Signal Processing Systems (521279S)
Part 1 : Number representations and arithmetic

Version 1.3

October 27, 2025

Goals of the study. DSP systems are number-crunching machines. One of the basic issues in the system design is, which format of representation should be used for real numbers. There are two main alternatives, fixed-point and floating-point. Some concepts for characterizing them are studied, which give an idea to their performance. Hardware for fixed-point arithmetic is simpler, but there are reasons for favoring the floating-point format also in low-power designs.

Contents

History

V1.0 15.10.2019 Original version.
V1.1 20.10.2022 Add information about floating-point dynamic range (Sec. 4.2).
V1.2 21.10.2024 Edit example in Sec. 5.3.
V1.3 27.10.2025 Add subsection 3.5. on NN quantization.

1 Introduction

One of the issues in the design of a DSP system is the choice of the number representation. Real number data types and corresponding arithmetic belong either to the *fixed-point* or *floating-point* category (Fig. 1). Basically, in both forms, a certain number of significant bits is used to represent an integer value and associated scaling maps it to some real number. The representations address the scaling in a different way: floating-point representations have an exponent part which encodes the scaling value whereas fixed-point numbers do not have this part. Related to this, floating-point hardware performs automatic scaling of the numbers, whereas simpler fixed-point hardware does not. In general, it is easier to program with a floating-point representation as one does not have to worry much about how small or large the result of a computational step can be.

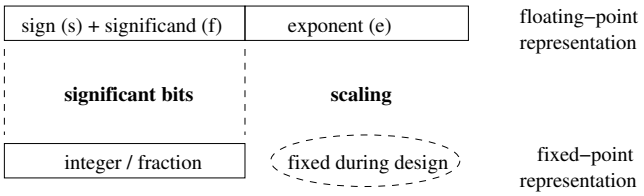


Figure 1: Principles of fixed-point and floating-point representations.

Before discussing these formats in more detail, it is necessary to know the following terminology used to characterize the capabilities of them:

- **Word length** is the total number of bits in a representation.
- **Range** of a representation is defined by the smallest and largest values that can be represented [KG05]. The result of a computation should always stay in the range of the output representation. Otherwise, we have an *overflow*.
- **Dynamic range** is a measure for the ratio between the largest and smallest number representable in a given data format [Lap+97; KG05]. To be more specific, we define it here as the ratio between the largest and the smallest representable non-zero absolute value ($AMax$, $AMin$) expressed in decibels:

$$\text{Dynamic range in dB} = 20\log_{10}(AMax/AMin). \tag{1}$$

  When the dynamic range is large, it is possible to represent both very small and very large values.
- **Precision** of a number representation can be defined as the smallest step between two consecutive numbers [KG05]. This step size is often referred

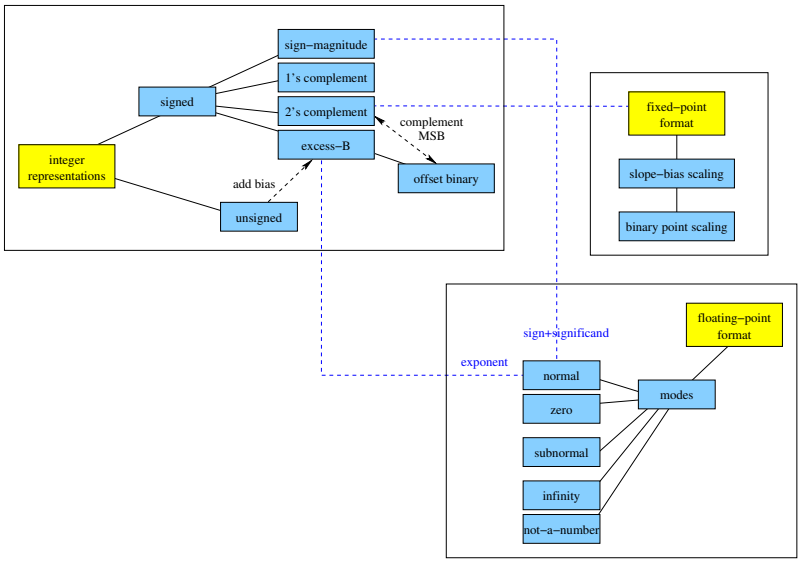


Figure 2: A mind map of the number representations: integers, fixed-point, floating-point.

to as the *unit in least position*, *ulp* [Gol91]. Due to limited word length, real values are *rounded* and precision is related to the rounding error (quantization noise). With weak precision, also *underflows* become possible.

