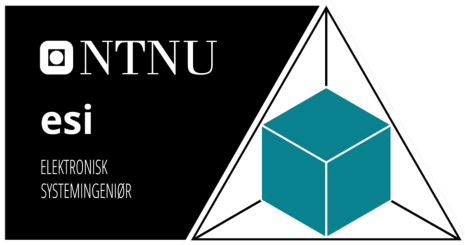
Designnotat 11



Tittel: Digitalt filter - Filtrering av pipetone

Forfatter(e): Temmam Nabil Altaie

Versjon: 1.0

Dato: 30. mars 2026

Innhold

1 Problembeskrivelse **2**
1.1 Innledning . . . 2
1.2 Systemoversikt . . . 2

2 Prinsipiell løsning **3**
2.1 Identifisering av pipetonefrekvensen . . . 3
2.2 Valg av filtertype . . . 3
2.3 Systemfunksjon . . . 3
2.4 Pol- og nullpunktdiagram . . . 4
2.5 Frekvensrespons . . . 5

3 Realisering og test **6**
3.1 Beregning av koeffisienter . . . 6
3.2 MATLAB-verifikasjon . . . 6
3.3 Implementasjon på mikrokontroller . . . 6
3.3.1 Q15 fast-point skalering . . . 6
3.3.2 C-implementasjon . . . 7

4 Konklusjon **8**

Takk **9**

1 Problembeskrivelse

1.1 Innledning

Et lydopptak fra selskapets produksjonsstudio inneholder en irriterende pipetone som ikke ble tilstrekkelig fjernet ved bruk av analoge filtre. Oppdraget er å fjerne pipetonen digitalt på en mikrokontroller (AVR128DB48), med krav om skarpt og sterkt demping av nettopp den aktuelle frekvensen – uten å påvirke resten av lydsignalet.

Dette designprosjektet beskriver identifisering av pipetonefrekvensen, design av et IIR notch-filter i MATLAB, og implementasjon av filteret i C på mikrokontrolleren.

1.2 Systemoversikt

Figur 1 viser den overordnede signalfløyen i systemet. Det analoge lydsignalet konverteres til digital form via ADC-en på mikrokontrolleren, behandles av IIR notch-filteret, og rekonstrueres til et analogt signal via DAC-en.

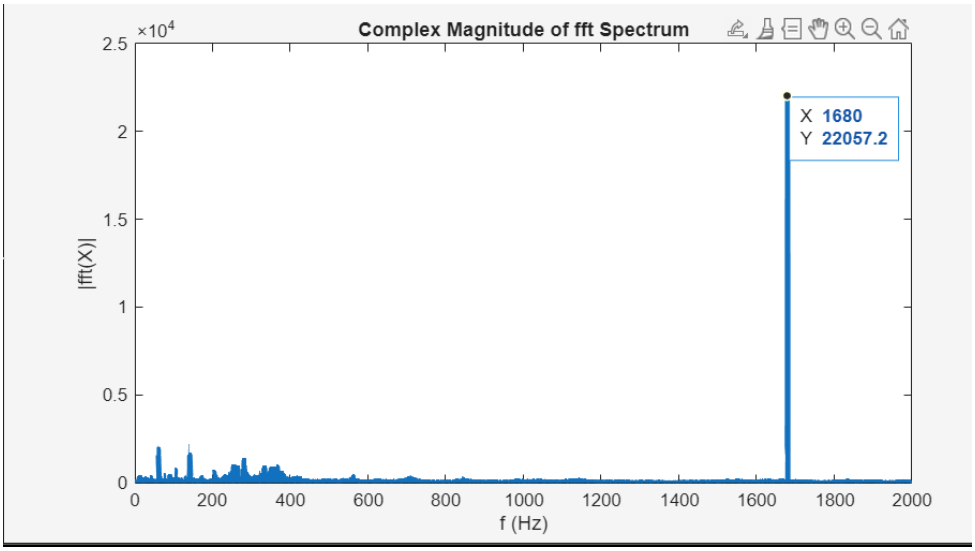


Figur 1: Overordnet systemoversikt: lydsignalet med pipetone behandles digitalt av IIR notch-filteret på mikrokontrolleren.

2 Prinsipiell løsning

2.1 Identifisering av pipetonefrekvensen

Lydfilen `Lydsignal_DP11.wav` ble lastet inn i MATLAB og analysert med FFT (`fft()`). Samplingsfrekvensen er $f_s = 44100$ Hz.



Figur 2: FFT-spektrum av lydsignalet. En tydelig topp er synlig ved 1680 Hz.

FFT-analysen avslørte en dominerende frekvenskomponent ved:

$$f_0 = 1680 \text{ Hz}$$

Filterdesign

2.2 Valg av filtertype

Et IIR (Infinite Impulse Response) notch-filter ble valgt fremfor FIR, da IIR gir et skarpt og sterkt notch med svært få koeffisienter (2. orden). FIR ville krevd et stort antall tapper for tilsvarende selektivitet (som i DP9: 165 taps).

