Author	Date	Time	Rev	No.	Reference
Randy Yates	16–May–2024	19:39	PA11	n/a	fp.tex

Contents

List of Figures

List of Tables

List of Examples

1 Introduction

Fixed-point arithmetic is the art of implementing algorithms which typically require operations over the field of real $\mathbb{R}$ or complex $\mathbb{C}$ numbers using signed, two's complement integers. The Matlab help has a good explanation of fixed-point numbers [1], the important aspects of which are word length, scale factor, and integer type (signed or unsigned).

Converting an algorithm that was originally implemented using floating-point arithmetic to one that uses fixed-point arithmetic is justified when one or more of the following implementation goals are sought

1. High speed, i.e., the ability to run the algorithm at a high speed relative to the clock rate of the processor.
2. Low power, i.e., the ability to implement the algorithm with lower power. Typically low power implementations consist of speed-optimized algorithms which can be executed using a lower percentage of CPU utilization.
3. Higher performance. In some cases, fixed-point implementations can provide better performance, e.g., lower quantization or residual noise performance, than the equivalent floating-point implementation.

We present 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.

Finally, the reader may be interested in the author's related paper [2] on the application of fixed-point arithmetic to the implementation of FIR filters.

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 j/k, where $j, k \in \mathbb{Z}, k \neq 0$, and $\mathbb{Z}$ is the set of integers. The subset to which we refer are those rationals for which $k = 2^N$, N a non-negative integer.

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. Appendix A.3 defines the notion of a "two's complement" and two's complement binary numbers and may be useful to review prior to reading section 2.3.

2.1 Unsigned Fixed-Point Rationals

An N-bit binary word is defined as an *unsigned fixed-point rational* when it is mapped to the subset $P_b(N)$ of the non-negative rationals given by

$$P_b(N) = \{p/2^b \mid 0 \leq p \leq 2^N - 1,\, p \in \mathbb{Z}\},$$

where b is a non-negative integer.

Note the following:

1. $P_b(N)$ contains 2^N elements.
2. $P_0(N)$ is the set of non-negative integers $\{0, 1, \ldots, 2^N - 1\}$.
3. $P_0(N)$ is the first 2^N elements of the *natural numbers* [3, p. 9].
4. $P_0(N)$ is commonly referred to in digital hardware, processor, and software design as N-bit *unsigned numbers* [4, p. 12].

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:

$$\hat{x} = \tilde{x} + 1$$

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

Example 2.1 (one's complement)

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

Example 2.2 (two's complement)

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

2.3 Signed Fixed-Point Rationals

An N-bit binary word is defined as a *signed fixed-point rational* when it is mapped to the subset $R_b(N)$ of the positive and negative rationals given by

$$R_b(N) = \{p/2^b \mid -2^{N-1} \leq p \leq 2^{N-1} - 1, \, p \in \mathbb{Z}\},$$

where b is a non-negative integer.

Note the following:

1. $R_b(N)$ contains 2^N elements.
2. $R_b(N)$ contains both positive and negative elements.
3. $R_0(N)$ is the set of integers $\{-2^{N-1}, -2^{N-1}+1, \ldots, -1, 0, +1, \ldots, +2^{N-1}-2, +2^{N-1}-1\}$.
4. $R_0(N)$ is called the set of signed two's complement integers for bit-width N [5, p. 519]. See Appendix A.3 for a discussion of two's complement binary numbers.

3 Fixed-Point Binary Notations

3.1 Unsigned $U(a,b)$ Notation

We define the notation $U(a,b)$ as the set $P_b(N)$, where $N = a + b$ (see section 2.1).

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 left 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."

Example 3.1 (8-bit unsigned fixed-point $U(6, 2)$)

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

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

Example 3.2 (16-bit unsigned fixed-point $U(-2, 18)$)

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

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

Example 3.3 (16-bit unsigned fixed-point $U(16, 0)$)

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

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

3.2 Signed $A(a, b)$ Notation

We define the notation $A(a, b)$ as the set $R_b(N)$, where $N = a + b + 1$ (see section 2.3).

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

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

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 3.4 (16-bit signed fixed-point $A(13, 2)$)

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

3.3 Q Notation

This notation has been defined by Texas Instruments and ARM (and possibly others) previously. See [6] and [7, sec.4.7.9], respectively.

Both signed and unsigned fixed-point numbers can be designated by the Q notation. The type of the representation, signed or unsigned, must be specified or understood from the context.

3.3.1 Unsigned Notation

The set of unsigned fixed-point rationals $P_b(N)$ can be designated by the popular Q notation as Qb or as $Qa.b$. When using the Qb notation, the word width N cannot be determined from the notation and must be specified or understood from the context. The $Qa.b$ notation is equivalent to the $U(a,b)$ notation, and the word width $N = a + b$. In either case, there are b fractional bits in the word. If $Qa.b$ is specified, or it is known that there are N total bits in the word, then there are $a = N - b$ integer bits.

Example 3.5 ($Q15$, unsigned fixed-point rational with 15 fractional bits)

This number has an unknown total number of total and integer bits and 15 fractional bits. If it is determined from context to have N total bits, it is comprised of 15 fractional bits and $N - 15$ integer bits, equivalent to a $U(N - 15, 15)$. If $N = 16$, for example, it is equivalent to the notations $Q1.15$ and $U(1, 15)$.

