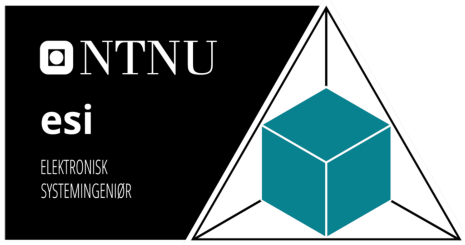
Designnotat 12



Tittel: Artikulasjonstrener - Spektalanalyse på OLED-skjerm

Forfatter(e): Temmam Nabil Altaie

Versjon: 1.0 | **Dato: 30. april 2026**

Innhold

1 Problembeskrivelse **2**

2 Prinsipiell løsning **3**

2.1 Overordnet arkitektur . . . 3
2.2 Spektralanalyse . . . 3
2.3 Amplitude- og effektspekter . . . 4
2.4 Visualisering på OLED . . . 4

3 Realisering og test **5**

3.1 FFT-implementasjon . . . 6
3.2 Tidsmåling . . . 7
3.3 OLED-visualisering . . . 7
3.4 Data Visualizer . . . 8
3.5 Forsøk med lydfil . . . 8

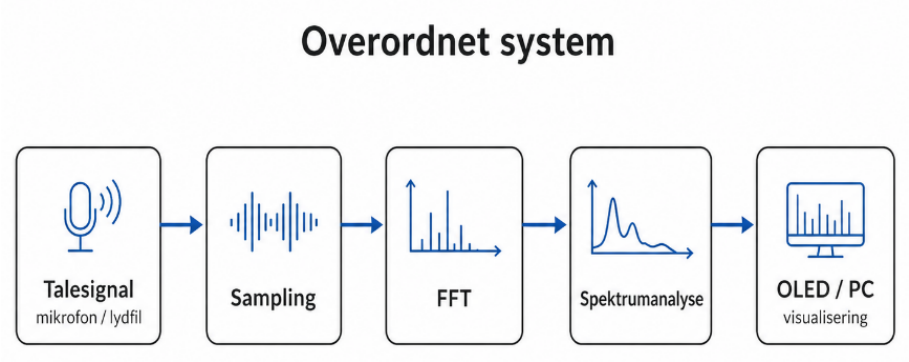
4 Konklusjon **9**

5 Takk **9**

6 Vedlegg **10**

1 Problembeskrivelse

I dette designprosjektet skulle det utvikles en enkel artikulasjonstrener basert på spektralanalyse av tale. Målet var å lage et mikrokontrollerbasert system som kan estimere og vise frekvensinnholdet i et signal på en OLED-skjerm. Systemet er tenkt brukt til analyse av vokaler, der ulike vokaler kan skilles fra hverandre ved hjelp av karakteristiske topper i spekteret, såkalte formanter. Den overordnede systemideen er vist i figur 1.



Figur 1: Overordnet systemidé for artikulasjonstreneren, der et talesignal behandles og presenteres som et frekvensspekter.

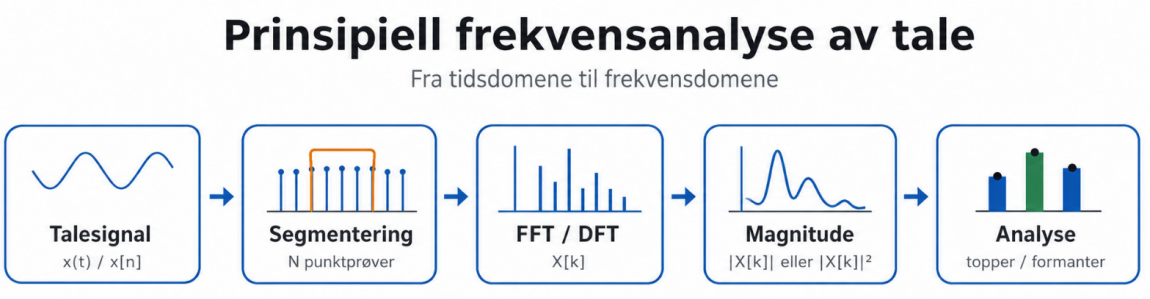
Figur 1 viser hovedprinsippet bak løsningen. Et inngangssignal fra tale eller lydopptak behandles digitalt, analyseres i frekvensdomenet og visualiseres slik at frekvensinnholdet i signalet blir lettere å tolke.