- **Precision (in bits)**: A representation scheme can also be characterized in terms of *maximal precision*, which is defined as the number of bits that can be used to refine the value *under particular scaling*. For a fixed-point representation, this is the same as the word length, because scaling is fixed. So, saying that a fixed-point representation has a precision of p bits means that the word length is p. For a floating-point representation, precision in bits can be understood as the number of the bits in the sign and the significand ($\neq$ word length).[1]

We discuss now to the ways of representing integers, which form the basis of fixed-point and floating-point formats (see Fig. 2).

[1]Note that in the formats defined by the IEEE-754 standard, the word 'precision' is used in connection to the word length: the *single* precision refers to the format having word length of 32 bits, and the *double* precision refers to a 64-bit format.

2 Integers

Fixed-point and floating-point representations are based on representing integers in suitable ways, and therefore we go through various binary number systems for this purpose.

2.1 Representations

Let us denote the string of bits in a p-bit representation with

$$a_{p-1}a_{p-2}...a_1a_0,$$

where each $a_i \in \{0,1\}$ corresponds to one bit. In the case of **unsigned** integers, this sequence corresponds to the value

$$V_{int} = \sum_{i=0}^{p-1} a_i 2^i. \tag{2}$$

The range is $[0, 2^p - 1]$. For example, $10110 = 16 + 4 + 2 = 22$. The range of the 5-bit representation is $[0, 31]$.

The main conventions used to represent **signed** integers are:

- **sign-magnitude**: the string of bits represents the integer value

$$V_{int} = (-1)^{a_{p-1}} \times \sum_{i=0}^{p-2} a_i 2^i. \tag{3}$$

  The range is $[-2^{p-1} + 1, 2^{p-1} - 1]$. For example, $10110 = -6$, and the range of the 5-bit representation is $[-15, 15]$.

- **one's complement** representation complements each bit to represent the negative value, and the value represented is

$$V_{int} = -a_{p-1}(2^{p-1} - 1) + \sum_{i=0}^{p-2} a_i 2^i. \tag{4}$$

  The range is $[-2^{p-1}+1, 2^{p-1}-1]$. For example, $10110 = -9$, and the range of the 5-bit representation is $[-15, 15]$. Both 00000 and 11111 represent zero.

- **two's complement** scheme is defined by

$$V_{int} = -a_{p-1}2^{p-1} + \sum_{i=0}^{p-2} a_i 2^i. \tag{5}$$

The range is $[-2^{p-1}, 2^{p-1}-1]$. For example,

$$10110 = -1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 = -16+4+2 = -10,$$

and the range of the 5-bit representation is $[-16, 15]$.

In all these approaches, the left-most bit a_{p-1} represents the sign of the number and it is therefore called the *sign bit*. The bit strings for the positive numbers are always the same.

Another representation for signed integers is the **biased** representation. In this case, the bit string represents the value

$$V_{int} = \sum_{i=0}^{p-1} a_i 2^i - B \tag{6}$$

where B is the bias. The range of the representation is $[-B, 2^p - 1 - B]$. Note that $B = 0$ corresponds to the unsigned number representation. Another name for this representation is **excess-B**. The biased representation where the bias $B = 2^{p-1}$ is called the **offset binary** form. In many cases, this is the form of the data obtained from ADC and data fed to DAC [IJ02, p. 809]. An example is given in the last column of Table 1. Note that complementation of the most significant bit (MSB) a_{p-1} converts the offset binary to the 2's complement format.

$a_2a_1a_0$	sign-magnitude	1's complement	2's complement	excess-4
011	+3	+3	+3	-1
010	+2	+2	+2	-2
001	+1	+1	+1	-3
000	0	0	0	-4
111	-3	0	-1	+3
110	-2	-1	-2	+2
101	-1	-2	-3	+1
100	0	-3	-4	0

Table 1: 3-bit signed integer representations. Note that sign-magnitude and 1's complement representations have two possible encodings for zero which is a potential disadvantage, but can also be a useful property.

The choice of the integer formats for encoding real numbers varies. Fixed-point computations are typically done using two's complement numbers. Floating-point numbers use the biased representation for exponents and the significant bits are represented by a system resembling the sign-magnitude. The structure of the bit string depends on the implementation of operations (see [Gol91]).

2.2 Arithmetic

Common arithmetic operations with integers that we are interested in are the addition, multiplication, and arithmetic shift. Some technical details of their implementation are presented in Sec. 5.1. Here, we take a look at the word lengths associated with the addition and multiplication operations. We consider just the **two's complement format**.

2.2.1 Addition

In the case of addition, we have the following rules:

- If two p-bit integers are added, the output can be represented with $p+1$ bits. If a p_1-bit and p_2-bit integers are added, the output can be represented with $\max(p_1, p_2)+1$ bits.
- In general, if N p-bit integers are summed, the output can be represented with $p+g$ bits, where $g = \lceil \log_2 N \rceil$.

