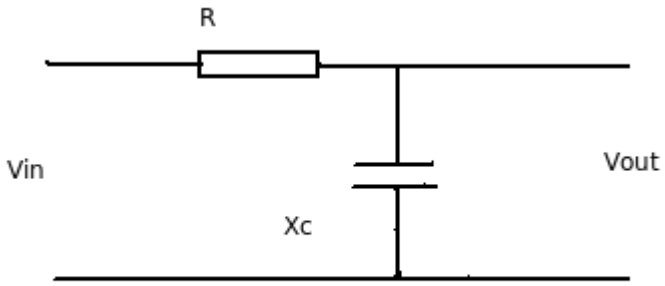
How can I do DSP on my microcontroller?

Introduction

Of course there are many techniques for designing DSP solutions but I am aiming in this paper to assist those who have a mainly analogue or ordinary digital background to come to believe that DSP implementations are possible and within the ability of radio amateurs to achieve. The main stumbling block is the horrendously mathematical underpinning of most DSP design techniques, so what is needed is a straightforward approach with as little theoretical modelling as possible. Also it is critical to move from what is relatively 'known' to radio amateurs toward what is relatively 'unknown', without great leaps of faith. In the first instance it would be good if a 'mechanism' or 'recipe' could be devised that would be 'do-able' for simple circuits. This is the aim of the article. Hopefully, individuals could become 'engaged' by this and spurred on to look into the background and theory. Thus there will little or no theory and little or no explanation here.

A simple analogue filter



This will be recognised as a 1st order low pass filter. So what are the steps to get an equivalent performance implemented as algorithm on a computer?

1. Derive the transfer function:

$$V_{out} = \frac{X_c}{R + X_c} \cdot V_{in} \quad \therefore \quad \frac{V_{out}}{V_{in}} = \frac{X_c}{R + X_c}$$

2. Express in R+jω form e.g.

$$\frac{V_{out}}{V_{in}} = \frac{\frac{1}{j\omega C}}{R + \frac{1}{j\omega C}}$$

3. Simplify. In this case multiply top and bottom by jωC

$$= \frac{1}{j\omega CR + 1}$$

$$= \frac{1}{sCR + 1}$$

4. Substitute s for jω
5. Substitute z-1/z+1 and forget CR for the moment

$$= \frac{1}{\frac{z-1}{z+1} + 1}$$

6. Simplify, multiply top and bottom by z+1 giving

$$= \frac{z+1}{z-1+z+1} = \frac{z+1}{2z}$$

7. Multiply top and bottom by z^{-1}

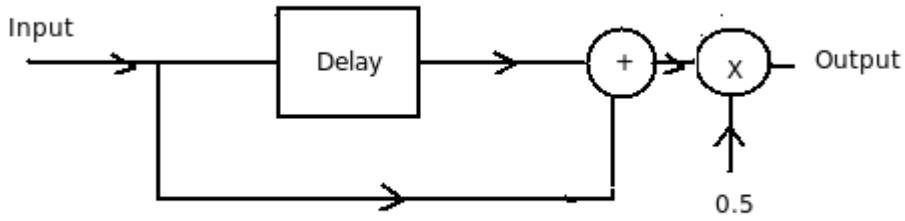
$$= \frac{1 + z^{-1}}{2}$$

This is the sampled data system transfer function and z^{-1} represents a delayed sample, so we can write the so-called recurrence formula as follows:

$y_n = 0.5(x_1 + \delta x_0)$

In this notation y_n is the current output sample, δ is one sample delay, x_1 is the current input sample and x_0 is the previous sample.

This gives an algorithmic structure like this:



All you have to do is program your PIC to perform the recurrence equation on samples from it's A/D and your output will be the input low pass filtered. Another name for the algorithm we have derived is a moving averager, which is indeed a low pass filter. You might now be wondering how to design it to have a certain performance e.g. Taking into account the sampling rate used and the desired corner frequency. Well, earlier we did forget CR and of course you will have to take account of this in a practical case, so plug back in the values for C and R and also because of the distortions inherent to the method you have, instead of using z-1/z+1 to substitute for s in step 5 above you have to use:

$$s = \frac{1}{K} \cdot \frac{z-1}{z+1}$$

where $K = \tan\left(\frac{\omega T}{2}\right)$ and $\omega T = 2\pi \frac{F_c}{F_s}$

where F_c = filter corner frequency
and F_s = sampling frequency

The effect of this just produces coefficients for the terms of the equation and changes the multipliers.

For those who want to know what the 'mechanism' actually does:

1. Decide on the analogue circuit you wish to emulate
2. Derive the Transfer Function
3. Map into a Laplace transfer function
4. Map into the z-plane by using the Bilinear Transform
5. Derive the sampled data structure and/or the recurrence formula
6. Program it!
7. Try it out

If you are designing for a specific parameters like the corner frequency and the sampling rate you MUST pre-warp (1/K) between 3 and 4 immediately above.

Let me know what you think of this effort via the 'Shoutbox' or private mail. Obviously I have to leave the programming in assembler/high level language to you since details of the processors you have and the development environments (compiler/assembler,linker,loader etc) could all be different. Clearly also the more complex the analogue circuit the more necessary it is to avoid the manual algebra and use a maths package like MathCAD (or cheaper alternative).

I have the sense that the impossible is being attempted here, but I think it's worth a try!

Good luck