Example 3.6 ($Q17.15$, unsigned fixed-point rational with $17 + 15 = 32$ total bits)

This is similar to the previous example except that the total number of bits is 32. It is equivalent to a $U(17, 15)$.

3.3.2 Signed Notation

The set of signed fixed-point rationals $R_b(N)$ can be designated by the popular Q notation as Qb or as $Qa.b$. When using the Qb notation, the word width N cannot be determined from the notation and must be specified or understood from the context. The $Qa.b$ notation is equivalent to the $A(a,b)$ notation, and the word width $N = a + b + 1$. In either case, there are b fractional bits in the word. If $Qa.b$ is specified, or it is known that there are N total bits in the word, then there are $a = N - b - 1$ integer bits. **As in the $A(a,b)$ notation, the sign bit is not considered as part of the integer (a) part.**

Example 3.7 ($Q15$, unsigned fixed-point rational with 15 fractional bits)

This number has an unknown total number of total and integer bits and 15 fractional bits. If it is determined from context to have N total bits, it is comprised of 15 fractional bits and $N - 15$ integer bits, equivalent to a $U(N - 15, 15)$. If $N = 16$, for example, it is equivalent to the notations $Q1.15$ and $U(1, 15)$.

Note the following:

1. The Q notation is ambiguous in that it can represent either signed or unsigned fixed-point numbers.
2. There is an alternate Q notation for signed fixed-point rationals $Qa.b$ in which $a = N - b$.

Example 3.8 ($Q0.15$, signed fixed-point rational with 15 fractional bits)

This number has $0 + 15 + 1 = 16$ total number of bits and 15 fractional bits. It is comprised of 15 fractional bits and one sign bit.

3.3.3 Exceptions to Standard Q Notation

Some special exceptions exist in the literature to the notation defined in sections 3.3.1 and 3.3.2:

1. ARM [7] defines $Q15$ to be shorthand for an unsigned $Q1.15$ and $Q31$ to be shorthand for a signed $Q1.31$.
2. TI [6] defines $Q15$ to be equivalent to the signed $A(1, 15)$ notation.

These exceptions are avoided in this paper in order to provide the most consistent and general notation.

4 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 $X(a,b)$.

4.1 Unsigned Wordlength

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

4.2 Signed Wordlength

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

4.3 Unsigned Range

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

4.4 Signed Range

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

4.5 Signed Addition and Subtraction

Two signed binary numbers must be scaled the same in order to be added or subtracted. That is, $A(c, d) + A(e, f)$ or $A(c, d) - A(e, f)$ is only valid if $c = e$ and $d = f$.

If both operands of are allowed occupy the full range of the underlying integers, then the sum or difference of two N-bit signed binary numbers scaled $A(a, b)$ requires $N + 1$ bits in order to maintain precision and avoid overflow. The result will be scaled $A(a + 1, b)$

In general, the sum or difference of M N-bit signed binary numbers requires $N + \lceil \log_2 M \rceil$ bits to maintain precision and avoid overflow.

This is referred to as *bit growth* and is one of the reasons ALUs in processors and digital hardware provide wide accumulators.

4.6 Unsigned Addition and Subtraction

TBD, special cases when subtracting two unsigned values.

4.7 Unsigned Multiplication

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

4.8 Signed Multiplication

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

4.9 Unsigned Division

Let $U(a_3, b_3) = \frac{U(a_1,b_1)}{U(a_2,b_2)}$ and consider the largest possible result:

$$\begin{aligned} \text{largest result} &= \frac{\text{largest dividend}}{\text{smallest divisor}} \\ &= \frac{2^{a_1} - 2^{-b_1}}{2^{-b_2}}. \\ &= 2^{a_1+b_2} - 2^{b_2-b_1}. \end{aligned} \tag{2}$$

Thus we require

$$2^{a_3} - 2^{-b_3} \geq 2^{a_1+b_2} - 2^{b_2-b_1}. \tag{3}$$

It is natural to let $a_3 = a_1 + b_2$, in which case the inequalities below result:

$$\begin{aligned}
2^{a_3} - 2^{-b_3} &\geq 2^{a_3} - 2^{b_2 - b_1} \\
-2^{-b_3} &\geq -2^{b_2 - b_1} \\
2^{-b_3} &\leq 2^{b_2 - b_1} \\
-b_3 &\leq b_2 - b_1 \\
b_3 &\geq b_1 - b_2. \qquad (4)
\end{aligned}$$

Thus we have a constraint on b_3 due to b_1 and b_2.

Now consider the smallest possible result:

$$\begin{aligned}
\text{smallest result} &= \frac{\text{smallest dividend}}{\text{largest divisor}} \\
&= \frac{2^{-b_1}}{2^{a_2} - 2^{-b_2}}. \qquad (5)
\end{aligned}$$

This then requires b_3 to obey the following constraint:

$$\begin{aligned}
2^{-b_3} &\leq \frac{2^{-b_1}}{2^{a_2} - 2^{-b_2}} \\
b_3 &\geq b_1 + \log_2(2^{a_2} - 2^{-b_2})
\end{aligned} \qquad (6)$$

If we assume b_2 is positive, (6) is the more stringent of the two constraints (4) and (6) on b_3. We then express (6) in a slightly simpler form:

$$b_3 \geq \log_2(2^{a_2+b_1} - 2^{b_1-b_2}). \qquad (7)$$

The final result is then

$$U(a_1, b_1)/U(a_2, b_2) = U(a_1 + b_2, \lceil \log_2(2^{a_2+b_1} - 2^{b_1-b_2}) \rceil). \qquad (8)$$

