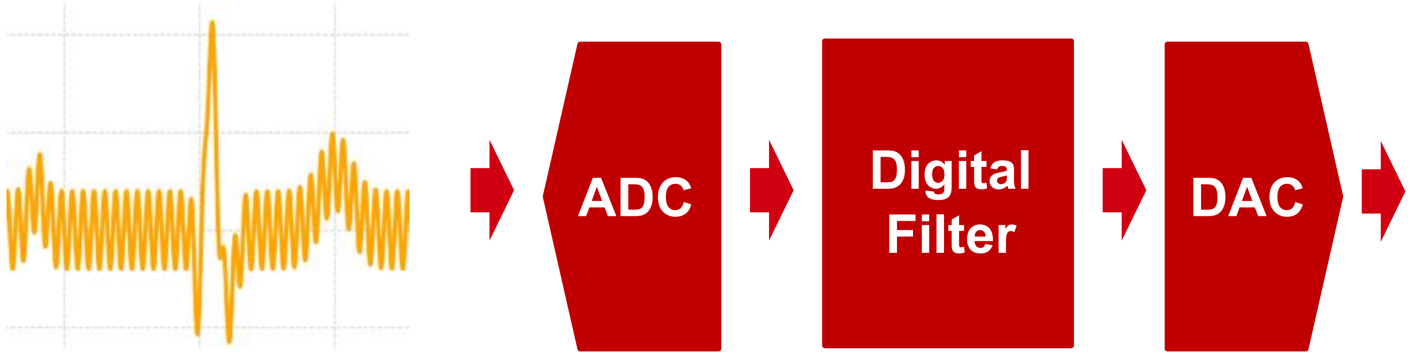
Embedded Implementation of Digital Filters

Prof. Kurian Polachan, IIIT-Bangalore

Filtering Basics



$$y[n] = \sum_{k=0}^{M} b[k] \cdot x[n-k]$$

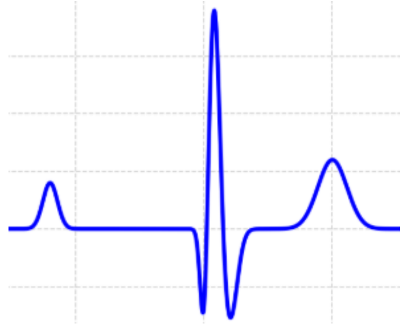
FIR (Finite Impulse Response)

- Output depends only on present and past inputs.
- Always stable

$$y[n] = \sum_{k=0}^{M} b[k] \cdot x[n-k] - \sum_{j=1}^{N} a[j] \cdot y[n-j]$$

IIR (Infinite Impulse Response)

- Output depends on inputs + past outputs (feedback)
- Uses feedback → can achieve sharp response with fewer coefficients.
- Potential stability issues



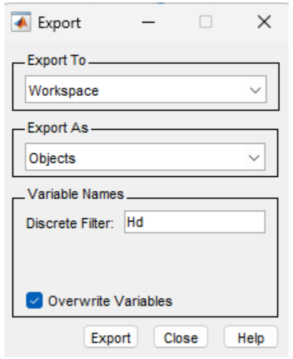
Filter Design

Question: Design an IIR low pass filter, with cutoff frequency of 100Hz,-40dB/decade, sampling frequency = 10KHz

Step1: Open MATLAB, >>filterDesigner

Step2: Enter the required fields. Next, generate filter design using "Design Filter". Next, verify the Filter Plots