Note that these rules provide upper bounds for the output word length. It is possible to get along with shorter output word length, if we know that the magnitude of the output is in the range of that word length.

Example 1. Consider the addition of two 8-bit integers in the case, where we have knowledge that the first input is in the range $[-120, -90]$ and the second input is in the range $[20, 50]$. Then the output is certainly in the range $[-100, -40]$, which can be represented with 8 bits. ■

Example 2. Consider the addition of three 8-bit integers, where the first input is in the range $[20, 90]$, second is in $[50, 65]$, and third is in $[-120, -90]$. We have $N = 3$ and therefore $g = \lceil \log_2 3 \rceil = 2$. So, 10 bits can represent the output, but the output range is $[-50, 65]$, which can be represented with 8 bits.

After the addition of the first two inputs, we have a value in the range $[70, 155]$, which requires 9 bits. However, if the addition of two's complement numbers does not saturate, we do not need 9 bits for storing intermediate results! The reason is that we have **modular arithmetic** (see Fig. 3). For example, consider computation of $90 + 65 + (-120)$, $90 + 65 = 155$ does not fit signed 8-bit representation, and the result maps to -101 ($155 - 256$). Then, $(-101) + (-120) = -221$ does not fit, and the result maps to 35 ($-221 + 256$). But, $90 + 65 + (-120)$ is 35, which is the correct result. ■

Later in the course, we will discuss CIC filters. Their simple structure is based on modular arithmetic.

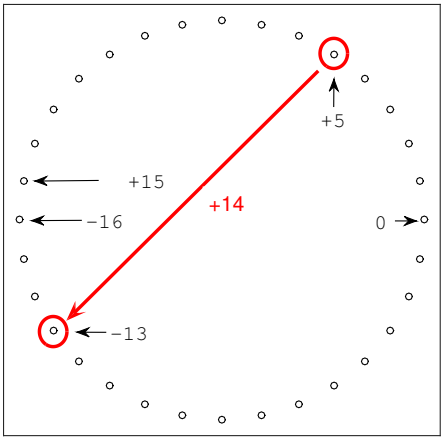


Figure 3: Non-saturating two's complement addition works modularly, that is, the result is one of the numbers in the ring. The ring here is for 5-bit integers. When $+14$ is added to $+5$, we get -13 due to *wrap-around*.

2.2.2 Multiplication

For multiplication, we have the rule: the multiplication of p_1-bit and p_2-bit two numbers with word lengths p_1c and p_2 can be represented with $p_1 + p_2$ bits. So, if two p-bit integers are multiplied, the output can be represented with $2p$ bits. However. if either of the inputs is not the negative maximum (NMAX), the result can always be stored with $2p - 1$ bits. The reason is that only the multiplication NMAX$\times$NMAX produces an output, where the second most significant bit (i.e. the bit after the sign bit) is equal to 1. This fact is exploited in the implementation of the so-called *fractional arithmetic* (see Sec. 3.3).

Example 3. Consider multiplication of 3-bit signed two's complement integers. The positive maximum (PMAX) is equal to 3 (011) and the negative maximum is equal to -4. Obviously, NMAX$\times$NMAX produces the largest positive output, which is 16. With $2p = 6$ bits, the bit string is 010000, and we note that the second bit is one. If we compute NMAX$\times$(NMAX$+1$), we get -12 (001100). The second bit is now zero. ■

In general, knowledge of the range of the inputs can be used to compute the sufficient multiplier output word length. Another idea is to use saturating arithmetic to deal with overflow situations.

3 Fixed-point representation

In the case of fixed-point, we associate scaling with integers. We consider scaling approaches, characteristics of the format, and basics of FIR filtering.

3.1 Scaling approaches

In general, fixed-point representation assigns a meaning to a two's complement integer value V_{int} according to the formula

$$\tilde{v} = sV_{int} + b, \tag{7}$$

where $s > 0$ is called the slope, b is the (offset) bias, and $\tilde{v}$ is the value represented. This is called *slope-bias scaling*. From the viewpoint of arithmetic, it is convenient to have zero bias, so usually $b = 0$.

For convenience, we will denote with triplet

$$(p, s, b)$$

the format for slope-bias scaled fixed-point numbers, whose word length is p.

A special case of slope-bias scaling is the *binary point scaling* defined as

$$\tilde{v} = V_{int}/2^n, \tag{8}$$

that is, $s = 2^{-n}$ and $b = 0$. Here, n denotes the number of fraction bits. Interpretation of individual bits and associated terminology is illustrated in Fig. 4. For convenience, we will denote with

$$\mathrm{s}p.n$$

a signed binary point scaled fixed-point number, whose word length is p.

