$\sum_{n=-\infty}^{+\infty}$ ***Digital Sound Labs***

Digital Audio Signal Processing

Fixed-Point Arithmetic: An Introduction

Randy Yates

March 3, 2001 11:52

Typeset using PICTEX

Table of Contents

1 Introduction

This document presents definitions of signed and unsigned fixed-point binary number representations and develops basic rules and guidelines for the manipulation of these number representations using the common arithmetic and logical operations found in fixed-point DSPs and hardware components.

While there is nothing particularly difficult about this subject, I found little documentation either in hardcopy or on the web. What documentation I did find was disjointed, never putting together *all* of the aspects of fixed-point arithmetic that I think are important. I therefore decided to develop this material and to place it on the web not only for my own reference but for the benefit of others who, like myself, find themselves needing a complete understanding of the issues in implementing fixed-point algorithms on platforms utilizing integer arithmetic.

During the writing of this paper, I was developing assembly language code for the Texas Instruments TMS320C50 Digital Signal Processor, thus my approach to the subject is undoubtedly biased towards this processor in terms of the operation of the fundamental arithmetic operations. For example, the C50 performs adds and multiplies as if the numbers are simple signed two's complement integers. Contrast this against the Motorola 56k series which performs two's complement fractional arithmetic, with values always in the range $-1 \leq x < +1$.

It is my hope that this material is clear, accurate, and helpful. If you find any errors or inconsistencies, please email me at `yates@ieee.org`.

2 Fixed-Point Binary Representations

A collection of N (N a positive integer) binary digits (bits) has 2^N possible states. This can be seen from elementary counting theory, which tells us that there are two possibilities for the first bit, two possibilities for the next bit, and so on until the last bit, resulting in

$$\underbrace{2 \times 2 \times \ldots \times 2}_{N \text{times}} = 2^N$$

possibilities.

In the most general sense, we can allow these states to represent anything conceivable. In the case of an N-bit binary word, some examples are up to 2^N:

1. students at a university;
2. species of plants;
3. atomic elements;
4. integers;
5. voltage levels.

Drawing from set theory and elementary abstract algebra, one could view a representation as an onto mapping between the binary states and the elements in the representation set (in the case of unassigned binary states, we assume there is an "unassigned" element in the representation set to which all such states are mapped).

The salient point is that there is no meaning inherent in a binary word, although most people are tempted to think of them (at first glance, anyway) as positive integers (i.e., the *natural binary* representation, defined in the next section). However, **the meaning of an N-bit binary word depends entirely on its *interpretation,*** i.e., on the representation set and the mapping we choose to use.

In this section, we consider representations in which the representation set is a particular subset of the rational numbers. Recall that the rational numbers are the set of numbers expressible as a/b, where $a, b \in Z, b \neq 0$. (Z is the set of integers.) The subset to which we refer are those rationals for which $b = 2^n$. We also further constrain the representation sets to be those in which every element in the set has the same number of binary digits and in which every element in the set has the binary point at the same position, i.e., the binary point is fixed. Thus these representations are called "fixed-point."

The following sections explain four common binary representations: unsigned integers, unsigned fixed-point rationals, signed two's complement integers, and signed two's complement fixed-point rationals. We view the integer representations as special cases of the fixed-point rational representations, therefore we begin by defining the fixed-point rational representations and then subsequently show how these can simplify to the integer representations. We begin with the unsigned representations since they require nothing more than basic algebra. Section 2.2 defines the notion of a "two's complement" so that we may proceed well-grounded to the discussion of signed two's complement rationals in section 2.3.

2.1 Unsigned Fixed-Point Rationals

An N-bit binary word, when interpreted as an unsigned fixed-point rational, can take on values from a subset P of the non-negative rationals given by

$$P = \{p/2^b \mid 0 \leq p \leq 2^N - 1, \, p \in \mathcal{Z}\}.$$

Note that P contains 2^N elements. We denote such a representation $U(a, b)$, where $a = N - b$.

In the $U(a, b)$ representation, the nth bit, counting from right to left and beginning at 0, has a weight of $2^n/2^b = 2^{n-b}$. Note that when $n = b$ the weight is exactly 1. Similar to normal everyday base-10 decimal notation, the binary point is between this bit and the bit to the right. This is sometimes referred to as the **implied binary point.** A $U(a, b)$ representation has a integer bits and b fractional bits.

