

Designnotat 9

Tittel: FIR-Filter	
Forfattere: Temmam Nabil Altaie	
Versjon: 2.0	Dato: 05.06.26

Innhold

1 Problembeskrivelse **2**

2 Prinsipiell løsning **3**

2.1 Tidsdiskret signalmodell . . . 3
2.2 FIR-filter . . . 3
2.3 Frekvensrespons . . . 3
2.4 Lavpassfiltrering . . . 3
2.5 Lineær fase . . . 3

3 Implementasjon og test **4**

3.1 Implementasjon i numerisk verifikasjonsmiljø . . . 4
3.2 Frekvensrespons og filterverifikasjon . . . 4
3.3 Spektrumanalyse av lydsignal . . . 5
3.4 Effektspektraltetthet før og etter filtrering . . . 5
3.5 Implementasjon på mikrokontroller . . . 6
3.6 Filterorden og prosesseringstid . . . 7

4 Konklusjon **8**

5 Takk **8**

Referanseliste **9**

6 Kodevedlegg **10**

6.1 Matlab.m . . . 10
6.2 Main.c . . . 11

1 Problembeskrivelse

I dette designprosjektet skal det utvikles et system som tar inn et analogt lydsignal og gir ut en forbedret utgave med redusert støyinnhold. Lydsignalet er tilgjengelig som en `wav`-fil og representerer et realistisk radiosignal med både ønsket informasjon og uønsket støy.



Figur 1: Illustrasjon av hvordan systemet implementeres

Av hensyn til kostnad og fleksibilitet skal løsningen være mikrokontrollerbasert. Problemstillingen innebærer dermed å forbedre lydkvaliteten innenfor de praktiske begrensningene som følger av digital signalbehandling i et innebygd system.

2 Prinsipiell løsning

Den prinsipielle løsningen bygger på grunnleggende teori for diskrete lineære tidsinvariante (LTI) systemer og digital filtrering, slik presentert i kapittel 5 og 6 i *DSP First* [1].

2.1 Tidsdiskret signalmodell

Et analogt lydsignal kan representeres som et tidsdiskret signal etter punktprøving. Dette signalet beskrives som en sekvens x[n] der n er diskret tidsindeks. Signalbehandlingen utføres videre utelukkende i det diskrete tidsdomenet.

Systemet som brukes for støydemping modelleres som et lineært tidsinvariant (LTI) system. Et slikt system er fullstendig karakterisert av sin impulsrespons $h[n]$ [1, Kap. 5]. Sammenhengen mellom inngang og utgang er gitt ved konvolusjon: y[n] = x[n] * h[n]

Dette uttrykket beskriver hvordan hver inngangsprøve bidrar til utgangen, og danner grunnlaget for digital filtrering.

2.2 FIR-filter

I dette prosjektet benyttes et filter med endelig impulsrespons (FIR). For et FIR-filter er impulsresponsen endelig lang, og konvolusjonen kan skrives som en endelig sum: $y[n] = \sum_{k=0}^{M} b[k]\, x[n-k]$ der $b[k]$ er filterkoeffisientene og M er filterorden [1, Kap. 6].

FIR-filtre er særlig egnet i lydapplikasjoner fordi de kan realiseres med lineær fase og er alltid stabile.

2.3 Frekvensrespons

Frekvensresponsen til et diskret LTI-system er definert som den diskrete tids-Fouriertransformen av impulsresponsen: $H(e^{j\omega}) = \sum_{k=0}^{M} b[k]e^{-j\omega k}$

Frekvensresponsen beskriver hvordan ulike frekvenskomponenter i inngangssignalet påvirkes av filteret. Amplituderesponsen $|H(e^{j\omega})|$ avgjør hvilke frekvenser som forsterkes eller dempes.

2.4 Lavpassfiltrering

For å redusere høyfrekvent støy benyttes et lavpassfilter. Ideelt ønskes en frekvensrespons som tilfredsstiller: $|H(e^{j\omega})| \approx 1$ for lave ω $|H(e^{j\omega})| \approx 0$ for høye ω

Et slikt filter bevarer lavfrekvent innhold som tale og musikk, samtidig som uønskede høyfrekvente komponenter dempes. Dette er i tråd med grunnleggende lavpassfiltrering beskrevet i [1, Kap. 6].

2.5 Lineær fase

Et FIR-filter med symmetriske koeffisienter har lineær fase [1, Kap. 6]. Fasen kan da skrives som: $\angle H(e^{j\omega}) = -\omega\tau$ der τ er konstant gruppedelay. Lineær fase er ønskelig i lydbehandling fordi tidsrelasjoner mellom frekvenskomponenter bevares, noe som gir minimal faseforvrengning av signalet.

3 Implementasjon og test

Dette kapittelet beskriver hvordan den prinsipielle løsningen er realisert i praksis, samt hvordan systemet er verifisert gjennom både objektive målinger og subjektiv lyttetest.