Den overordnede ideen er at et talesignal først punktprøves, deretter behandles med en diskret Fouriertransformasjon, før amplitudespekteret eller effektspekteret presenteres grafisk på skjerm. I den endelige løsningen skulle signalet komme fra innspilte lydopptak eller senere fra mikrofon og ADC. I dette prosjektet ble hovedfokuset lagt på å få FFT-beregning, visualisering på OLED og måling av kjøretid til å fungere på en mikrokontroller.

2 Prinsipiell løsning

2.1 Overordnet arkitektur

Den prinsipielle løsningen er basert på en klassisk signalbehandlingskjede, der et talesignal først representeres som et tidsdiskret signal før det analyseres i frekvensdomenet. Dette samsvarer med grunnideen i digital signalbehandling, der signaler kan representeres matematisk og behandles av systemer for å hente ut ny informasjon eller en ny representasjon av signalet [**DSPFirst**]. Den overordnede arkitekturen er vist i figur 2.



Figur 2: Prinsipiell signalbehandlingskjede for frekvensanalyse av tale.

Figur 2 viser hvordan talesignalet først behandles som et tidsdiskret signal $x[n]$. Deretter deles signalet inn i et utsnitt med N punktprøver før DFT/FFT brukes til å transformere signalet fra tidsdomenet til frekvensdomenet. Resultatet av transformasjonen er komplekse frekvenskomponenter $X[k]$, som videre kan brukes til å beregne amplitude- eller effektspekter. Til slutt kan spekteret brukes til å analysere hvilke frekvenskomponenter som dominerer i signalet, for eksempel topper som kan knyttes til formanter i tale.

2.2 Spektralanalyse

Et talesignal kan representeres som en tidsdiskret sekvens $x[n]$ etter sampling. For å analysere hvilke frekvenser signalet består av, kan et utsnitt av signalet transformeres til frekvensdomenet ved hjelp av DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N} \tag{1}$$

der N er antall punktprøver i FFT-blokken, $x[n]$ er inngangssignalet og $X[k]$ er den komplekse frekvenskomponenten ved frekvensindeks k.

I implementasjonen ble det valgt en FFT-lengde på

$$N = 256 \tag{2}$$

Dette gir et kompromiss mellom frekvensoppløsning, minnebruk og beregningstid på mikrokontrolleren.

2.3 Amplitude- og effektspekter

FFT-resultatet består av komplekse verdier med real- og imaginærdel. Amplituden til hver frekvenskomponent kan beregnes som

$$|X[k]| = \sqrt{\mathrm{Re}\{X[k]\}^2 + \mathrm{Im}\{X[k]\}^2} \tag{3}$$

I prosjektbeskrivelsen er det effektspekteret som er ønsket, altså det kvadrerte amplitudespekteret:

$$|X[k]|^2 = \mathrm{Re}\{X[k]\}^2 + \mathrm{Im}\{X[k]\}^2 \tag{4}$$

I den realiserte koden ble amplitudespekteret brukt som visualiseringsgrunnlag, siden dette ga en tydelig og stabil visning på OLED-skjermen. Dette avviker noe fra den ideelle kravformuleringen, men demonstrerer fortsatt selve FFT-operasjonen og frekvensanalysen.

2.4 Visualisering på OLED

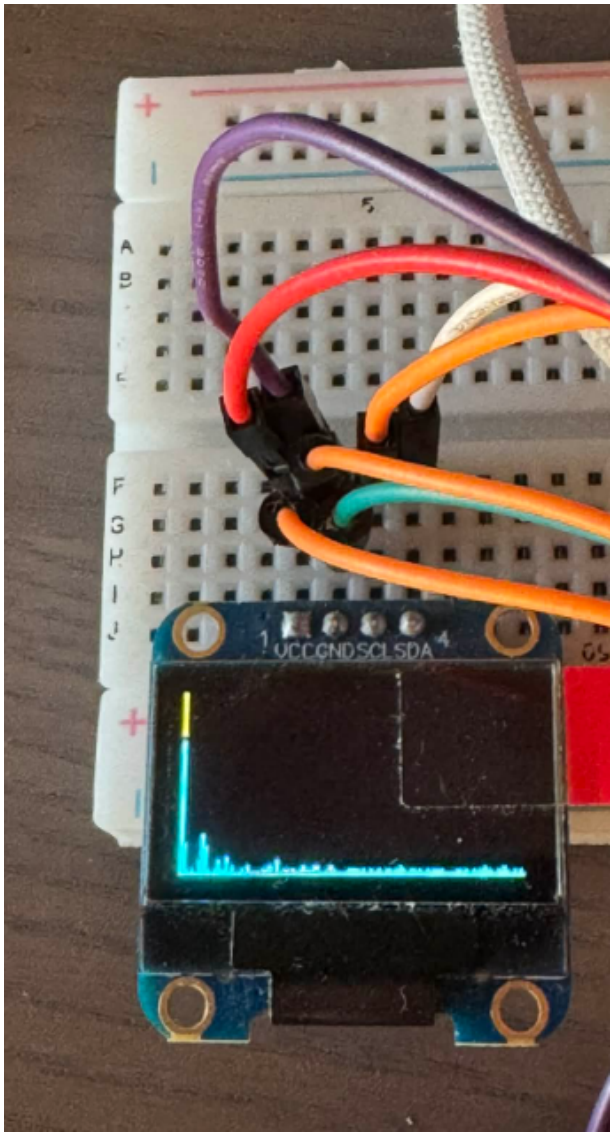
OLED-skjermen har oppløsning 128×64 piksler. Siden en reell FFT med $N = 256$ gir