The value of a particular N-bit binary number x in a $U(a, b)$ representation is given by the expression

$$x = (1/2^b) \sum_{n=0}^{N-1} 2^n x_n$$

where x_n represents bit n of x. The range of a $U(a, b)$ representation is from 0 to $(2^N - 1)/2^b = 2^a - 2^{-b}$.

For example, the 8-bit unsigned fixed-point rational representation $U(6, 2)$ has the form

$$b_5 b_4 b_3 b_2 b_1 b_0 . b_{-1} b_{-2},$$

where bit b_k has a weight of 2^k. Note that since $b = 2$ the binary point is to the right of the second bit from the right (counting from zero), and thus the number has six integer bits and two fractional bits. This representation has a range of from 0 to $2^6 - 2^{-2} = 64 - 1/4 = 63\,3/4$.

The unsigned integer representation can be viewed as a special case of the unsigned fixed-point rational representation where $b = 0$. Specifically, an N-bit unsigned integer is identical to a $U(N, 0)$ unsigned fixed-point rational. Thus the range of an N-bit unsigned integer is

$$0 \leq U(N, 0) \leq 2^N - 1.$$

and it has N integer bits and 0 fractional bits. The unsigned integer representation is sometimes referred to as "natural binary."

Examples:

1. $U(6,2)$. This number has $6+2=8$ bits and the range is from 0 to $2^6 - 1/2^2 = 63.75$. The value 8Ah (1000,1010b) is

$$(1/2^2)(2^1 + 2^3 + 2^7) = 34.5.$$

2. $U(-2,18)$. This number has $-2+18 = 16$ bits and the range is from 0 to $2^{-2} - 1/2^{18} = 0.2499961853027$. The value 04BCh (0000,0100,1011,1100b) is

$$(1/2^{18})(2^2 + 2^3 + 2^4 + 2^5 + 2^7 + 2^{10}) = 1212/2^{18} = 0.004623413085938.$$

3. $U(16,0)$. This number has $16+0=16$ bits and the range is from 0 to $2^{16} - 1 = 65,535$. The value 04BCh (0000,0100,1011,1100b) is

$$(1/2^0)(2^2 + 2^3 + 2^4 + 2^5 + 2^7 + 2^{10}) = 1212/2^0 = 1212.$$

2.2 The Operations of One's Complement and Two's Complement

Consider an N-bit binary word x interpreted as if in the N-bit natural binary representation (i.e., $U(N,0)$). The *one's complement* of x is defined to be an operation that inverts every bit of the original value x. This can be performed arithmetically in the $U(N,0)$ representation by subtracting x from $2^N - 1$. That is, if we denote the one's complement of x as $\tilde{x}$, then

$$\tilde{x} = 2^N - 1 - x.$$

The two's complement of x, denoted $\hat{x}$, is determined by taking the one's complement of x and then adding one to it:

$$\begin{aligned}\hat{x} &= \tilde{x} + 1 \\ &= 2^N - x.\end{aligned}$$

Examples:

1. The one's complement of the U(8,0) number 03h (0000,0011b) is FCh (1111,1100b).

2. The two's complement of the U(8,0) number 03h (0000,0011b) is FDh (1111,1101b).

2.3 Signed Two's Complement Fixed-Point Rationals

An N-bit binary word, when interpreted as a signed two's complement fixed-point rational, can take on values from a subset P of the rationals given by

$$P = \{p/2^b \mid -2^{N-1} \leq p \leq 2^{N-1} - 1,\, p \in \mathcal{Z}\}.$$

Note that P contains 2^N elements. We denote such a representation $A(a,b)$, where $a = N - b - 1$.

The value of a specific N-bit binary number x in an A(a,b) representation is given by the expression

$$x = (1/2^b)\left[-2^{N-1}x_{N-1} + \sum_{0}^{N-2} 2^n x_n\right],$$

where x_n represents bit n of x. The range of an A(a,b) representation is

$$-2^{N-1-b} \leq x \leq +2^{N-1-b} - 1/2^b.$$

Note that the number of bits in the magnitude term of the sum above (the summation, that is) has one less bit than the equivalent prior unsigned fixed-point rational representation. Further note that these bits are the $N-1$ least significant bits. It is for these reasons that the most-significant bit in a signed two's complement number is usually referred to as the *sign bit*.

Example:

A(13,2). This number has 13+2+1=16 bits and the range is from $-2^{13} = -8192$ to $+2^{13} - 1/4 = 8191.75$.