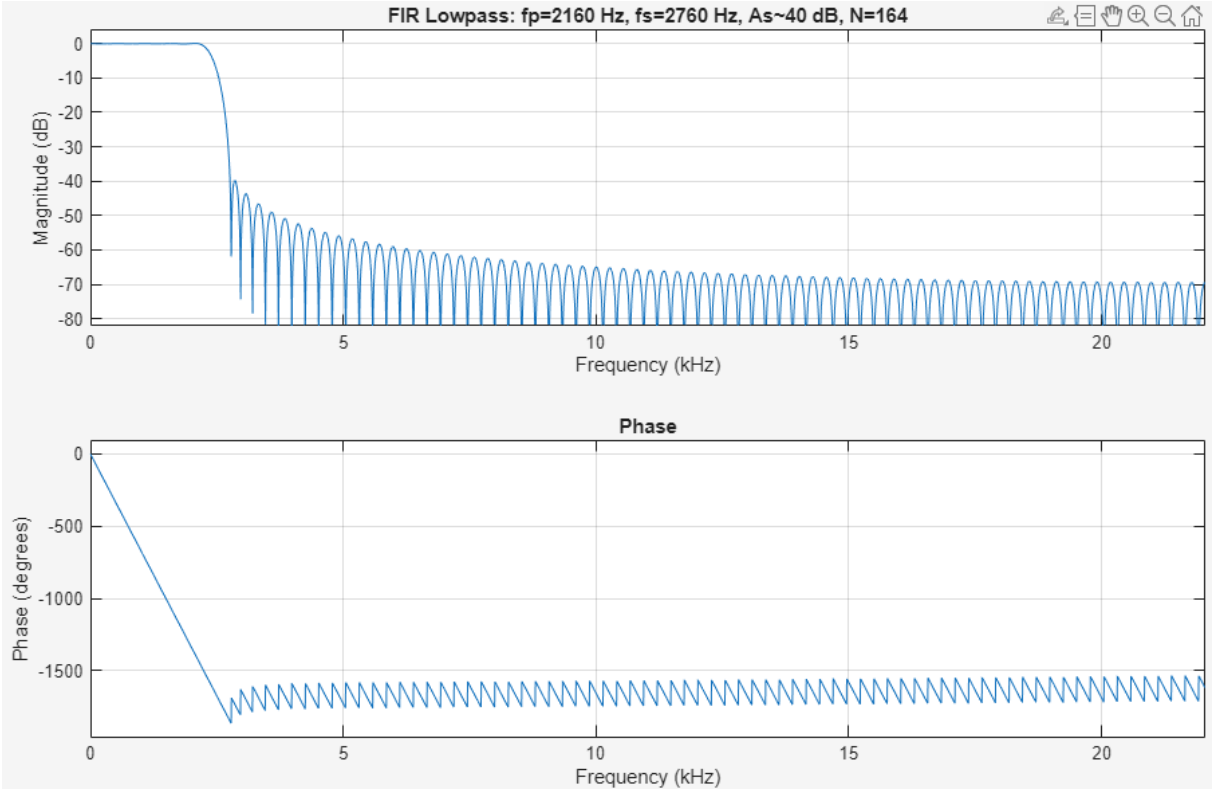
3.1 Implementasjon i numerisk verifikasjonsmiljø

Før implementasjon på mikrokontroller ble systemet verifisert i et numerisk miljø. Lydsignalet ble lest inn fra fil og behandlet som et tidsdiskret signal. For å sikre stabile og sammenlignbare resultater ble signalet først gjort monofonisk, DC-komponenten fjernet, og amplituden normalisert.

FIR-filteret ble deretter anvendt på signalet ved konvolusjon mellom inngangssignalet og filterets koeffisienter. Dette ga et filtrert utgangssignal som kunne analyseres videre både i tids- og frekvensdomenet.

3.2 Frekvensrespons og filterverifikasjon

For å verifisere at filteret oppfører seg som forventet, ble filterets frekvensrespons analysert. Figur 2 viser amplituderesponsen til FIR-lavpassfilteret.