4.10 Signed Division

Let $r = n/d$ where n is scaled $A(a_n, b_n)$ and d is scaled $A(a_d, b_d)$. What is the scaling of r $(A(a_r, b_r))$?

$$\begin{aligned}
|r_M| &= \frac{|n_M|}{|d_m|} \qquad (9) \\
&= \frac{2^{a_n}}{2^{-b_d}} \qquad (10) \\
&= 2^{a_n + b_d}. \qquad (11)
\end{aligned}$$

Since this maximum can be positive (when the numerator and denominator are both negative), $a_r = a_n + b_d + 1$.

Similarly,

$$\begin{aligned}
|r_m| &= \frac{|n_m|}{|d_M|} \qquad (12) \\
&= \frac{2^{-b_n}}{2^{a_d}} \qquad (13) \\
&= 2^{-(a_d + b_n)}. \qquad (14)
\end{aligned}$$

This implies $b_r = a_d + b_n$.

Thus

$$\frac{A(a_n, b_n)}{A(a_d, b_d)} = A(a_n + b_d + 1, a_d + b_n). \tag{15}$$

4.11 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,

$$\begin{aligned} HIn(A(a, b)) &= A(a, n - a - 1) \text{ and} \\ LOn(A(a, b)) &= A(n - b - 1, b). \end{aligned} \tag{16}$$

Similarly, for unsigned values,

$$\begin{aligned} HIn(U(a, b)) &= U(a, n - a) \text{ and} \\ LOn(U(a, b)) &= U(n - b, b). \end{aligned} \tag{17}$$

4.12 Shifting

We define two types of shift operations below, *literal* and *virtual*, and describe the scaling results of each.

Note that shifts are expressed in terms of right shifts by integer n. Shifting left is accomplished when n is negative.

4.12.1 Literal Shift

A *literal shift* occurs when the bit positions in a register move left or right. A literal shift can be performed for two possible reasons, to divide or multiply by a power of two, or to change the scaling.

In both cases note that this will possibly result in a loss of precision or overflow assuming the output register width is the same as the input register width.

4.12.1.1 Multiplying/Dividing By A Power of Two A literal shift that is done to multiply or divide by a power of two shifts the bit positions but keeps the output scaling the same as the input scaling.

Thus we have the following scaling:

$$X(a, b) >> n = X(a, b). \tag{18}$$

Example 4.1 (Fixed-Point Division)

Let's say X is a 16-bit signed two's complement integer that is scaled A(14,1), or Q1. Let's set that integer equal to 128: $X = +128 = 0x0080$, and thus its scaled value is

$$\begin{aligned} x &= X/2^1 && (19) \\ &= 128/2 && (20) \\ &= 64.0 && (21) \end{aligned}$$

Now we want to divide that by 4, so we shift it right by 2, $X = X >> 2$, so that the new value of X is 32. This shift didn't change the scaling since we are dividing by actually shifting, so it's still scaled Q1 after the shift. So now $X = 32$ and $x = X/2 = 16.0$. Since the original value of x was 64.0, we see that we have indeed divided that value by 4, which was the objective.

Note that this is probably a bad way to multiply or divide a fixed-point value since, if you're multiplying, you run the risk of overflowing, and if you're dividing, you run the risk of losing precision. It would be much better to perform the multiplication or division using the "virtual shift" method described in section 4.12.2.

4.12.1.2 Modifying Scaling A literal shift that is done to modify the scaling shifts the bit positions and makes the output scaling different than the input scaling. Thus we have the following scaling:

$$X(a,b) >> n = X(a+n, b-n). \tag{22}$$

Example 4.2 (Division By Modifying Scaling)

Again let's say X is a 16-bit signed two's complement integer that is scaled A(14,1), or Q1, and let's set that integer equal to 128: $X = +128 = 0x0080$, and thus its scaled value is

$$\begin{aligned} x &= X/2^1 && (23)\\ &= 128/2 && (24)\\ &= 64.0 && (25) \end{aligned}$$

Now let's say we want to change the scaling from Q1 to Q3 (or equivalently, from A(14,1) to A(12,3)). So we shift the integer left by two bits: $X = X << 2$. So our new integer value is 512, but we've now also changed our scaling as in equation 22 so that it's A(12,3) ($n = -2$ here since we're shifting left).

So in this case our final fixed-point value is still 512 / 8 = 64.0.

4.12.2 Virtual Shift

A *virtual shift* shifts the virtual binary point[1] without modifying the underlying integer value. It can be used as an alternate method of performing a multiplication or division by a power of two. However, unlike the literal shift case, the virtual shift method loses no precision and avoids overflow. This is because the bit positions don't actually move—the operation is simply a reinterpretation of the scaling.

$$X(a,b) >> n = X(a-n, b+n). \tag{26}$$

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,

$$X_R = X_{MAX+} - X_{MAX-}. \tag{27}$$

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

[1]The virtual binary point is called virtual since it doesn't actually exist anywhere except in the programmer's mind.

5.4 Accuracy

Accuracy is the magnitude of the maximum difference between a real value and it's representation.

The accuracy of both signed A(a,b) and unsigned U(a,b) representations is $1/2^{b+1}$ when rounding and $1/2^b$ when truncating.