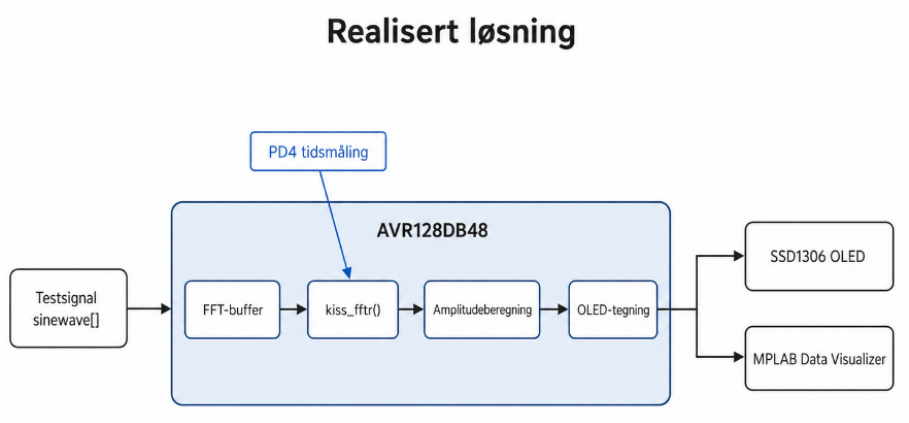
$$\frac{N}{2} + 1 = 129 \tag{5}$$

frekvensverdier, passer FFT-resultatet godt til skjermbredden. Hver frekvenskomponent tegnes som en vertikal søyle, der søylehøyden bestemmes av amplituden til FFT-bin-en.

3 Realisering og test

Systemet ble implementert på AVR128DB48 ved å ta utgangspunkt i Microchip sitt kissFFT-eksempel og en SSD1306 OLED-driver. Programmet initialiserer klokke, USART og OLED-skjerm, fyller en FFT-buffer med et kjent testsignal, utfører FFT og viser resultatet som et spektrum på skjermen. Den realiserte løsningen er vist i figur 3.



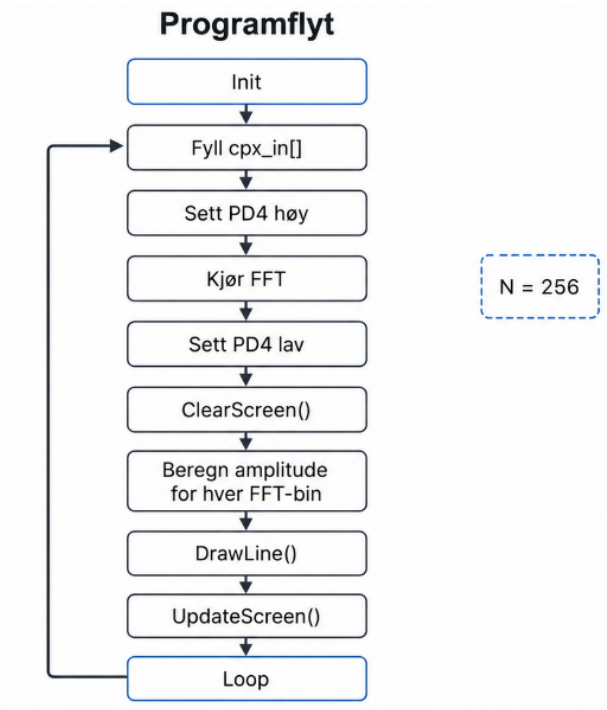


Figur 3: Realisert løsning for FFT-basert spektrumanalyse, med testsignal som inngang og visualisering på OLED og PC.