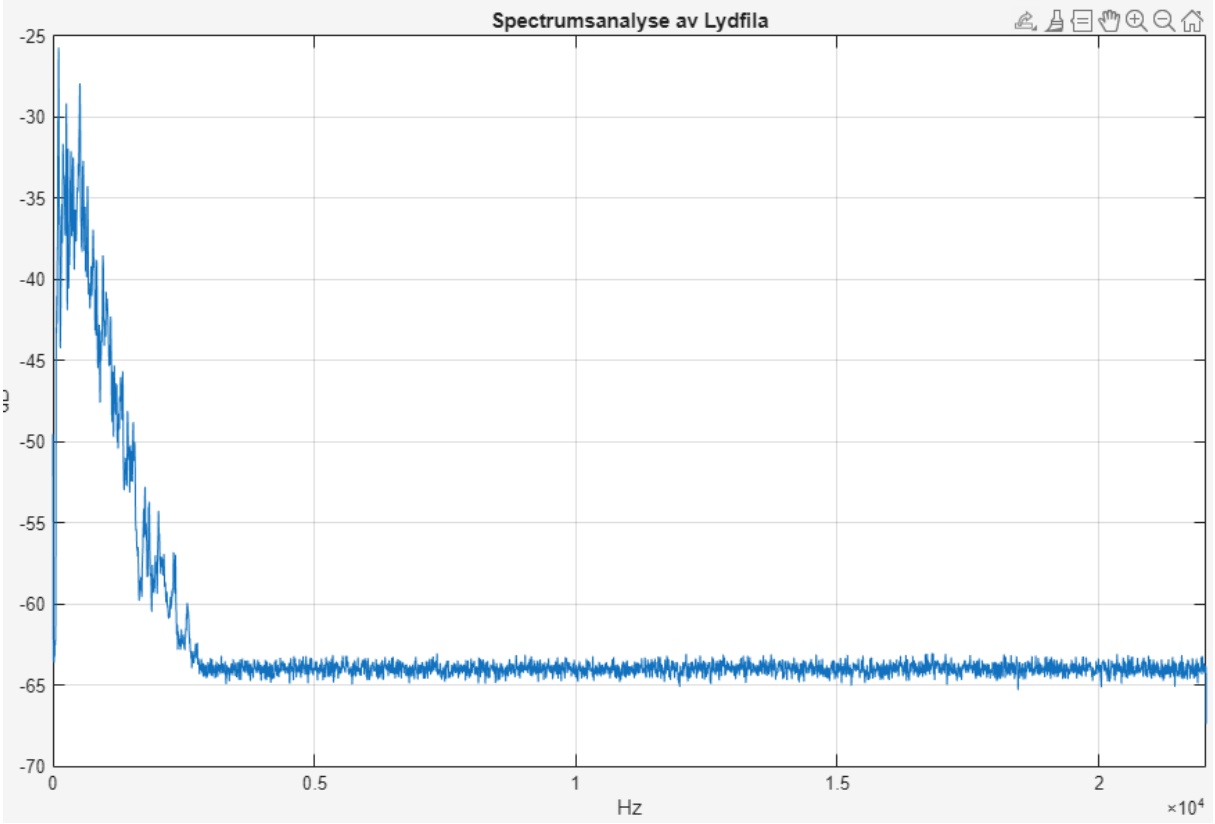


Figur 2: Frekvens- og faserespons for FIR-lavpassfilter. Den lineære faseresponsen indikerer konstant gruppedelay.

Faseresponsen er vist sammen med amplituderesponsen i figur 2. Det observeres en tilnærmet lineær fase i passbåndet, noe som indikerer konstant gruppedelay og bekrefter at filteret egner seg for lydbehandling.

3.3 Spektrumanalyse av lydsignal

For å undersøke hvordan filtreringen påvirker det faktiske lydsignalet, ble spekteret analysert. Figur 3 viser spektrumanalysen av det opprinnelige lydsignalet.

