8-bit microcontroller implements digital lowpass filter

Abel Raynus, Armatron International, Malden, MA

Filtering occurs frequently in the analog world. Unfortunately, in the digital world, engineers apply it mainly to the DSPs (digital-signal processors) and not to the small 8-bit microcontrollers that designers commonly use. This situation occurs because the math for the filter design is more complicated than most engineers are willing to deal with. Moreover, digital filtering requires calculations on integers instead of on floating-point numbers. This scenario causes two problems. First, the rounding-off error from the limited number of bits can degrade the filter response or even make it unstable. Second, you must handle the fractional values with integer math.

Several ways exist to solve these issues. For example, you can use operations with 16-, 32-, and 64-bit numbers, or you can scale for better accuracy. These and other methods usually require more memory, and, as a result, the program often does not fit into a small microcontroller. A literature search shows that published digital-filter firmware is written in C. Programs in C need more memory than those written in assembler. This situation often makes them unacceptable for small microcontrollers with limited memory resources.

Listing 1, available at the Web version of this Design Idea at www.edn.com/080124di1, shows a simple engineering method to design single-pole, lowpass-digital-filter firmware for 8-bit microcontrollers. The low-end Freescale (www.freescale.com) MC68HC908QT2 is the target of the assembler program, but you can apply this Design Idea to any type of microcontroller because it uses only standard assembler instructions.

Leaving aside the sophisticated design methods based on Z transformation with its extensive math, this idea uses another approach based on a recursive equation. You calculate each output-signal sample as the sum of the input signal and the previous output signal with corresponding coefficients. A recursive equation defines a single-pole lowpass filter as: $Y[n]=X[n]\times a0+Y[n-1]\times b1$, where X[n] and Y[n] are input and output values of sample [n], Y[n-1] is an output value of the previous sample [n−1], and a0 and b1 are weight coefficients that decrement δ controls. The coefficients have the value of $0<\delta<1$, $a0=1-\delta$, and $b1=\delta$. Physically, δ is the amount of decay between adjacent output samples when the input signal drops from a high level to a low level. You can directly specify the value of δ or find it from the desired time constant of the filter, d, which is the number of samples it takes the output to rise to 63.2% of the steady-state level for a lowpass filter. A fixed relationship exists between d and δ: $\delta=e^{-1/d}$, where e is the base of natural logarithms. The preceding **equations** yield $Y[n]=Y[n-1]+(1-\delta)\times(X[n]-Y[n-1])$.

NUMERICALLY PERFORMING THE FILTERING FUNCTION PROVIDES THE BENEFIT OF CONSISTENCY BECAUSE COMPONENT TOLERANCES, TEMPERATURE DRIFT, AND AGING DO NOT AFFECT THE FILTER'S ALGORITHM.

Instead of multiplying a decimal-point number, $1-\delta$, it is more convenient for assembler programming to divide by the reciprocal integer, $F=1/(1-\delta)$: $Y[n]=Y[n-1]+(X[n]-Y[n-1])/F$. Thus, you can determine the digital filter's parameters using the following steps:

1. Choose the parameter F. For assembler, it is convenient to perform division as right shifts. For right shifts, the value of F should be 2S, where S is the number of shifts. Let F equal 8, which you reach after three right shifts.

2. Calculate the decrement: $\delta=1-1/F=1-1/8=0.875$.

3. Calculate the time constant as $d=-1/\ln\delta=-1/\ln 0.875=7.49$ samples.

The **equation** $Y[n]=Y[n-1]+(X[n]-Y[n-1])/F$ determines the design of the microcontroller's algorithm for the filter. The algorithm needs three registers: input for X[n], output for Y[n], and an increment register to keep the $(X[n]-Y[n-1])/F$ term. The size of these registers depends on the inputs. In this application, the signals from the built-in 8-bit ADC range from 00 to $FF and must go through the lowpass filter. So, the input and the output registers are 1 byte in size. To increase the accuracy of division, add half the divisor to the dividend. This action increases the increment register to 2 bytes.

Numerically performing the filtering function provides the benefit of consistency because component tolerances, temperature drift, and aging do not affect the filter's algorithm. The implementation of the digital filter in the microcontroller gives the additional benefit of flexibility to adjust the filter's parameters, because this flexibility depends only on the firmware. **EDN**