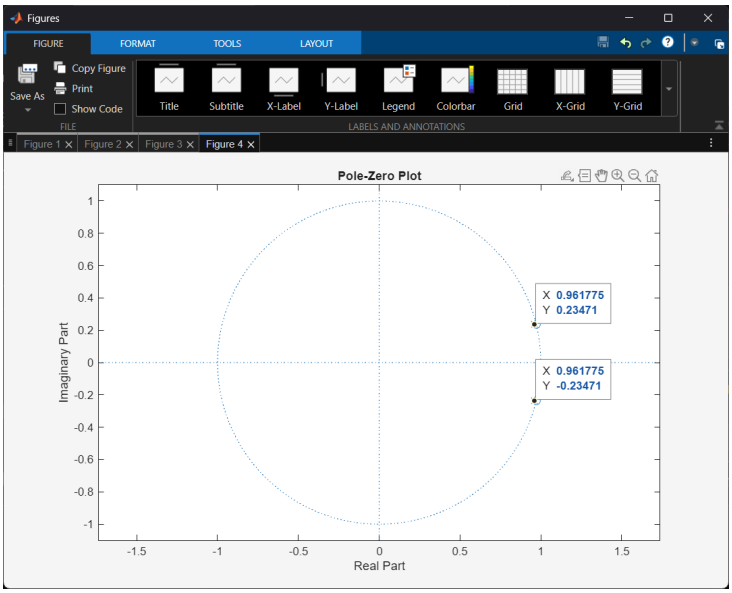
2.3 Systemfunksjon

IIR notch-filteret har følgende systemfunksjon:

$$H(z) = \frac{1 - 2\cos(\hat{\omega}_0)\, z^{-1} + z^{-2}}{1 - 2r\cos(\hat{\omega}_0)\, z^{-1} + r^2 z^{-2}}$$

der $\hat{\omega}_0$ er den normaliserte vinkelfrekvensen og r er polradius som styrer notch-bredden.

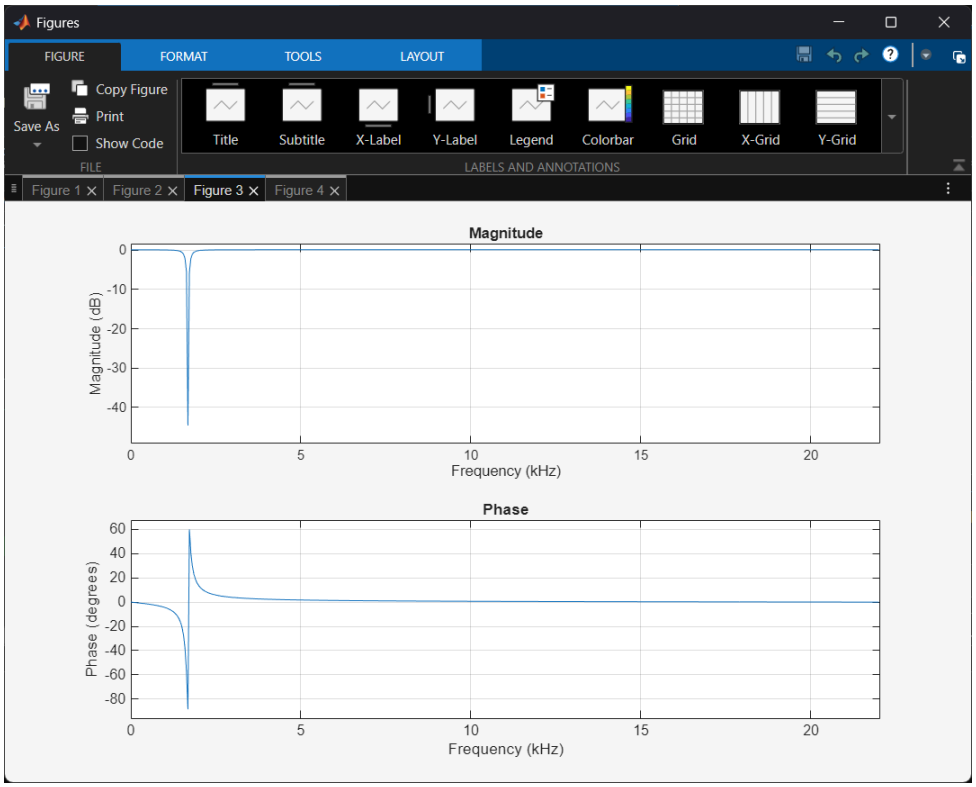
2.4 Pol- og nullpunktdiagram



Figur 3: Pol- og nullpunktdiagram. Nullpunkter på enhetssirkelen ved $\hat{\omega}_0$, poler like innenfor ved $r = 0{,}99$.