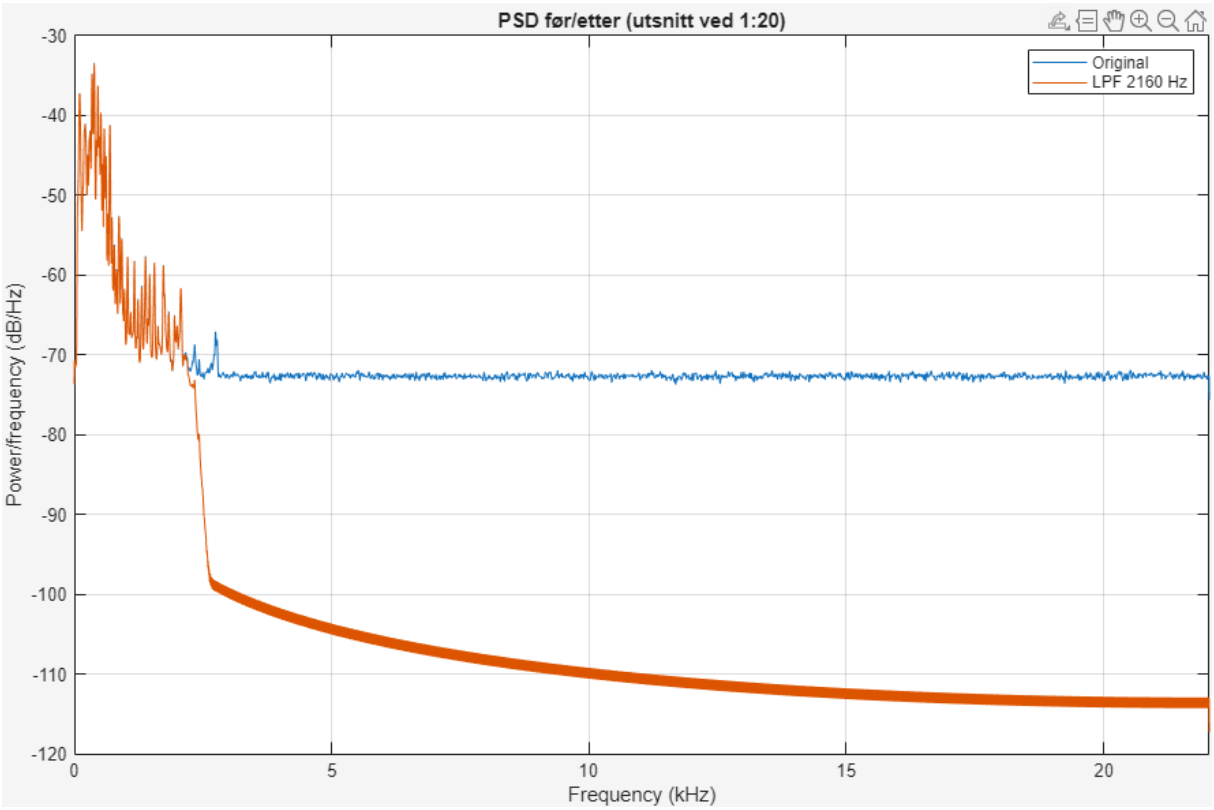


Figur 3: Spektrumanalyse av det opprinnelige lydsignalet. Det observeres betydelig energi i lavere frekvenser, mens høye frekvenser virker å ha konstant/periodisk bølge .

Ved sammenligning med spekteret etter filtrering observeres en tydelig reduksjon av energi i de høyere frekvensområdene, mens lavfrekvent innhold i stor grad bevares.

3.4 Effektspektraltetthet før og etter filtrering

For ytterligere dokumentasjon ble effektspektraltetthet (PSD) estimert for et representativt tidsutsnitt av lydsignalet før og etter filtrering. Resultatet er vist i figur 4.

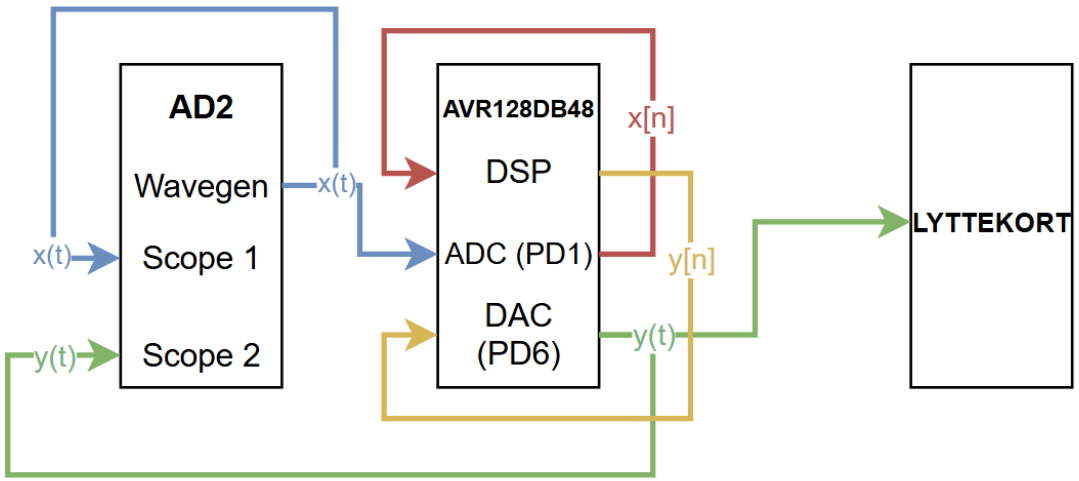


Figur 4: Amplituderesponsen for Lydfila før og etter filtrering.

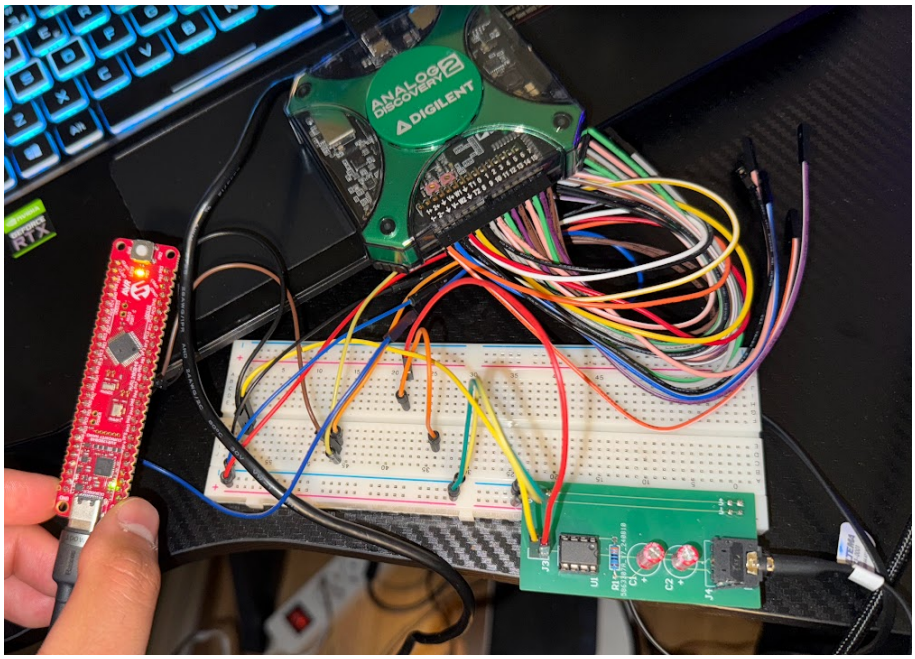
Figuren viser at det filtrerte signalet har betydelig lavere spektral effekt i høye frekvenser sammenlignet med det opprinnelige signalet, noe som bekrefter effektiv støydemping.