For example, the accuracy of an A(13,2) number is $1/8$ when rounding and $1/4$ when truncating.

Note that accuracy and resolution are related as follows:

$$A(x) = R(x)/2, \tag{28}$$

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

Figure 1 is a visual illustration of accuracy versus precision. The origin is a two-dimensional value to be represented. The green dots show representations with high accuracy but low precision. The red dots show representations with low accuracy but high precision.

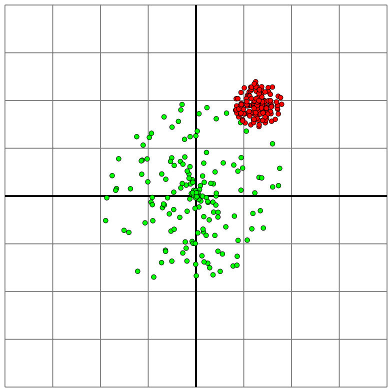

Figure 1: Accuracy vs. Precision

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

$$\times 2^a/2^{-b} = 2^{a+b} = 2^{N-1}. \tag{29}$$

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

$$(2^a - 2^{-b})/2^{-b} = 2^{a+b} - 1 = 2^N - 1. \tag{30}$$

For N of any significant size, the "-1" is negligible.

6 Dimensional Analysis in Fixed-Point Arithmetic

6.1 Weights and Units

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

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

Such a variable, for example a `uint32_t X;` in the C programming language, is inherently *unitless*.

Units, such as inches, seconds, furlongs/fortnight, etc., may be associated with a fixed-point variable by assigning a *weight* w_x to the variable:

$$w_x = \mu_x \cdot u_x \tag{31}$$

where

$$\mu_x = \text{unit scaling factor of } x, \mu_x \in \mathbb{R} \tag{32}$$
$$u_x = \text{units of } x\text{, e.g., } [\mathrm{m/s}]. \tag{33}$$

Units are defined in [8] and the tables of SI base and derived units shown there are nice examples. Note that in this document we enclose units with square brackets ([ ]).

We may define separate weights for the scaled value x and the unscaled value X:

$$w_x = \mu_x \cdot u_x \tag{34}$$
$$w_X = \mu_X \cdot u_X. \tag{35}$$

However, note that the units of the variable does not depend on whether it is the scaled or unscaled value. Thus

$$u_x = u_X. \tag{36}$$

The value *and units* of a physical quantity that is represented by x may be denoted α_x and can be expressed as

$$\alpha_x = x \cdot w_x. \tag{37}$$

Similarly for the unscaled variable X:

$$\alpha_X = X \cdot w_X. \tag{38}$$

However, note that $\alpha_x = \alpha_X$, so that

$$x \cdot w_x = X \cdot w_X \tag{39}$$
$$X/2^{b_x} \cdot w_x = X \cdot w_X \tag{40}$$
$$\tag{41}$$

Factoring out X and expanding w, it can be shown that

$$\mu_x = 2^{b_x} \mu_X. \tag{42}$$

Equivalently,

$$w_x = 2^{b_x} w_X. \tag{43}$$

6.2 Principle of Dimensional Homogeneity

The principle of dimensional homogeneity states:

> Only physical quantities having the same units may be compared, equated, added, or subtracted.

Example 6.1 (Dimensional Homogeneity)

Richard ran two kilometers during baseball practice after school and then walked another 800 meters home. What is the total distance Richard traveled after school?

Solution

Kilometers cannot be added to meters. To add the two distances, either kilometers must be converted to meters or meters must be converted to kilometers. Choosing the latter,

$$d = 2\,[\text{km}] + 800\,[\text{m}] \times \frac{1\,[\text{km}]}{1000\,[\text{m}]} \tag{44}$$

$$= 2.8\,[\text{km}]\,. \tag{45}$$

6.3 Principle of Scaling Homogeneity

The principle of scaling homogeneity for signed, fixed-point arithmetic states:

> Only signed, fixed-point quantities having the same scaling may be compared, equated, added, or subtracted.

A common situation which often must be addressed when the operations of addition or subtraction are performed is that the number of resulting bits may be required to grow after the operations in order avoid overflow and maintain precision (see section 4.5). This is referred to as *bit growth*.

Example 6.2 (Scaling Homogeneity: DC Cancellation)

A signal $x[n]$ contains significant amounts of dynamic DC offset that is to be removed. Figure 2 represents an architecture for estimating and cancelling the DC offset:

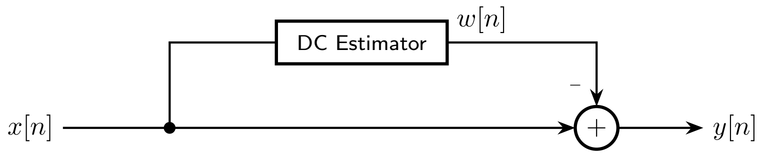

Figure 2: DC Canceller

Assume the input signal $x[n]$ is scaled $A(15, 0)$ and the DC estimator output $w[n]$ is a 32-bit signed two's complement integer scaled $A(15, 16)$. Using the principle of scaling homogeneity, determine a method of computing $x[n] - w[n]$ which will avoid overflow.

Solution

First convert $x[n]$ to a 32-bit value. If we sign-extend $x[n]$ on the right to a 32-bit value $x_s[n]$, the result will be scaled $A(15, 16)$.

