Department of Electrical & Electronic Engineering

Imperial College of Science, Technology and Medicine

Real-time Implementation of IIR Filters

In this lab session, you will use MATLAB to design some IIR filters and then implement them in real-time on the DSP processor.

The transfer function of an IIR filter is given by $H(z) = \dfrac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}$.

This corresponds to a time-domain recurrence relation:

$$y(n) = b_0 x(x) + b_1 x(n-1) + \cdots + b_M x(n-M) - a_1 y(n-1) - \cdots - a_N y(n-N)$$

Single-Pole Filter

As an initial check, you should implement the single-pole low-pass filter $H(z) = \dfrac{b_0}{1 + a_1 z^{-1}}$. The impulse response of this filter is a negative exponential and for a time constant of τ, you need to set $a_1 = -\exp(-T/\tau)$ where $T = 1/8000$ is the sample period. Implement this filter with $\tau = 1\,\text{ms}$ and choose b_0 to give a DC gain of unity. By driving the input with a low-frequency squarewave, verify that the impulse response is correct. Verify also that the frequency response is as expected and has the correct corner frequency.

Bandpass Filter: Direct Form II

For the next example we want an elliptic bandpass filter with the following specifications:

Order	4th
Passband	100 Hz to 500 Hz
Passband ripple	0.5 dB
Stopband attenuation	20 dB

This can be designed using the MATLAB function `ellip()`. You should write a MATLAB function that calculates the coefficients for such a filter and writes them into a text file called `coef.txt` in a format suitable for inclusion in a C program. For example, for a third-order filter, the file might contain:

```
float a[] = {     1,    -1.76,    1.1829,    -0.2781,};
float b[] = {  0.0181,    0.0543,    0.0543,    0.0181,};
```

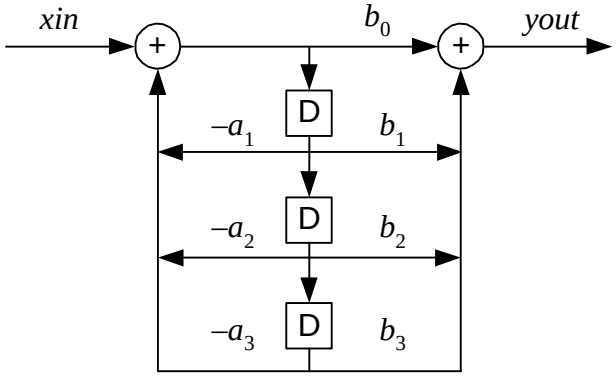
The comma following the final value in each line is optional but makes your MATLAB output routine easier. Note that the `ellip()` function requires you to specify frequencies normalized to the Nyquist frequency and to specify the order as half the desired value.

Write a C program to implement an IIR filter in Direct Form II as shown in the Figure below. Your program should work for any filter order, but you can assume that `a[]` and `b[]` are the same

length. You can determine the filter order (which is one less than the length of a[]) and allocate the required temporary storage with the following statements:

```
order = sizeof(a)/sizeof(a[0]) - 1;
w = (float *) calloc(order, sizeof(float));
```

You may require the w[] array to be of length order or order+1 according to the nature of your algorithm. Verify that the filter frequency response agrees with the MATLAB prediction.

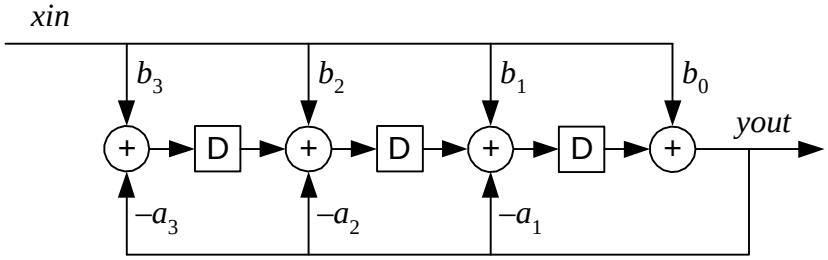


Use the profiler to determine how many instruction cycles per sample are needed for are needed for a filter of order *n* in the form $A + Bn$. You should include only the instructions between the calls to AD535_HWI_read() and AD535_HWI_write().

Now recompile your program but using the Compiler option –o2 to optimise the program for speed. See what difference this makes to the number of instruction cycles required.

Bandpass Filter: Direct Form II Transposed

Rewrite your program so that it implements a Direct Form II Transposed structure:

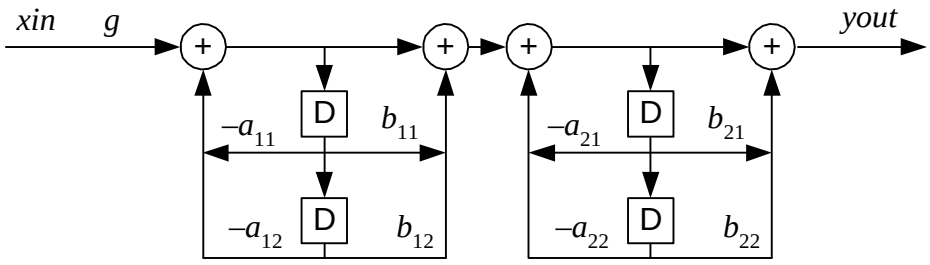


Verify that the filter response is unchanged. Determine the cycle count with and without optimisation.

Optional Bit for extra marks

Cascaded Biquad Implementation

Write the code to implement a filter as a cascade of second-order (biquad) filters. You can use any of the direct form implementations as the basic building block: I have shown Direct Form II below:



$$H(z) = g \times \frac{1 + b_{11}z^{-1} + b_{12}z^{-2}}{1 + a_{11}z^{-1} + a_{12}z^{-2}} \times \cdots \times \frac{1 + b_{K1}z^{-1} + b_{K2}z^{-2}}{1 + a_{K1}z^{-1} + a_{K2}z^{-2}}$$

Note that you only need a single gain coefficient, *g*, for the whole filter and do not need individual b_0 coefficients for each stage. You may find it easiest to have four separate coefficient arrays for the $a_{*1}, a_{*2}, b_{*1}, b_{*2}$ values. In MATLAB, you can convert a direct-form filter into cascaded second-order sections using the function `tf2sos()`. Verify that your bandpass filter works correctly.

The main reason for using a cascaded biquad implementation is that it is much less sensitive to coefficient errors. To see this, design the following enhanced version of your filter and try it out using both your direct form and your biquad versions:

Order	12th
Passband	100 Hz to 500 Hz
Passband ripple	0.5 dB
Stopband attenuation	40 dB

You will probably find that your direct form filter doesn't work: to see what is going wrong, use the watch window to examine the signal values. Even for a comparatively uncomplicated filter such as this, the coefficients need to be fantastically precise. MATLAB works to a precision of 52 significant bits but we can artificially reduce the precision to `n` bits with the following routine:

```
function y=bitsprec(x,n)
    [x,e]=log2(x);
    y=pow2(round(pow2(x,n)),e-n);
end
```

By evaluating `abs(roots(bitsprec(a,n)))` for various values of `n`, determine how many bits of precision are required to ensure that the 12th order filter is stable (a decimal digit corresponds to 3.3 bits). Do the same for the biquad filter sections.