Nullpunktene ligger på enhetssirkelen ved $\pm\hat{\omega}_0$, som gir eksakt demping ved 1680 Hz. Polene plasseres rett innenfor ved radius $r = 0{,}99$, noe som gjør notchet skarpt og holder filteret stabilt.

2.5 Frekvensrespons



Figur 4: Amplituderespons for IIR notch-filteret. Dempingen ved 1680 Hz er ca. 45 dB.

3 Realisering og test

3.1 Beregning av koeffisienter

Den normaliserte vinkelfrekvensen beregnes som:

$$\hat{\omega}_0 = \frac{2\pi f_0}{f_s} = \frac{2\pi \cdot 1680}{44100} \approx 0{,}2394 \text{ rad}$$

Polradius ble satt til $r = 0{,}99$ for å oppnå et skarpt notch med minimal påvirkning på naborfrekvenser. Koeffisientene blir:

$$b = [1,\ -2\cos(\hat{\omega}_0),\ 1] = [1,\ -1{,}94296,\ 1]$$
$$a = [1,\ -2r\cos(\hat{\omega}_0),\ r^2] = [1,\ -1{,}92353,\ 0{,}9801]$$

3.2 MATLAB-verifikasjon

Lydfilen ble filtrert i MATLAB med `filter()`-funksjonen:

```
f0 = 1680;
omega_0 = 2*pi*(f0/fs);   % -> 0.2394 rad
r = 0.99;

b_koeffs = [1, -2*cos(omega_0), 1];
a_koeffs = [1, -2*r*cos(omega_0), r^2];

y = filter(b_koeffs, a_koeffs, x_pipetone);
soundsc(y, fs, 16);
```

Kodeutdrag 1: MATLAB: IIR notch-filter

Etter filtrering var pipetonen ikke lenger hørbar. MATLAB-resultatene bekrefter at filterdesignet er korrekt.

3.3 Implementasjon på mikrokontroller

3.3.1 Q15 fast-point skalering

Mikrokontrolleren (AVR128DB48) har ingen flytepunkts-FPU, så koeffisientene representeres i Q15-format: en `int16_t`-verdi v representerer desimaltallet $v/32768$.

Siden $b_0 = b_2 = 1{,}0$ og $|a_1| \approx 1{,}92$ overskrider Q15-maksimum ($\approx 0{,}9999$), skaleres alle koeffisienter med $\frac{1}{2}$:

$$B_0 = \lfloor 0{,}5 \cdot 32768 \rfloor = 16384, \quad B_1 = -31858, \quad B_2 = 16384$$
$$A_1 = -31506, \quad A_2 = 16056$$

Etter summering skiftes resultatet $\gg 15$ (fjerner Q15-skaleringen), deretter $\ll 1$ for å kompensere $\frac{1}{2}$-skaleringen.

3.3.2 C-implementasjon

```c
static int16_t x1 = 0, x2 = 0, y1 = 0, y2 = 0;

static int16_t compute_iir(int16_t x)
{
    int32_t acc = 0;

    acc += B0 * x;
    acc += B1 * x1;
    acc += B2 * x2;
    acc -= A1 * y1;
    acc -= A2 * y2;

    acc = acc >> 15;    /* fjern Q15-skalering */
    acc = acc << 1;     /* kompenser 0.5-skalering */

    x2 = x1;  x1 = x;
    y2 = y1;  y1 = (int16_t)acc;

    return (int16_t)acc;
}
```

Kodeutdrag 2: compute_iir() – IIR notch-filter i Q15

```c
ISR(TCA0_OVF_vect)
{
    int16_t x = (int16_t)adc_get_sample() - 512;
    int16_t y = compute_iir(x);

    int32_t out = (int32_t)y + 512;
    if (out < 0)    out = 0;
    if (out > 1023) out = 1023;

    dac_write((uint16_t)out);
    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
}
```

Kodeutdrag 3: ISR – ADC → IIR → DAC, 44100 ganger/sek

4 Konklusjon

Et 2. ordens IIR notch-filter ble designet for å fjerne en pipetone på $f_0 = 1680$ Hz fra et lydopptak med samplingsfrekvens $f_s = 44100$ Hz. Filteret gir ca. 45 dB demping ved pipetonefrekvensen med minimal påvirkning på øvrige frekvenser.

MATLAB-verifikasjon bekreftet at pipetonen er hørbar borte etter filtrering. C-implementasjonen på AVR128DB48 bruker Q15 fast-point aritmetikk og bygger på ISR-strukturen fra ERT28/DP9, men erstatter ringbufferet og 165 FIR-tapper med kun fire tilstandsvariabler og fem koeffisienter.

Takk

Takk til faglærer og medstudenter for veiledning og diskusjon underveis i prosjektet.