If we were to simply compute $x_s[n] - w[n]$ using an $A(15, 16)$ scaling, we would be obeying the principle of scaling homogeneity, but we would run the risk of overflow. For example, if $x_s[n]$ is full-scale positive and $w[n]$ is full-scale negative, then the difference $x_s[n] - w[n]$ would overflow.

Instead, in order to avoid overflow and minimize the loss of precision, and in order to utilize typical word lengths available in programming languages such as C, we perform a literal right shift of both $x_s[n]$ and $w[n]$ by $n = 1$ (see section 4.12.1.2) to obtain $x_{sr}[n]$ and $w_r[n]$, respectively, both scaled $A(16, 15)$. Such a literal right shift will result in no loss of precision in $x_s[n]$ and loss of only one fractional bit in $w[n]$.

We may then apply the principle of scaling homogeneity to compute $y[n] = x_{sr}[n] - w_r[n]$, so that $y[n]$ is also a 32-bit signed two's complement value scaled $A(16, 15)$. Bit growth and overflow are avoided at the cost of a very small loss of precision in the DC estimate.

These principles may be applied individually or *in-toto*.

6.4 Examples

Example 6.3 (Inertial Measurment Dimensional Analysis)

An inertial sensor provides an estimate of linear acceleration $\hat{\alpha}(t)$ which is proportional to a voltage $v(t)$:

$$\hat{\alpha}(t) = k \cdot v(t), \tag{46}$$

where the value of k depends on the units used to represent $\hat{\alpha}(t)$,

$$k = \kappa \cdot \frac{u_{\hat{\alpha}}}{u_v}, \tag{47}$$

where

$$\kappa = \text{constant of proportionality specific to these units} \tag{48}$$
$$u_{\hat{\alpha}} = \text{units used to represent } \hat{\alpha} \tag{49}$$
$$u_v = \text{units used to represent } v \tag{50}$$

For this system, $\kappa = 10$ when $u_{\hat{\alpha}} = \text{m/s}^2$ and $u_v = \text{volts}$, so that the acceleration estimate $\hat{\alpha}(t)$ in m/s^2 can be determined from the sensor voltage $v(t)$ in volts as

$$\hat{\alpha}(t) = v(t)\,[\text{volts}] \cdot \left[\frac{10\text{m}}{\text{s}^2 \cdot \text{volt}}\right]. \tag{51}$$

For example, if $v(t_0) = 1\,[\text{volt}]$, then $\hat{\alpha}(t_0) = 10\,\left[\text{m/s}^2\right]$.

The linear acceleration is processed digitally by converting the analog voltage $v(t)$ to a signed, two's complement 16-bit digital value X using a 16-bit bipolar A/D converter with a signed, two's complement format and a reference voltage of $-1\,[\text{volt}]$ for a full-scale negative value $X = -32768$. If we consider the incoming A/D samples to be scaled $A(1, 14)$, then

1. What are the corresponding scaled and unscaled weights?

Solution

We know that -32768 corresponds to -1.0 volts. From equation 51,

$$\hat{\alpha}(t) = v(t)\,[\text{volts}] \cdot \left[\frac{10\text{m}}{\text{s}^2 \cdot \text{volt}}\right] \tag{52}$$
$$= -1\,[\text{volts}] \cdot \left[\frac{10\text{m}}{\text{s}^2 \cdot \text{volt}}\right] \tag{53}$$
$$= -10\left[\frac{\text{m}}{\text{s}^2}\right] \tag{54}$$

Using the equation

$$\alpha_x = x \cdot w_x, \tag{55}$$

we can determine the scaled weighting from equation 37,

$$w_x = \frac{\alpha_x}{x}, \tag{56}$$

and using the fact that $\hat{\alpha} = \alpha_x$,

$$w_x = \frac{-10\left[\frac{\text{m}}{\text{s}^2}\right]}{-32768/2^{14}} \tag{57}$$
$$= 5\left[\frac{\text{m}}{\text{s}^2}\right]. \tag{58}$$

The unscaled weighting can be found from equation 43:

$$\begin{aligned}\mu_X &= \frac{\mu_x}{2^{b_x}} && (59)\\ &= \frac{5\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]}{2^{14}} && (60)\\ &= 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]. && (61)\end{aligned}$$

Unscaled Check:

$$32768 \times 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] = 10\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]. \tag{62}$$

Scaled Check:

$$\frac{32768}{2^{14}} \times 5\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] = 10\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]. \tag{63}$$

2. What are the maximum positive and maximum negative accelerations that can be sensed?

Solution

We use the unscaled values and weights:

Maximum Positive:

$$\alpha_x(maxpos) = 32767 \times 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] = 9.9999694\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \tag{64}$$

Maximum Negative:

$$\alpha_x(maxneg) = -32768 \times 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] = -10\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \tag{65}$$

3. Assuming quantization using rounding is used, what is the quantization step size in m/s^2 and what is the mean-square error of the quantization noise?

Solution

The quantization step size is

$$\alpha_x(min) = 1 \times 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] = 3.051757 \times 10^{-4}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \tag{66}$$

Example 6.4 (Inertial Measurement System Bias)

The inertial sensor voltage $v(t)$ in the previous example with the system at rest can be modeled as a Gaussian random variable with an average value of 2.7803×10^{-3} [volts]. What is the bias of the system in m/s^2?

