Convolution Using Fixed-Point Arithmetic

1 Introduction

Digital Signal Processors (DSPs) can be classified as fixed- or floating-point based on the arithmetic they use. Floating-point DSPs are easier to program but are costlier. Hence commercial applications are dominated by fixed-point DSPs. But care has to be taken to deal with issues related to dynamic range and precision. This experiment aims to introduce the basics of fixed-point arithmetic, understanding which is essential for programming any fixed-point DSP.

Let M+N bits be used to represent a number. The notation M.N means that there are M bits to the left of the radix point and N bits to the right of it. For 16-bit numbers, the format can vary from 1.15 to 16.0. To belabour the obvious, all numbers in the 16.0 format are integers, whereas the remaining include fractions.

The product of two 16-bit numbers can be as wide as 32 bits. The format of the product of an M.N number and a P.Q number is (M+P).(N+Q). If the numbers are represented in twos-complement, the product has two sign bits, one of which is redundant, and is eliminated by a single left shift. As an example, if the two numbers are in 1.15 format, the product's format will be 2.30 and a left shift by one bit will result in a 1.31 format.

Example: Consider the product of -0.125 and 0.375, each expressed in 1.3 format. The multiplication is done using sign-magnitude form.

$$\begin{aligned} -(0.125 \cdot 0.375) &= -(0001 \cdot 0011) \\ &= -(0000\ 0011) \\ \text{left shift by one} &\rightarrow -(0000\ 0110) \\ &= 1111\ 1010 \end{aligned}$$

which is -0.048675 in 1.7 format.

This left shift is not required when the numbers are in integer format. Hence, prior to multiplication, the processor has to know whether or not the multiplication is to be carried out in integer mode. In the ADI 2100 DSP family this is achieved by a mode-setting instruction. Compared to fractional mode, integer mode is very easy to simulate in Scilab. The basic arithmetic operations for fixed point integer arithmetic are described in file `BASIC_OP.C` (see last section), and should be carefully studied.

The multiply-accumulate (MAC) operation is a fundamental one in DSP mathematics. The intermediate results of $\sum_k a_k \cdot b_k$ is typically stored using higher precision, greater than that required for a single product. A typical example is a 40-bit register for storing the intermediate sum of products when 16 bits are used to represent a_k and b_k. After the entire sum is computed the final result is rounded to 16 bits before storing it in memory.

Many basic DSP functions change the range of their input signals. The use of a fixed number of bits limits the range it can represent. If a signal exceeds this range overflow can occur. Overflow can be avoided by scaling the signal but the price paid is loss of precision and dynamic range; one has to take care to balance these two conflicting requirements. When an arithmetic operation results in a number that is too small to be represented *underflow* is said to occur (e.g., product of two tiny numbers). Once again, careful selection of the scale factor can minimize this problem.

Even after proper scaling the result of a running sum can exceed the largest number representable or become smaller than the most negative number. *Saturation arithmetic* reduces the error by replacing the unrepresentable result by the most positive or negative number (as the case may be). The hardware unit that implements saturation arithmetic is called *limiter* by some manufacturers.

2 Experiment Specification

Obtain a 33-tap equiripple FIR filter in Scilab using the command `h = eqfir(33,[0 .2;.2875 .5],[1 0],[0.025 1])`. Generate 300 samples of a signal that is the sum of two unit-amplitude sine waves with frequencies 1 kHz and 2.4 kHz. Assume a sampling frequency of 8 kHz. The main goal is to filter the signal using fixed-point arithmetic.

- Represent the filter coefficients and the signal using 16.0 format after scaling them appropriately and subsequent rounding.

- Plot the magnitude spectrum of the filter with quantized coefficients and compare it with that of the unquantized one.

- Perform convolution of the quantized signal and filter. Find the maximum and minimum values and determine the minimum number of bits needed to represent the result without overflow. Denote this as L, i.e., L bits will be used to represent the intermediate results. The final result of the convolution is to be stored in 16.0 format. Compare the results obtained from fixed- and floating-point arithmetic.

- Repeat the above by assuming that the intermediate results can be stored using the following number of bits: (a) $L-1$, (b) $L-2$, and (c) $L-3$. Once again compare these results with that obtained using floating point arithmetic.

3 Reference Material

- Review twos-complement representation of numbers (both integer and fractional), paying attention to the largest positive and smallest negative numbers that can be represented.
- `http://personal.bellsouth.net/lig/y/a/yatesc/papers.htm`. Read the first two papers: (i) Fixed-Point Arithmetic—An Introduction, and (ii) Practical Considerations in Fixed-Point FIR Implementations.
- `ftp://saarang.ee.iitm.ernet.in`. Numeric formats used in Analog Devices' ADSP-2100 family are discussed in `pub/dsp/ee4x1/fixed_point/appendixC.pdf`. C functions for fixed-point addition, multiplication, etc. can be found in `pub/speech/g729a/src/BASIC_OP.C`.
- `http://www.acm.org/pubs/citations/journals/surveys/1991-23-1/p5-goldberg/` is a classic reference. Although it talks about only floating point arithmetic many important issues related to finite precision implementation are discussed.