3 Fundamental Rules of Fixed-Point Arithmetic

The following are practical rules of fixed-point arithmetic. For these rules we note that when a scaling can be either signed ($A(a,b)$) or unsigned ($U(a,b)$), we use the notation $X(a,b)$.

3.1 Unsigned Wordlength

The number of bits required to represent $U(a,b)$ is $a+b$.

3.2 Signed Wordlength

The number of bits required to represent $A(a,b)$ is $a+b+1$.

3.3 Unsigned Range

The range of $U(a,b)$ is $0 \leq x \leq 2^a - 2^{-b}$.

3.4 Signed Range

The range of $A(a,b)$ is $-2^a \leq \alpha \leq 2^a - 2^{-b}$.

3.5 Addition Operands

Two binary numbers must be scaled the same in order to be added. That is, $X(c,d) + Y(e,f)$ is only valid if $X = Y$ (either both A or both U) and $c = e$ and $d = f$.

3.6 Addition Result

The scale of the sum of two binary numbers scaled $X(e, f)$ is $X(e+1, f)$, i.e., the sum of two M-bit numbers requires $M+1$ bits.

3.7 Unsigned Multiplication

$U(a_1, b_1) \times U(a_2, b_2) = U(a_1 + a_2, b_1 + b_2)$.

3.8 Signed Multiplication

$A(a_1, b_1) \times A(a_2, b_2) = A(a_1 + a_2 + 1, b_1 + b_2)$.

3.9 Unsigned Division

$U(a_1, b_1)/U(a_2, b_2) = U(a_1 + b_2, \lceil \log_2(2^{a_2+b_1} - 2^{b_1-b_2}) \rceil)$.

3.10 Signed Division

$A(a_1, b_1)/A(a_2, b_2) = A(a_1 + b_2 + 1, a_2 + b_1)$.

3.11 Wordlength Reduction

Define the operation $HIn(X(a, b))$ to be the extraction of the n most-significant bits of $X(a, b)$. Similarly, define the operation $LOn(X(a, b))$ to be the extraction of the n least-significant bits of $X(a, b)$. For signed values,

$$\begin{aligned} HIn(A(a, b)) &= A(a, n - a - 1) \text{ and} \\ LOn(A(a, b)) &= A(n - b - 1, b). \end{aligned}$$

Similarly, for unsigned values,

$$\begin{aligned} HIn(U(a, b)) &= U(a, n - a) \text{ and} \\ LOn(U(a, b)) &= U(n - b, b). \end{aligned}$$

3.12 Shifting

A shift must be either "left" or "right" by a positive integer number of bits. The term "shift" can take on either of two meanings in this context and are defined below. Note that shift definition 1 requires an actual logical operation to be performed on the data, while the shift of definition 2 does not. The shift of definition 2 is a reinterpretation of the scaling.

Shift Definition 1 (shifting to modify scaling): To move the entire binary word, including the binary point, filling with zeros from the side being shifted from and truncating from the side being shifted to.

$$X(a,b) >> n = X(a+n, b-n)$$
$$X(a,b) << n = X(a-n, b+n).$$

Shift Definition 2 (shifting to multiply or divide by a power of two): To move the binary point the opposite of the shift direction, preserving the binary bit pattern.

$$X(a,b) >> n = X(a-n, b+n)$$
$$X(a,b) << n = X(a+n, b-n).$$

4 Dimensional Analysis in Fixed-Point Arithmetic

Consider a fixed-point variable x that is scaled $A(a_x, b_x)$. Denote the scaled value of the variable as lowercase x and the unscaled value as uppercase X so that

$$x = X/2^{b_x}.$$

Units, such as inches, seconds, furlongs/fortnight, etc., may be associated with a fixed-point variable by assigning a weight to the variable. Denote the scaled weight as w and the unscaled weight as W, so that the value *and dimension* of a quantity α_x that is to be represented by x can be expressed as

$$\begin{aligned}\alpha_x &= x \times w \\ &= X \times W.\end{aligned}$$

Since $x = X/2^{b_x}$,

$$x \times w = X/2^{b_x} \times w,$$

and since $x \times w = X \times W$,

$$X/2^{b_x} \times w = X \times W \Rightarrow w = 2^{b_x} W.$$

Example 1:

An inertial sensor provides a linear acceleration signal to a 16-bit signed two's complement A/D converter with a reference voltage of 2V peak. The analog sensor signal is related to acceleration in meters per second squared through the conversion $m/(s^2 - Volt)$, i.e., the actual acceleration $\alpha(t)$ in meters/second^2 can be determined from the sensor voltage $v(t)$ as

$$\alpha(t) = v(t)\,[\text{volts}] \times \left[\frac{\text{m}}{\text{s}^2\text{-Volt}}\right].$$

If we consider the incoming A/D samples to be scaled $A(16, -1)$, what is the corresponding scaled and unscaled weights?