Solution

If θ is a value to be estimated and $\hat{\theta}$ is an *estimator* for θ, then the *bias* $B(\theta)$ of the estimator is defined as

$$B(\theta) = E\{\hat{\theta}\} - \theta, \tag{67}$$

where $B(\theta)$ is the bias, $\hat{\theta}$ is the estimator, and θ is the true value of the quantity being estimated (See Hayes [9, p.72] and Kay [10, p.18]).

At rest, the true value of $v(t)$ is 0 [volts]. Therefore the bias in volts is

$$\begin{aligned} B(\theta) &= E\{\hat{\theta}\} - \theta && (68)\\ &= 2.7803 \times 10^{-3}\,[\text{volts}] - 0\,[\text{volts}] && (69)\\ &= 2.7803 \times 10^{-3}\,[\text{volts}]\,. && (70) \end{aligned}$$

Note that we use the Principle of Dimensional Homogeneity (section **??**) in the equation above.

To convert this voltage in m/s^2, use equation 51:

$$\begin{aligned} \hat{\alpha}(t) &= v(t)\,[\text{volts}] \cdot \left[\frac{10\text{m}}{\text{s}^2 \cdot \text{volt}}\right] && (71)\\ &= 2.7803 \times 10^{-3}\,[\text{volts}] \cdot \left[\frac{10\text{m}}{\text{s}^2 \cdot \text{volt}}\right] && (72)\\ &= 2.7803 \times 10^{-2}\left[\frac{\text{m}}{\text{s}^2}\right]. && (73) \end{aligned}$$

7 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):

$$y(n) = \frac{1}{N}\sum_{k=0}^{N-1} x^2(n-k). \tag{74}$$

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, $A(15, 0) \cdot A(15, 0) = A(31, 0)$. 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.

8 Applications of Fixed-Point Binary Arithmetic

9 Acknowledgments

Thanks and acknowledgements to:

1. 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;
2. Dr. Donald Heckathorn, for introducing me to the $A(x, y)$ and $U(x, y)$ fixed-point binary notations and for nurturing my knowledge of fixed-point arithmetic during the same time-frame;
3. 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;
4. Dan Boschen.

10 Terms and Abbreviations

ALU Arithmetic Logic Unit.

DSP Digital Signal Processor, or Digital Signal Processing.

classes of integers

$\mathbb{Z} = \{\ldots, -2, -1, 0, +1, +2 \ldots\}$ (all integers).

$\mathbb{Z}^+ = \{0, 1, 2, \ldots\}$ (non-negative integers).

$\mathbb{P} = \{1, 2, 3, \ldots\}$ (positive integers).

wind-up Wind-up refers to the problem of increasing (or decreasing) values in a perfect digital integrator. Two's complement arithmetic can handle wind-up when the final value is back within the two's complement range. See section **??**.

11 References

[1] "Fixed-Point Numbers," MATLAB. (), [Online]. Available: `https://www.mathworks.com/help/simulink/ug/fixed-point-numbers.html`.

[2] R. Yates, "practical considerations in fixed-point arithmetic: fir filter implementations,"

[3] T. W. Hungerford, *Algebra*. Springer-Verlag, 1974.

[4] M. Morris Mano, *Computer Engineering Hardware Design*. Prentice Hall, 1988.

[5] Charles H. Roth, *Fundamentals of Logic Design*. West Publishing Company, 1992.

[6] "q-values in the watch window," *Texas Instruments*, vol. SPRA109, Feb. 2002.

[7] "axd and armsd debuggers guide (arm developer suite, version 1.2)," *ARM Limited*, vol. DUI 0066D, 1999.

[8] "international system of units," *Wikipedia*,

[9] Monson H. Hayes, *Statistical Digital Signal Processing and Modeling*. Wiley, 1996.

[10] Steven M. Kay, *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Prentice Hall, 1993.

[11] I. Herstein, *Topics in Algebra*, second. Wiley, 1975.

[12] John R. Durbin, *Modern Algebra: An Introduction*, 3rd. John Wiley & Sons, Inc., 1992.

12 Revision History

Table 1 lists the revision history for this document.

Rev.	SVN Rev.	Date/Time	Person	Changes
PA1	n/a	circa 1997	Randy Yates	Initial Version
PA2	n/a	unknown	Randy Yates	1. Converted to Digital Signal Labs document format. 2. Corrected signed dynamic range in equation 29.
PA3	n/a	29-Mar-2007	Randy Yates	1. Updated document design.
PA4	n/a	17-Apr-2007	Randy Yates	1. Removed "this is a test" from introductory paragraph (!). 2. Added justification of signed division scaling.
PA5	n/a	23-Aug-2007	Randy Yates	Updated shift section.
PA6	n/a	07-Jul-2009	Randy Yates	1. Added derivation of unsigned division result. 2. Changed units notation to remove what looked like a subtractions (−).
PA7	n/a	01-Jan-2013	Randy Yates	1. Changed "right" to "left" in description of binary point position in last paragraph of page four (section 2.1). Thanks to Rick Lyons for finding this error! 2. Added examples to sections 4.12.1.1 and 4.12.1.2.
PA8	n/a	02-Jan-2013	Randy Yates	1. Changed "will lose precision" to "run the risk of losing precision" in section 4.12.1.1.
PA9	2185	17-Sep-2020	Randy Yates	Implemented corrections suggested by Heath Caldwell: 1. Added missing n in equation in section 2.3. 2. Made range variable consistently x in sections 4.3 and 4.4. 3. Changed fonts to TEX Gyre Termes. 4. Added SVN Rev column to this table.
PA10	n/a	03-Jul-2021	Randy Yates	This revision brings massive changes, essentially a largely-rewritten document. Some of the major points: 1. Moved signed and unsigned representations sections under one major heading, "Fixed-Point Binary Representations" (section 2). 2. Added Q representations for both unsigned and signed to section 2. 3. Created an example environment and placed all examples into it. 4. Added accuracy versus precision figure (Figure 1). 5. Changed fonts to tgschola.
PA11	n/a	present	Randy Yates	1. Changed bibliography method to biblatex using the biblatex-ieee package. 2. Added definition of fixed-point arithmetic to the introduction.