Step3: File > Export



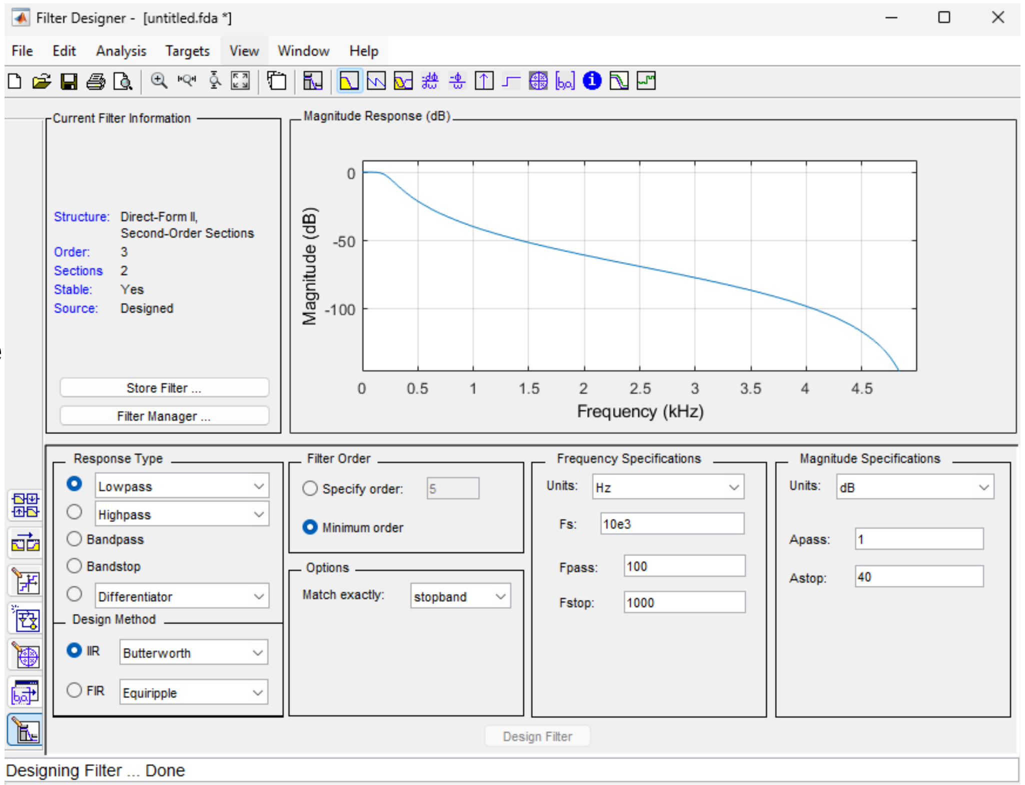
Export To: Workspace

Export As: Objects

Variable Names

Discrete Filter: Hd

☑ Overwrite Variables



Current Filter Information

Structure:	Direct-Form II, Second-Order Sections
Order:	3
Sections	2
Stable:	Yes
Source:	Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: [✓ IIR] Butterworth; FIR Equiripple

Filter Order: Specify order: 5; [✓ Minimum order]

Options: Match exactly: stopband

Frequency Specifications: Units: Hz; Fs: 10e3; Fpass: 100; Fstop: 1000

Magnitude Specifications: Units: dB; Apass: 1; Astop: 40

Designing Filter ... Done

Prof. Kurian Polachan, IIIT-Bangalore

Filter Design…

Step4: In MATLAB,

```
>> [b,a] = tf(Hd);
>> fprintf('int32 b[%d] = {%s};\n', length(b), sprintf('%g, ', b));
>> fprintf('int32 a[%d] = {%s};\n', length(a), sprintf('%g, ', a));
```

This prints a and b coefficients
e.g.,

```
int32 b[4] = {0.00029826, 0.000894781, 0.000894781, 0.00029826, };
int32 a[4] = {1, -2.72067, 2.479, -0.755947, };
```

Step5: Use the coefficients to craft the difference equation in C as,

```
y[n] = b[0]*x[n] + b[1]*x[n-1]  + b[2]*x[n-2]  + b[3]*x[n-3] +
                - a[1]*y[n-1] - a[2]*y[n-2] - a[3]*y[n-3];
```

Step6: Generate step response for future testing of the filter:

```
>> stepz(b,a,100)
```

Implementation

Loop:

Start ADC conversion.

Read the ADC sample → x(n).

Compute the filter output using the difference equation:

$$y(n) = b[0] \cdot x(n) + b[1] \cdot x(n-1) + \cdots - a[1] \cdot y(n-1) - a[2] \cdot y(n-2) - \cdots$$

Update the filter states for next iteration:

$x(n-1) \leftarrow x(n)$, $x(n-2) \leftarrow x(n-1)$, …

$y(n-1) \leftarrow y(n)$, $y(n-2) \leftarrow y(n-1)$, …

Write the output y(n) to the DAC.

Wait for the T_{wait} $\longrightarrow$

$T_{wait} = T_s - T_P$

T_P = time to read ADC + compute filter + update states + write DAC

T_S = sampling time = $1/F_S$

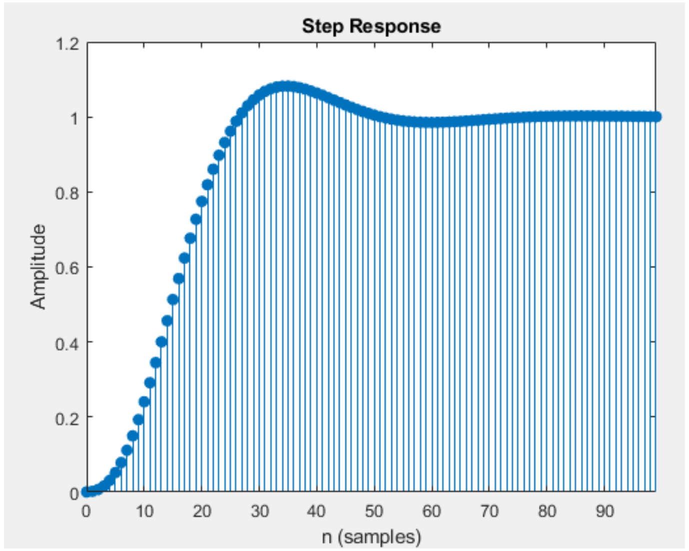
F_S = sampling frequency = $1/F_S$

Repeat the loop

Filter Testing

- Apply a step input (or feed a low frequency square wave to the input of ADC)
- Check the output of DAC
- Cross check this output with the step response simulated in MATLAB

```
>> stepz(b,a,100)
```


Step Response
Amplitude
n (samples)