0/nominal_period_us;
199
200     printf("# %ld samples in %.6f seconds (actual T_p = %f us,
201 nominal Fs = %.2f kHz).\n",
202         num_samples, end_time-start_time, nominal_period_us,
203         nominal_sample_freq_khz);
204
205     double output_nominal_period_us = floor(nominal_period_us);
206     //the clock is accurate only to us resolution
207
208     // Path to your data directory/file from previous define
209     const char *output_filename;
210     char hold_fname[32];
211     if (argc < 3) { // No filename supplied, get default
212         time_t t = time(NULL);
213         struct tm tm = *localtime(&t);
214         snprintf(hold_fname, 32, "./out-%4d-%02d-%02d-%02d.%02d
215         .%02d.bin",
216             tm.tm_year+1900, tm.tm_mon+1, tm.tm_mday,
217             tm.tm_hour, tm.tm_min, tm.tm_sec);
218         output_filename = hold_fname;
219     } else {
220         output_filename = OUTPUT_DATA;
221     }
222
223     // Write sample period and data to file
224     FILE *adc_data_file = fopen(output_filename, "wb+");
225     if (adc_data_file == NULL) {
226         fprintf(stderr, "# Couldn't open file for writing: %s (
227         did you remember to change OUTPUT_DATA?)\n", output_filename)
228     ;
229     return 1;
230 }
231
232     fwrite(&output_nominal_period_us, sizeof(double), 1,
233     adc_data_file);
234     fwrite(val, sizeof(uint16_t), ADCS*num_samples,
235     adc_data_file);
236     fclose(adc_data_file);
237     printf("# Data written to file. Program ended successfully.\n\n");
```

```
229     gpiotTerminate();
230     free(val);
231
232     return 0;
233 }
234 }
```