Figur 3 viser hvordan systemet ble bygget opp i praksis. Et syntetisk testsignal ble lagt inn i en FFT-buffer, behandlet med `kissFFT`, og deretter brukt til amplitudeberegning. Resultatet ble sendt både til OLED-skjermen og til MPLAB Data Visualizer for kontroll og testing.

3.1 FFT-implementasjon

FFT-en ble implementert ved hjelp av funksjonen `kiss_fftr()`, som er beregnet for reelle inngangssignaler. Inngangsbufferen har lengde 256, og utgangen består av komplekse verdier for de positive frekvenskomponentene. Den praktiske programflyten er vist i figur 4.



Figur 4: Programflyt for FFT-beregning, amplitudeberegning og oppdatering av OLED-skjermen.

Figur 4 viser hovedsekvensen i programmet. Først initialiseres systemet, deretter fylles FFT-

bufferen med inngangsdata. Etter dette kjøres FFT-en, amplitudeverdiene beregnes, og resultatet tegnes på OLED-skjermen før prosessen gjentas.

```
#define NFFT 256

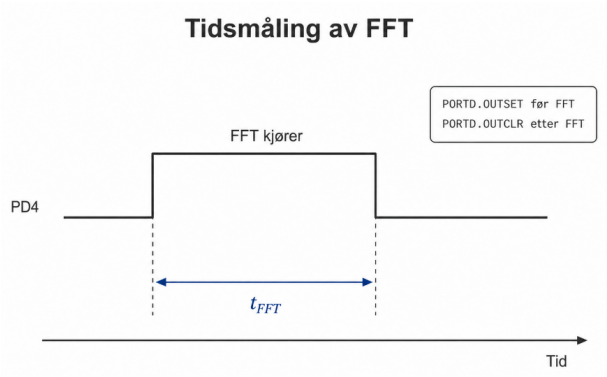
kiss_fft_scalar cpx_in[NFFT];

kiss_fftr_cfg cfg = kiss_fftr_alloc(NFFT, 0, NULL, NULL);
kiss_fft_cpx cpx_out[(NFFT / 2) + 1];
```

Testsignalet ble hentet fra `sinewave[]`. Dette gjorde det mulig å verifisere at FFT-beregningen og visualiseringen fungerte før videre arbeid med lydfil og ADC.

3.2 Tidsmåling

For å dokumentere kjøretiden til FFT-operasjonen ble pinne PD4 brukt som målepinne. Prinsippet for tidsmålingen er vist i figur 5. Pinnen settes høy rett før FFT-kallet og lav rett etterpå:



Figur 5: Prinsipp for tidsmåling av FFT-operasjonen ved hjelp av en GPIO-pinne.

Figur 5 viser hvordan pulsbredden på PD4 tilsvarer tiden mikrokontrolleren bruker på selve FFT-beregningen. Når pinnen er høy, kjøres FFT-funksjonen, og når pinnen settes lav igjen, er beregningen ferdig.

```
PORTD.OUTSET = PIN4_bm;
kiss_fftr(cfg, cpx_in, cpx_out);
PORTD.OUTCLR = PIN4_bm;
```

Ved å måle pulsbredden på denne pinnen med oscilloskop eller logikkanalysator kan kjøretiden til FFT-en bestemmes. Dette er relevant fordi prosjektet stiller krav om at tidsbruken til sentrale programdeler skal dokumenteres.

3.3 OLED-visualisering

Etter FFT-beregningen går programmet gjennom frekvenskomponentene og beregner amplituden:

```
pwr = sqrt(cpx_out[n].r * cpx_out[n].r +
           cpx_out[n].i * cpx_out[n].i);
```

Verdien begrenses deretter til skjermhøyden og tegnes som en vertikal linje:

```
uint8_t h = (uint8_t)pwr;
if (h > 63) h = 63;

SSD1306_DrawLine(n, n, 63 - h, 63);
```

OLED-skjermen oppdateres til slutt med

```
SSD1306_UpdateScreen(SSD1306_ADDR);
```

Dette ga en fungerende spektrumvisning på OLED-skjermen.

3.4 Data Visualizer

I tillegg til OLED-visning ble realdel, imaginærdel, frekvensindeks og amplitude sendt til MPLAB Data Visualizer:

```
variableWrite_SendFrame(watch_real, watch_imag, cnt, pwr);
```

Dette gjorde det mulig å observere FFT-resultatet også på PC, og ga en ekstra kontroll på at FFT-beregningen produserte data.

3.5 Forsøk med lydfil

Det ble også gjort forsøk på å konvertere en lydfil til C-array for bruk som FFT-inngang. Lydfilen måtte først konverteres fra MP3-format til rå PCM-data, fordi FFT-en krever tidsdomene-samples og ikke komprimerte MP3-bytes.