Solution:

We know that -32768 corresponds to -2 volts. Using the equation

$$\begin{aligned}\alpha_x &= x \times w \\ &= X/2^{b_x} \times w\end{aligned}$$

we simply plug in the values to obtain

$$-2\,[\text{volts}] = -32768/2^{-1} \times w_v.$$

Solving for w_v provides the intermediate result

$$w_v = \left[\frac{\text{volt}}{32768}\right].$$

Let's check: An unscaled value of 32767 corresponds to a scaled value of $v = 32767/2^{-1} = 65534$, and thus the physical quantity α_v to which this corresponds is

$$\begin{aligned}
\alpha_v &= v \times w_v \\
&= 65534 \times \left[\frac{\text{volt}}{32768}\right] \\
&= 1.999939\,[\text{volts}]\,.
\end{aligned}$$

Now simply multiply w_v by the original analog conversion factor $\left[\frac{\text{m}}{\text{s}^2\text{-Volt}}\right]$ to obtain the acceleration weighting w_a directly:

$$\begin{aligned}
w_a &= w_v \times \left[\frac{\text{m}}{\text{s}^2\text{-Volt}}\right] \\
&= \left[\frac{\text{volt}}{32768}\right] \times \left[\frac{\text{m}}{\text{s}^2\text{-Volt}}\right] \\
&= \left[\frac{\text{m}}{32768\text{s}^2}\right].
\end{aligned}$$

The unscaled weight is then determined from the scaled weight and the scaling as

$$\begin{aligned}
W_a &= w_a/2^{b_a} \\
&= \left[\frac{\text{m}}{32768\text{s}^2}\right]/2^{-1} \\
&= \left[\frac{\text{m}}{16384\text{s}^2}\right].
\end{aligned}$$

Example 2:

Bias is an important error in inertial measurement systems. An average scaled value of 29 was measured from the inertial measurement system in example 1 with the system at rest. What is the bias β?

Solution:

$$\begin{aligned}
\beta &= x \times w \\
&= 29 \times \left[\frac{\text{m}}{32768\text{s}^2}\right] \\
&= 0.88501 \times 10^{-3}\left[\frac{\text{m}}{\text{s}^2}\right].
\end{aligned}$$

---

5 Concepts of Finite Precision Math

5.1 Precision

Precision is the maximum number of non-zero bits representable. For example, an A(13,2) number has a precision of 16 bits. For fixed-point representations, precision is equal to the wordlength.

5.2 Resolution

Resolution is the smallest non-zero magnitude representable. For example, an A(13,2) has a resolution of $1/2^2 = 0.25$.

5.3 Range

Range is the difference between the most negative number representable and the most positive number representable,

$$X_R = X_{MAX+} - X_{MAX-}.$$

For example, an A(13,2) number has a range from -8192 to +8191.75, i.e., 16383.75.

5.4 Accuracy

Accuracy is the magnitude of the maximum difference between a real value and it's representation. For example, the accuracy of an A(13,2) number is 1/8. Note that accuracy and resolution are related as follows:

$$A(x) = R(x)/2,$$

where $A(x)$ is the accuracy of x and $R(x)$ is the resolution of x.

5.5 Dynamic Range

Dynamic range is the ratio of the maximum absolute value representable and the minimum positive (i.e., non-zero) absolute value representable. For a signed fixed-point rational representation $A(a, b)$, dynamic range is

$$2 \times 2^a/2^{-b} = 2^{a+b+1} = 2^N.$$

For an unsigned fixed-point rational representation $U(a, b)$, dynamic range is

$$(2^a - 2^{-b})/2^{-b} = 2^{a+b} - 1 = 2^N - 1.$$

For N of any significant size, the "-1" is negligible and therefore signed and unsigned representations of the same wordlength have practically the same dynamic range.