3.5 Implementasjon på mikrokontroller

Etter vellykket numerisk verifikasjon ble løsningen implementert på mikrokontrolleren AVR128DB48. Filterkoeffisientene ble kvantisert til fastpunktformat og benyttet i en FIR-struktur som kjører i sanntid.



Figur 5: Blokkskjema som viser signalflyten i systemet.



Figur 6: Fysisk oppkobling mellom AD2, AVR og lyttekort.

Analog Discovery 2 ble brukt både som signalgenerator og måleinstrument. Wavegen-modulen genererte et testsignal som ble koblet til ADC-inngangen på mikrokontrolleren. Det digitaliserte signalet ble deretter filtrert før det ble sendt ut igjen via DAC.

Utgangen fra DAC ble koblet til et lyttekort, slik at det filtrerte signalet kunne evalueres auditivt. Denne oppkoblingen muliggjorde direkte sammenligning mellom signal før og etter filtrering i sanntid.

3.6 Filterorden og prosesseringstid

MATLABs innebygde `lowpass()`-funksjon velger filterorden automatisk basert på ønsket stoppbånddempning og bredden på transisjonsbåndet. Med en kuttfrekvens på 800 Hz og en samplingsfrekvens på 44 100 Hz er den normaliserte kuttfrekvensen lav ($\hat{\omega}_c \approx 0{,}114\pi$), noe som krever et relativt høyt antall koeffisienter for å oppnå tilstrekkelig bratt utrulling. Resultatet er et filter med god stoppbånddempning, men spørsmålet er om alle koeffisientene bidrar meningsfylt til den opplevde lydkvaliteten.

I praksis avtar gevinsten av å øke filterordenen. Når støyen allerede er godt dempet med et moderat antall koeffisienter, gir ekstra koeffisienter kun marginalt brattere utrulling i transisjonsbåndet, noe som knapt er hørbart for et menneskelig øre. Et filter med lavere orden, for eksempel halvparten av det MATLAB valgte, ville trolig gitt tilnærmet samme subjektive lydkvalitet.

Dette har direkte konsekvenser for sanntidsapplikasjonen. FIR-filtrering krever én multiplikasjon og én akkumulering per koeffisient per inngangsprøve. Ved $f_s = 44\,100$ Hz er samplingperioden kun $T_s \approx 22{,}7\ \mu$s. All behandling, inkludert ADC-avlesning, konvolusjonsberegning og DAC-skriving, må fullføres innen dette vinduet. På AVR128DB48, som er en 8-bits mikrokontroller, tar en 16-bits multiplikasjon typisk 2 klokkesykler. Med et høyt antall koeffisienter kan den totale beregningsbelastningen overskride den tilgjengelige samplingperioden. En lavere filterorden ville redusert denne risikoen betydelig og gitt mer marginal kapasitet til øvrige systemoppgaver.

4 Konklusjon

I dette designprosjektet ble det utviklet og implementert et digitalt FIR-lavpassfilter for reduksjon av støy i et lydsignal. Resultatene fra både numerisk verifikasjon og mikrokontrollerbasert testing viser at løsningen fungerer etter hensikten. Frekvens- og spektralanalyser bekrefter effektiv demping av høyfrekvent støy, og subjektiv lyttetest indikerer en merkbar forbedring i lydkvalitet.

Implementasjonen på mikrokontrolleren AVR128DB48 viste at filteret lar seg realisere i sanntid, og oppkoblingen med ADC, DAC og Analog Discovery 2 muliggjorde både objektiv måling og auditiv evaluering av systemet. Samlet sett vurderes prosjektet som vellykket.

Det finnes likevel flere forbedringsmuligheter. FIR-filteret har symmetriske koeffisienter, og implementasjonen kunne derfor vært optimalisert ved å utnytte denne symmetrien. En slik tilnærming ville redusert antall multiplikasjoner per sample og gitt en mer effektiv og ressursbesparende kode, noe som er særlig relevant i innebygde systemer.

