C++ Implementation

Dividing by 2

To get an efficient implementation, we can minimize the number of multiplications needed by rewriting the difference equation:

$$y[n] = \alpha x[n] + (1 - \alpha)y[n - 1]$$

 $y[n] = y[n - 1] + \alpha(x[n] - y[n - 1])$

We now have to calculate two sums and one multiplication by a number between 0 and 1. Floating point numbers are relatively inefficient on integrated systems, so we'll replace the multiplication by a division by $1/\alpha$, which is a positive integer.

Going even further, we can pick value for α in such a way that $1/\alpha = 2^n, n \in \mathbb{N}$. Division is pretty expensive, but by using powers of 2, it can be replaced by a very fast right bitshift:

$$lpha x = rac{x}{2^n} = x \gg n$$

Fixed-point arithmetic

As mentioned before, it's not a great idea to use floating point numbers. An alternative is to use integers, and interpret them as fixed-

For example, you could decide to place the radix point between bits 3 and 4. In that case, you would interpret the number 0b00001001 (9 in decimal) as 0b0000.1001 = 1/2 + 1/16 = 0.5625.

To convert a normal integer to the fixed-point format, you can just bitshift it to the left.

To convert a fixed-point value back to an integer, one half is added to the number, in order to round it, and finally, it is bitshifted to the right

For example, let's use 3 fractional places to calculate 25 / 40.

$$\begin{array}{l} 25 = 0b00011001 \rightarrow 25 \ll 3 = 0b11001.000 = 200 \\ 200/40 = 5 = 0b00000101 = 0.625 \\ \rightarrow 0b00000.101 = 0.5 + 0.125 = 0.625 = 25/40 \end{array}$$

Rounding the result:

$$\begin{array}{l} 0.625 + 0.5 = 0b00000.101 + 0b00000.100 = 0b00001.001 \\ \rightarrow 0b00001001 \gg 3 = 0b00000001 = 1 \end{array}$$

TODO