Arbeidet med å generere en korrekt `int16_t`-array fra lydfil ble påbegynt, men ble ikke ferdigstilt i en stabil løsning på mikrokontrolleren. Det ble observert at OLED-visningen ble flat eller lite informativ når lydarrayet ble brukt. Årsaken vurderes å være knyttet til skalering, valg av lydutsnitt og/eller hvordan amplitudeverdiene ble mappet til skjermhøyde.

På grunn av tidsbegrensning ble den endelige demonstrasjonen derfor gjennomført med et syntetisk testsignal. Dette ga en mer stabil verifikasjon av selve FFT-kjeden, OLED-visningen og tidsmålingen.

4 Konklusjon

I dette designprosjektet ble det utviklet en fungerende FFT-basert spektrumanalysator på AVR128DB48. Systemet klarer å utføre en 256-punkts reell FFT ved hjelp av kissFFT, sende FFT-data til MPLAB Data Visualizer og vise et spektrum på en SSD1306 OLED-skjerm.

Den realiserte løsningen oppfyller ikke alle kravene i prosjektet fullt ut, siden integrasjonen med ferdiginnspilt lydfil ikke ble ferdigstilt. Likevel ble de sentrale delene av systemet implementert og verifisert: FFT-beregning, visuell presentasjon av spektrum og måling av kjøretid.

Videre arbeid vil være å fullføre konverteringen fra lydfil til PCM-basert C-array, forbedre skaleringen av amplituden eller effektspekteret til OLED-skjermen, og til slutt erstatte testsignalet med sanntids sampling fra ADC. En mer komplett løsning bør også bruke effektspekteret direkte, eventuelt med logaritmisk skalering, siden dette vil gi bedre visning av formanter i talesignaler.

Samlet sett viser prosjektet at det er mulig å implementere grunnleggende spektralanalyse på AVR128DB48, men at videre arbeid kreves for å gjøre løsningen robust for faktiske taleopptak.

5 Takk

Takk til faglærer og medstudenter for veiledning underveis i prosjektet. ChatGPT ble brukt til feilsøking, forklaring av FFT, hjelp med kodeforståelse og formulering av rapporttekst i tillegg til hjelp med figurer.

6 Vedlegg

```c
/*
 * File:   DP12_main.c
 * Author: temma
 *
 * Created on 30 April 2026, 13:33
 */

#include <xc.h>
#include "filter/kiss_fftr.h"
#include "filter/sine.h"
#include "peripherals/clock/clkctrl.h"
#include "peripherals/usart/usart3.h"
#include "peripherals/data_streamer/data_streamer.h"
#include "display/ssd1306.h"
#include <avr/cpufunc.h>
#include <math.h>

#define NFFT 256

kiss_fft_scalar cpx_in[NFFT];

int main(void)
{
    clkctrl_init();
    SSD1306_Init(SSD1306_ADDR);
    USART3_Initialize();

    // Pin for tidsmaaling av FFT
    PORTD.DIRSET = PIN4_bm;

    volatile int out = (NFFT / 2) + 1;

    kiss_fftr_cfg cfg = kiss_fftr_alloc(NFFT, 0, NULL, NULL);
    kiss_fft_cpx cpx_out[out];

    for (int m = 0; m < NFFT; m++) {
        cpx_in[m] = sinewave[m];
    }

    volatile int16_t watch_real;
    volatile int16_t watch_imag;
    volatile uint32_t pwr;
    volatile uint16_t cnt;

    PORTD.OUTSET = PIN4_bm;
    kiss_fftr(cfg, cpx_in, cpx_out);
    PORTD.OUTCLR = PIN4_bm;

    while (1) {
        SSD1306_ClearScreen();
        for (int n = 0; n < out; n++) {
            cnt = n;
            watch_real = cpx_out[n].r;
            watch_imag = cpx_out[n].i;

            pwr = sqrt(cpx_out[n].r * cpx_out[n].r + cpx_out[n].i * cpx_out[n].i
    );

            uint8_t h = (uint8_t)pwr;
            if (h > 63) h = 63;
```

```
60              SSD1306_DrawLine(n, n, 63 - h, 63);
61
62              variableWrite_SendFrame(watch_real, watch_imag, cnt, pwr);
63          }
64          SSD1306_UpdateScreen(SSD1306_ADDR);
65      }
66
67      kiss_fft_cleanup();
68  }
```

Kodeutdrag 1: Main.c