Videre kunne valg av filtertype og filterkarakteristikk vært begrunnet og analysert enda grundigere. Selv om lavpassfilter viste seg å være hensiktsmessig for denne applikasjonen, kunne alternative filterstrategier vært vurdert for å oppnå ytterligere forbedring i støydemping.

Til slutt kunne kvaliteten på utgangssignalet fra DAC vært forbedret ytterligere gjennom mer avansert skalering, bedre fastpunktshåndtering eller supplerende analog filtrering. Dette kunne gitt et enda renere analogt utgangssignal og redusert gjenværende artefakter.

Disse punktene representerer naturlige videreutviklinger av prosjektet og gir et godt grunnlag for videre arbeid med digital signalbehandling i innebygde systemer.

5 Takk

ChatGPT ble brukt for forklaringer av begreper, tilbakemeldinger og forbedringspotensialer i designnotatet.

Referanseliste

[1] J. H. McClellan, R. W. Schafer og M. A. Yoder, *DSP First*, 2. utg., Pearson Education, 2017.

[2] Microchip Technology Inc., *AVR128DB28/32/48/64 Datasheet*, 2020. Tilgjengelig på: `https://ww1.microchip.com/downloads/en/DeviceDoc/AVR128DB28-32-48-64-DataSheet-DS40002247A.pdf`, lest 21. oktober 2025.

[3] Digilent Inc., *Analog Discovery 2 – Waveforms User Manual*, 2024. Tilgjengelig på: `https://digilent.com/reference/test-and-measurement/analog-discovery-2/start`, lest 21. oktober 2025.

[4] Institutt for elektroniske systemer, NTNU, *ERT28 – Digital signal behandling, supplerende ERT*, Læringsmateriell i emnet IELS202x, 2026.

6 Kodevedlegg

6.1 Matlab.m

```
%% DP9: FIR Lowpass design @ 2160 Hz + verifisering + eksport til MCU
clear; close all; clc;

%% 1) Last inn lyd
filename = "Radiolyd1.wav";
[x, Fs] = audioread(filename);
x = mean(x,2);
x = x - mean(x);
x = x / (max(abs(x)) + eps);

fprintf("Fs = %.0f Hz, lengde = %.1f s\n", Fs, length(x)/Fs);

%% 2) Spesifikasjon (du har valgt fc=2160 Hz)
fp = 2160;                                  % passband edge (Hz)
fsb = fp + 600;                                % stopband start (Hz)  ->
    overgangsbaand 200 Hz
As = 40;                                   % onsket stopbaandsdemping (dB)

%% 3) FIR-design med Kaiser (gir deg kontroll paa As og orden)
% Estimer orden fra Kaiser-formel
DeltaF = fsb - fp;                         % Hz
DeltaW = 2*pi*(DeltaF/Fs);                 % rad/sample
N = ceil((As - 8) / (2.285 * DeltaW));     % filterorden (antall taps = N+1)
if mod(N,2)==1, N = N+1; end               % gjor N partall -> pen linearfase/
    gruppedelay

% Kaiser beta
if As > 50
    beta = 0.1102*(As - 8.7);
elseif As >= 21
    beta = 0.5842*(As - 21)^0.4 + 0.07886*(As - 21);
else
    beta = 0;
end

% Cutoff legges midt i overgangsbaandet (klassisk valg)
fc = (fp + fsb)/2;

b = fir1(N, fc/(Fs/2), kaiser(N+1, beta), "low");   % FIR-koeffisienter

fprintf("FIR order N = %d (taps = %d), group delay     %d samples\n", N, N+1, N
    /2);

%% 4) Frekvensrespons (kontroll)
figure;
freqz(b, 1, 4096, Fs);
title(sprintf("FIR Lowpass: fp=%d Hz, fs=%d Hz, As~%d dB, N=%d", fp, fsb, As, N)
    );

%% 5) Filtrer et relevant utsnitt (f.eks. ved 1:20)
t0 = 80;  L = 15;
i0 = round(t0*Fs) + 1;
i1 = min(i0 + round(L*Fs) - 1, length(x));
seg = x(i0:i1);

% Offline: filtfilt gir 0-fase (bra for lyttetesting i MATLAB)
y_seg = filtfilt(b, 1, seg);
```

```
54
55 % Normaliser for rettferdig sammenligning
56 segN = seg   / (max(abs(seg))   + eps);
57 yN   = y_seg / (max(abs(y_seg)) + eps);
58
59 fprintf("\nSpiller ORIGINAL utsnitt...\n");
60 p1 = audioplayer(segN, Fs); play(p1); pause(8); stop(p1);
61
62 fprintf("Spiller FILTRERT (LP 800 Hz) utsnitt...\n");
63 p2 = audioplayer(yN, Fs); play(p2); pause(8); stop(p2);
64
65 %% 6) PSD for/etter (dokumentasjon i rapport)
66 nfft = 8192;
67 figure;
68 pwelch(seg,   hamming(4096), 2048, nfft, Fs); hold on
69 pwelch(y_seg, hamming(4096), 2048, nfft, Fs);
70 legend("Original", "LPF 2160 Hz"); grid on
71 title("PSD for/etter (utsnitt ved 1:20)");
72
73 %% 7) Eksport til Q15 (for mikrokontroller)
74 % Skaler til int16 (Q15). Pass paa at |b| summering kan gi overflow i
      akkumulering
75 b_q15 = int16(round(b * 32767));
76
77 % Lag en enkel C-header
78 hdr = "fir_coeffs_q15.h";
79 fid = fopen(hdr, "w");
80 fprintf(fid, "#pragma once\n#include <stdint.h>\n\n");
81 fprintf(fid, "#define FIR_TAPS %d\n", length(b_q15));
82 fprintf(fid, "static const int16_t fir_b_q15[FIR_TAPS] = {\n");
83 for k = 1:length(b_q15)
84     if k < length(b_q15)
85         fprintf(fid, "  %d,\n", b_q15(k));
86     else
87         fprintf(fid, "  %d\n", b_q15(k));
88     end
89 end
90 fprintf(fid, "};\n");
91 fclose(fid);
92
93 fprintf("\nSkrev Q15-koeffisienter til %s\n", hdr);
```