Table 1: Revision History

A Signed Binary Numbers

Historically, there have been three methods used to represent signed binary numbers: signed magnitude, one's complement, and two's complement ([4, section 1-5] and [5, section 20.1]). Examples of a 3-bit number under each of these three representations are shown in Table 2.

In an N-bit signed binary integer, there are N binary digits $0, 1, \ldots, N-1$ (counted right to left). Binary digit $N-1$ is a sign bit: 0 denotes positive values and zero (+0, or simply 0), and 1 denotes negative values (possibly negative zero, -0). The remaining $N-1$ binary digits $0, 1, \ldots, N-2$ are used to represent the magnitude of the integer in some form as determined by the representation.

Note that we will use M to denote the number of bits in the magnitude, so that $M = N - 1$.

The non-negative integers m, $0 \leq m \leq 2^M - 1$ of an N-bit signed binary number have the same mappings to binary values in all representations. It is only the negative numbers which vary depending on representation. The following subsections delve into more details on each.

	Signed Magnitude	One's	Two's
Decimal		Complement	Complement
-4	—	—	100
-3	111	100	101
-2	110	101	110
-1	101	110	111
-0	100	111	—
$+0$	000	000	000
$+1$	001	001	001
$+2$	010	010	010
$+3$	011	011	011

Table 2: Example Signed Binary Representations for $N = 3$

Decimal	**Signed Magnitude**	**One's Complement**	**Two's Complement**
-4	—	—	100
-3	111	100	101
-2	110	101	110
-1	101	110	111
-0	100	111	—
+0	000	000	000
+1	001	001	001
+2	010	010	010
+3	011	011	011

Table 3: Old Example Signed Binary Representations for $N = 3$

A.1 Signed Magnitude

In this representation, bits $0, 1, \ldots, M-1$ specify the magnitude of integer, whether the sign bit is positive or negative.

Zero has two possible representations, +0 (000) and -0 (100). If -0 and +0 are considered equivalent, then this representation under-utilizes the binary information in the sense that 2^N binary states can only represent $2^N - 1$ integers.

Signed magnitude is also awkward when used in digital hardware, thus one of the complement representations are typically used instead.

A.2 One's Complement

In this representation, bits $0, 1, \ldots, M-1$ are the *one's complement* of the magnitude for a non-positive integer, or the normal magnitude for a positive integer. Note that since there two forms of zero, -0 and +0, zero can be negative. In this case, as the rules dictate, the complement applied to the magnitude.

The one's complement of an M-bit (non-negative) integer m, $\bar{m}$, can be expressed as

$$\bar{m} \triangleq (2^M - 1) - m. \tag{75}$$

If m is expressed as a binary number, This is equivalent to the bitwise inversion of the entire M-bit word m.

As in signed magnitude, zero has two possible representations, +0 (000) and -0 (100), and this representation under-utilizes the binary information.

A.3 Two's Complement

In this representation, bits $0, 1, \ldots, M-1$ are the *two's complement* of the magnitude for a negative integer, or the magnitude for a positive integer.

The two's complement of an M-bit (non-negative) integer m, $\tilde{m}$, can be expressed as

$$\tilde{m} \triangleq (2^M - 1) - m + 1 \tag{76}$$
$$= \bar{m} + 1. \tag{77}$$

This shows that the two's complement is the one's complement plus one.

Two's complement representation has the useful property that it fully utilizes the binary information since the redundant "-0" no longer occupies a value. Note also that the full-scale negative value is now $-2^{(N-1)}$.

B Binary Addition

Our goal in this section is to derive how to implement two's complement signed binary addition, but in order to do that, let us first review how to implement unsigned binary addition.

B.1 Unsigned Binary Addition

A half adder is shown in Figure 3. It has two binary inputs, w_i and x_i, and two binary outputs, the result y_i and the carry out co_i. Table 4 is the truth table for the half adder.

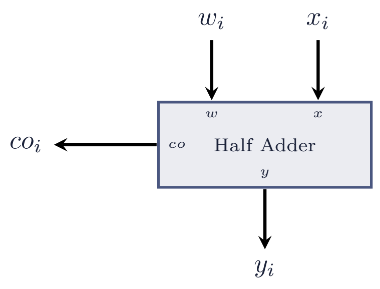

Figure 3: Half Adder

Inputs		Outputs	
w_i	x_i	co_i	y_i
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 4: Half Adder Truth Table

A full adder (shown in Figure 4) extends the half adder so that it has three inputs: binary inputs w_i and x_i, and the carry in ci_i from a previous stage's carry out co_{i-1}. Table 5 is the truth table for the full adder.

