DSP Implementation Notes

ESE2014

fixed-point versus floating-point math

- when using software tools like Matlab or Octave, we are using floating-point number representations and seldom do we concern ourselves with issues like overflow or precision, since floating-point is a powerful representation:
    - it has a large range (due to the exponent portion), representing numbers spanning several orders of magnitude
    - it has excellent precision, especially on desktop computers, where many bytes can be used to represent the mantissa
- in embedded systems, while floating-point units are available on some processors, fixed-point arithmetic is still important:
    - a fixed-point implementation can be more portable (since it works on simpler MCUs without floating point units)
    - it can be faster or more energy efficient to implement arithmetic in fixed point, even if a floating-point unit is available
    - because of limited range and precision, we must be careful in our use of fixed-point techniques

fixed-point math

- in fixed-point arithmetic, we are actually using integers in calculations, keeping in mind an explicit scaling factor:

$$x = x^{\mathrm{norm}} \times 2^{M}$$

a normalized quantity, like the variables in Matlab, on some range like [-1,1]

the scaling factor, where *M* represents the number of bits used in the fractional part of the fixed point number, *x*

- note that if x^{norm} = 1.0, then x , in binary, looks like:

0000	0000	0000	0001	.	0000	0000	0000	0000
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16		15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

32-bit word

implicit “binary point”

fixed-point examples

- M = 16, x^{norm} = 0.536, y^{norm} = 0.251

- in binary form, x^{norm} is: 0000 0000 0000 0000. 1000 1001 0011 0111
- we get:
  - x = 35127
  - y = 16450
- addition/subtraction is straightforward:
  - x+y = 51577 => a "normalized" value of 51577/2^16 = 0.787 (correct)
- multiplication is more tricky, because you must keep in mind the scaling factors:
  - x * y = 577 839 150 => a 30-bit quantity (running the risk of overflow!)
    - this is a normalized value of 577 839 150/2^32

floating-point examples (cont'd)

- division can be awful: while we don't need to worry about the scaling factors (they cancel out), we can get substantial rounding errors:
    - x/y = 2.1354, which gets "integerized" to 2 (incorrect!) !
  - to deal with divisional rounding, we can use a larger range for our dividend, x, for example, [-10,10] instead of [-1,1], however, with larger numbers, we push the risk of overflow on multiplications
  - you have to decide what's more important for you: avoiding overflow in multiplications, or rounding errors from divisions
- fortunately, in most filtering applications, we are dealing primarily with addition (of signals) and multiplications (signals with filter coefficients or amplifier gains).
- if it all possible, it is best to have power-of-2 divisors (y) with integers, as this results in a simple bitwise-shift operation, which can be implemented with great speed on any MCU

floating-point examples (cont'd)

- in the previous division example, y = 0.251 which approximately 0.250, or 2^-2; therefore x/y is approximately x*2^2, or simply shift x twice to the left. This results in:
    - x << 2 = 0000 0000 0000 0010 . 0010 0100 1101 1100

      = 140508

      = 2.1440

  which is closer to 2.1354 than 2 is (error of 0.0086 versus 0.1354).

- a cool trick if you can make it happen!

scaling filter coefficients

- whether fixed or floating point, we should strive to have the signals in our filter realizations vary roughly in the same range; for most purposes, we can think of our signals as ideally “swinging” in the [-1, 1] range
- by doing so:
  - we can more easily implement analog versions of our filters, since we want signals to vary within (and fully utilize) the supply limits in order to maintain the highest signal-to-noise ratios
  - we can reduce the possibility of underflow or overflow errors
- given a state-space realization of a filter, this can often mean “balancing the coefficients, so that they aren’t too many orders of magnitude apart