6 Fixed-Point Analysis—An Example

An algorithm is usually defined and developed using an algebraically complete number system such as the real or complex numbers. To be more precise, the operations of addition, subtraction, multiplication, and division are performed (for example) over the field $(\Re, +, \times)$, where subtraction is equivalent to adding the additive inverse and division is equivalent to multiplying by the multiplicative inverse.

As an example, consider the algorithm for calculating the average of the square of a digital signal $x(n)$ over the interval N (here, the signal is considered to be quantized in time but not in amplitude):

$$y(n) = \frac{1}{N} \sum_{k=0}^{N-1} x^2(n-k).$$

In this form, the algorithm implicitly assumes $x(n) \in \Re$, and the operations of addition and multiplication are performed over the field $(\Re, +, \times)$. In this case, the numerical representations have infinite precision.

This state of affairs is perfectly acceptable when working with pencil and paper or higher-level floating-point computing environments such as Matlab or MathCad. However, when the algorithm is to be implemented in fixed-point hardware or software, it must necessarily utilize a finite number of binary digits to represent $x(n)$, the intermediate products and sums, and the output $y(n)$.

Thus the basic task of converting such an algorithm into fixed-point arithmetic is that of determining the wordlength, accuracy, and range required for each of the arithmetic operations involved in the algorithm. In the terms of the fundamentals given in section 2, we need to determine a) whether the value should be signed ($A(a, b)$) or unsigned ($U(a, b)$), b) the value of N (the wordlength), and c) the values for a and b (the accuracy and range). Any two of wordlength, accuracy, and range determine the third. For example, given wordlength and accuracy, range is determined. In other words, we cannot independently specify all of wordlength, accuracy, and range.

Continuing with our example, assume the input $x(n)$ is scaled $A(15, 0)$, i.e., plain old 16-bit signed two's complement samples. The first operation to be performed is to compute the square. According to the rules of fixed-point arithmetic, $A(15, 0) \times A(15, 0) = A(31, 0)$. In other words, we require 32 bits for the result of the square in order to guarantee that we will avoid overflow and maintain precision. It is at this point that design tradeoffs and other information begin to affect how we implement our algorithm.

For example, in one possible scenario, we may know *a-priori* that the input data $x(n)$ do not span the full dynamic range of the $A(15, 0)$ representation, thus it may be possible to reduce the 32-bit requirement for the result and still guarantee that the square operation does not overflow.

Another possible scenario is that we do not require all of the precision in the result, and this also will reduce the required wordlength.

In yet a third scenario, we may look ahead to the summation to be performed and realize that if we don't scale back the result of each square we will overflow the sum that is to subsequently be performed (assuming we have a 32-bit accumulator). On the other hand, we may be using a fixed-point processor such as the TI TMS320C54x which has a 40-bit accumulator, thus we have 8 "guard bits" past the 32-bit result which may be used in the accumulations to prevent overflow for up to 256 (8=log2(256)) sums.

To complete our example, let's further assume that a) we keep all 32 bits of the result of the squaring operation, b) the averaging "time," N, does not exceed $2^4 = 16$ samples, c) we are using a fixed-point processor with an accumulator of $32 + 4 = 36$ bits or greater, and d) the output wordlength for $y(n)$ is 16 bits ($A(15, 0)$). The final decision that must be made is to determine which method we will use to form a 16-bit value from our 36-bit sum. It is clear that we should take the 16 bits from bits 20 to 35 of the accumulator (where bit 0 is the LSB) in order to avoid overflowing the output, but shall we truncate or round? Shall we utilize some type of dithering or noise-shaping? These are all questions that relate to the process of *quantization* since we are quantizing a 36-bit word to a 16-bit word. The theory of quantization and the tradeoffs to be made are outside the scope of this topic.

7 Acknowledgments

I wish to thank my colleague John Storbeck, a fellow DSP programmer and full-time employee of GEC-Marconi Hazeltine (Wayne, NJ), for his guidance during my early encounters with the C50 and fixed-point programming, and for his stealy critiques of my errant thoughts on assorted programming solutions for the C50. I also wish to thank Dr. Donald Heckathorn for nurturing my knowledge of fixed-point arithmetic during the same time-frame. Finally, I wish to thank my friend and fellow DSP engineering colleague Robert Bristow-Johnson for his encouragement and guidance during my continuing journey through the world of fixed-point programming and digital signal processing.