Kodeutdrag 1: Matlab kode for analyse og testing av filtertype, orden og FIR-koeffisienter.

6.2 Main.c

```
1
2 /*
3  * AVR128DB48 - sanntids FIR lowpass (Q15) via ADC->FIR->DAC
4  *
5  * PINOUT:
6  * - PD1: ADC inngang (AIN1)
7  * - PD6: DAC utgang
8  *
9  * Samplingsfrekvens: 16 kHz
10 * FIR: 165 taps, Q15-koeffisienter (fir_b_q15_const)
11 */
12
13 #define F_CPU 24000000UL
14
15 #include <avr/io.h>
```

```
16 #include <avr/interrupt.h>
17 #include <avr/cpufunc.h>
18 #include <stdint.h>
19
20 /* --- KONFIGURASJON --- */
21
22 #define SAMPLING_FREQUENCY 16000UL   // Punktprovingsfrekvens i Hz
23 #define N 165                         // Antall filtertapper (taps)
24
25 #define TIMER_TOP ((F_CPU / SAMPLING_FREQUENCY) - 1)
26
27 /* --- FIR KOEFFISIENTer (Q15) --- */
28 /* Disse er hentet fra din FIR_TAPS=165 liste (Q15). */
29 static const int16_t fir_b_q15_const[N] = {
30   -8,  -2,   5,  13,  21,  26,  29,  29,  24,  16,   4, -10, -24, -37, -46, -51,
31   -49, -40, -25,  -6,  17,  40,  60,  74,  81,  77,  62,  39,   8, -28, -63,
       -93,
32   -114, -122, -115, -93, -57,  -9,  43,  95, 139, 170, 181, 170, 136,  81,
33    11, -67, -144, -209, -254, -270, -253, -201, -119,  -13, 106, 224,
34   325, 395, 422, 397, 317, 186,  14, -184, -385, -565, -699, -762, -735,
35   -603, -362,  -15, 425, 934, 1482, 2035, 2554, 3004, 3351, 3570, 3645,
36   3570, 3351, 3004, 2554, 2035, 1482,  934, 425,  -15, -362, -603, -735,
37   -762, -699, -565, -385, -184,  14, 186, 317, 397, 422, 395, 325, 224,
38   106,  -13, -119, -201, -253, -270, -254, -209, -144, -67,  11,  81, 136,
39   170, 181, 170, 139,  95,  43,  -9, -57, -93, -115, -122, -114, -93, -63,
40   -28,   8,  39,  62,  77,  81,  74,  60,  40,  17,  -6, -25, -40, -49, -51,
       -46,
41   -37, -24, -10,   4,  16,  24,  29,  29,  26,  21,  13,   5,  -2,  -8
42 };
43
44 /* --- DATASTRUKTURER --- */
45
46 static int16_t b[N];   // Arbeidsbuffer for koeffisienter (Q15-format)
47
48 typedef struct {
49     int16_t buffer[N];
50     uint8_t writeIndex;     // 0..N-1 (N=165 passer i uint8_t)
51 } ringbuffer;
52
53 static ringbuffer fir_buffer;
54
55 /* --- PERIFERIENHETS-INITIALISERING --- */
56
57 static void clock_init(void)
58 {
59     // Sett systemklokken til 24 MHz
60     ccp_write_io((void *)&CLKCTRL.OSCHFCTRLA, CLKCTRL_FRQSEL_24M_gc);
61 }
62
63 static void adc_init(void)
64 {
65     // Konfigurer PD1 (AIN1) som analog inngang
66     PORTD.PIN1CTRL &= ~PORT_ISC_gm;
67     PORTD.PIN1CTRL |= PORT_ISC_INPUT_DISABLE_gc;
68     PORTD.PIN1CTRL &= ~PORT_PULLUPEN_bm;
69
70     VREF.ADC0REF = VREF_REFSEL_VDD_gc;
71
72     ADC0.CTRLC  = ADC_PRESC_DIV12_gc;      // ADC-klokke p  2 MHz (24MHz/12)
73     ADC0.MUXPOS = ADC_MUXPOS_AIN1_gc;
74     ADC0.CTRLA  = ADC_ENABLE_bm | ADC_RESSEL_10BIT_gc;
75 }
```

```
76
77  static void dac_init(void)
78  {
79      // Konfigurer PD6 som DAC-utgang
80      PORTD.PIN6CTRL &= ~PORT_ISC_gm;
81      PORTD.PIN6CTRL |= PORT_ISC_INPUT_DISABLE_gc;
82      PORTD.PIN6CTRL &= ~PORT_PULLUPEN_bm;
83
84      VREF.DAC0REF = VREF_REFSEL_VDD_gc;
85      DAC0.CTRLA = DAC_ENABLE_bm | DAC_OUTEN_bm;