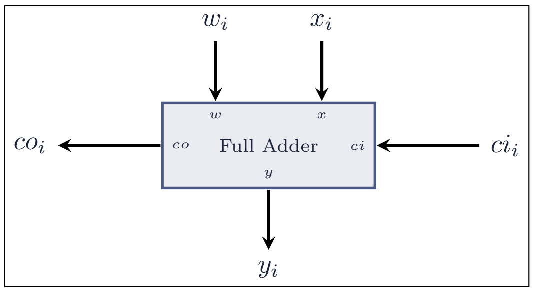

Figure 4: Full Adder

Inputs			Outputs	
ci_i	w_i	x_i	co_i	y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 5: Full Adder Truth Table

We can connect N full adders together to make an *N-bit unsigned binary (parallel) adder*. An example is shown in Figure 5 for $N = 4$.

Now note that the range of an unsigned, N-bit binary word is 0 to $2^N - 1$. Thus the maximum value S_M which can result from adding two N-bit binary words $w + x$ is

$$\begin{aligned} S_M &= 2^N - 1 + 2^N - 1 &(78)\\ &= 2^{N+1} - 2. &(79)\end{aligned}$$

In other words, the number of bits required to represent an N-bit addition without overflow is $N + 1$.

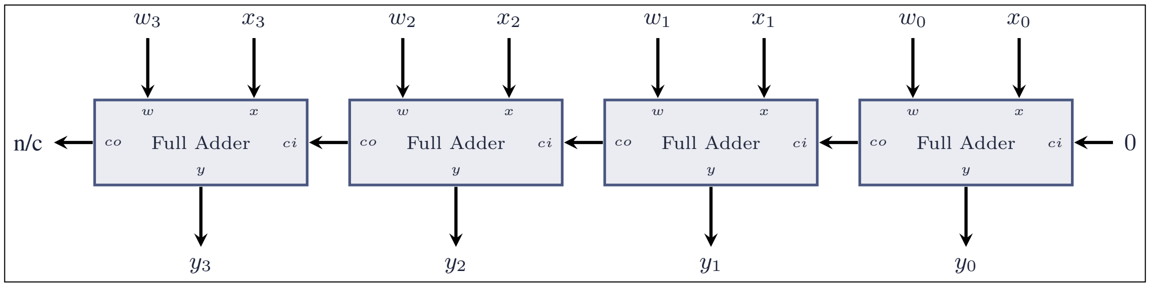

Figure 5: Four-Bit Unsigned Binary Adder, $y = w + x$, $N = 4$

Now also consider this fact regarding unsigned binary parallel adders: the output of the ith full adder does not depend on any of the subsequent full adders $i+1, i+2, \ldots, N-1$. This means that bits 0 to $N-1$ of a full adder are correct regardless of whether or not the adder is extended to $N+m$, bits, $m >= 1$.

Therefore if we use N bits to perform the sum of two N-bit unsigned binary values $x+y$, then if the result does not overflow, the result $x+y$ is correct and in the range 0 to 2^N-1. But if the result does overflow, then the output result is $x+y-2^N$, since the $N+1$ bit is truncated (i.e, subtracted), and again in the range 0 to 2^N-1. Thus the N-bit unsigned binary addition (denoted $\oplus$) of two N-bit values $x+y$ is given as

$$x \oplus y = (x+y) \bmod 2^N, \tag{80}$$

and where we define the modulus function a mod n as follows:

Definition: Modulus Function (a mod n)

If a is a non-negative integer and n is a positive integer, we define the modulus function a mod n to be the remainder r of a/n, where $0 \leq r < n$.

Examples:

- $13 \bmod 6 = 1$
- $8 \bmod 2 = 0$
- $0 \bmod n = 0$.

B.2 Signed Binary Addition

This section utilizes abstract algebra, e.g. from [11] or [12].

Two's complement addition is performed using the exact same full adder arrangement as in Figure 5. To prove this we need the following:

1. The set of N-bit, unsigned fixed-point integers $P_0(N)$ as defined in section 2.1.
2. The set of N-bit, signed fixed-point integers $R_0(N)$ as defined in section 2.1.
3. A one-to-one and onto mapping ψ from $R_0(N)$ to $P_0(N)$, $\psi : R_0(N) \to P_0(N)$, defined as follows.
   Given $N \in \mathbb{P}$ there are 2^N elements of $R_0(N)$ and $P_0(N)$. We map the 2^{N-1} negative elements of $R_0(N)$, -2^{N-1}, $-2^{N-1}-1$, $-2^{N-1}-2, \ldots, -1$, to the the 2^{N-1} elements of $P_0(N)$, 2^{N-1}, $2^{N-1}+1$, $2^{N-1}+2, \ldots, 2^N-1$, and the 2^{N-1} positive

elements of $R_0(N), 0, 1, 2, \ldots, 2^{N-1} - 1$ to the the 2^{N-1} elements of $P_0(N), 0, 1, 2, \ldots, 2^{N-1} - 1$. Since this mapping is finite and onto, it is also 1-1, and therefore also has the obvious inverse mapping ψ^{-1}.

To prove our assertion, we need to show that ψ is a isomorphism from

C Quantization

C.1 Rounding

C.2 Truncation

C.3 Dithering