86  }
87
88  static void timer_init(void)
89  {
90      TCA0.SINGLE.PER = (uint16_t)TIMER_TOP;
91      TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;    // Aktiver overflow-avbrudd
92      TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1_gc | TCA_SINGLE_ENABLE_bm;
93  }
94
95  /* --- DSP-funksjoner --- */
96
97  static void buffer_push(ringbuffer *f, int16_t sample)
98  {
99      f->buffer[f->writeIndex] = sample;
100     f->writeIndex++;
101
102     if (f->writeIndex >= N) {
103         f->writeIndex = 0;
104     }
105 }
106
107 static uint16_t adc_get_sample(void)
108 {
109     ADC0.COMMAND = ADC_STCONV_bm;
110
111     while (!(ADC0.INTFLAGS & ADC_RESRDY_bm)) {
112         ;
113     }
114
115     ADC0.INTFLAGS = ADC_RESRDY_bm;
116     return ADC0.RES;
117 }
118
119 static int16_t compute_fir(ringbuffer *f)
120 {
121     int32_t accumulator = 0;
122
123     // Start lesing fra den nyeste punktproven
124     int16_t read_idx = (int16_t)f->writeIndex - 1;
125
126     for (uint16_t i = 0; i < N; i++) {
127
128         // H ndter folding av ringbufferen
129         if (read_idx < 0) {
130             read_idx = N - 1;
131         }
132
133         accumulator += (int32_t)b[i] * (int32_t)f->buffer[read_idx];
134         read_idx--;
135     }
136
137     // Q15: (b * x) gir Q15*Q0-ish her, men >>15 fungerer som skaleringspunkt
```

```
138     // siden b er Q15. x er sentrert 10-bit (ca +/-512) og behandles som heltall
        .
139     return (int16_t)(accumulator >> 15);
140 }
141 
142 static void dac_write(uint16_t value)
143 {
144     if (value > 1023) {
145         value = 1023;   // Tving verdien til 10-bit
146     }
147 
148     DAC0.DATA = value << 6;   // Venstrejuster for 16-bit register
149 }
150 
151 /* --- INIT av koeffisienter --- */
152 
153 static void coeff_init(void)
154 {
155     for (uint16_t i = 0; i < N; i++) {
156         b[i] = fir_b_q15_const[i];
157     }
158 }
159 
160 /* --- INTERRUPT SERVICE ROUTINE (ISR) --- */
161 
162 ISR(TCA0_OVF_vect)
163 {
164     // 1) Hent punktprove og sentrer signalet rundt 0
165     //    (10-bit: 0..1023 -> -512..511)
166     int16_t x = (int16_t)adc_get_sample() - 512;
167 
168     // 2) Legg ny punktprove i buffer og beregn filterutgang
169     buffer_push(&fir_buffer, x);
170     int16_t y = compute_fir(&fir_buffer);
171 
172     // 3) Rekonstruer signalet for DAC og clamp
173     int32_t out = (int32_t)y + 512;
174     if (out < 0) out = 0;
175     if (out > 1023) out = 1023;
176 
177     dac_write((uint16_t)out);
178 
179     TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
180 }
181 
182 /* --- HOVEDPROGRAM --- */
183 
184 int main(void)
185 {
186     // Nullstill ringbuffer state
187     fir_buffer.writeIndex = 0;
188     for (uint16_t i = 0; i < N; i++) {
189         fir_buffer.buffer[i] = 0;
190     }
191 
192     clock_init();
193     adc_init();
194     dac_init();
195     timer_init();
196     coeff_init();
197 
198     sei();   // Aktiver globale avbrudd
```

```
199
200     while (1) {
201     }
202 }
```

Kodeutdrag 2: Insert caption