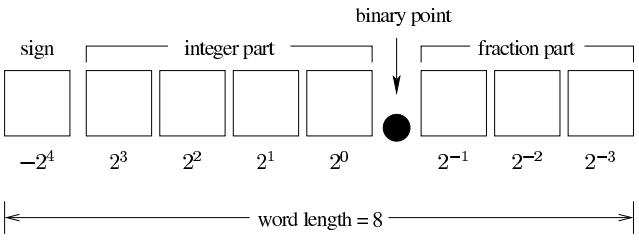


Figure 4: Signed two's complement fixed-point number with 4 integer bits and 3 fraction bits (s8.3). Weights associated with the bits are shown below. The sign bit has negative weighting. The binary point is also called the radix point.

Example 4. Consider the slope bias scaling $(p, s, b) = (8, 0.3, 0)$, and the bit string 10001001. The string corresponds to the integer -119 in two's complement format, and in the specified fixed point format $0.3 \times -119 = -35.7$. Under the binary point scaling with two fraction bits (s8.2), 10001001 corresponds to $0.25 \times -119 + 0 = -29.75$ as the scaling s is 2^{-2}. ■

One typical task in DSP implementation is mapping of various fixed coefficients (e.g. filter coefficients) to the available system format. Let us assume that we want to represent a real value v. Then, under particular slope-bias scaling (p, s, b) we obtain the integer

$$V_{int} = \text{round}((v - b)/s), \tag{9}$$

where round$(\cdot)$ denotes rounding towards the nearest integer value. The result of the operation should be in the range of the p-bit integers (of two's complement format).

In the case of binary point scaling s$p.n$, we have

$$V_{int} = \text{round}(2^n v). \tag{10}$$

Fig. 5 summarizes the discussion. Note that due to the rounding operation, the value represented by V_{int} is generally not equal to the value we wanted to represent exactly, that is, typically $\tilde{v} \neq v$. This is called *rounding error*. In some cases (e.g. FIR filter coefficients), we may instead want to maintain the ratios of multiple numbers in the mapping.

Example 5. In the s8.4 format, the value $v = 0.17$ maps to $V_{int} = \text{round}(2^4 \times 0.17) = 3$ (bit string 00000011). The value represented is $\tilde{v} = 3/2^4 = 0.1875$, so the magnitude of the rounding error is 0.0175. ■

3.2 Range, dynamic range, and precision

Formulas for obtaining the range and precision for the scaling methods are provided in Table 2. There is *a direct tradeoff between the range and precision for a specific word length*: if the range is enlarged by changing the scaling, the precision decreases (ulp gets higher). The dynamic range of a zero-biased fixed-point format is dictated solely by the word length p . According to our definition (recall Eq. 1), it is

$$\text{Dynamic range in dB} = (p - 1)6.02 \text{ dB}.$$

as $20 \log_{10} \frac{AMax}{AMin} = 20 \log_{10} 2^{p-1} = (p - 1)20 \log_{10} 2$. So, each extra bit corresponds to approximately 6 dB improvement in the dynamic range. With 16 bits,

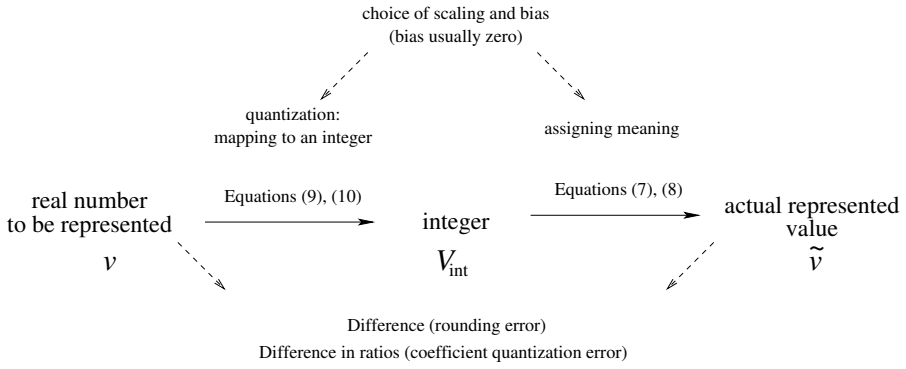


Figure 5: Overview of the fixed-point quantization.

the dynamic range is about 90 dB.[2]

Scaling	Precision	Range	
		Min	Max
Slope-bias (p, s, b)	s	$b - s\,2^{p-1}$	$b + s\,(2^{p-1} - 1)$
Binary point s$p.n$	$1/2^n$	-2^{p-1-n}	$2^{p-1-n} - 2^{-n}$

Table 2: Characteristics of fixed-point mappings of p-bit integers.

Example 6. A real number is in the range (-9.2,+9.2), and the maximum allowed error is 0.01. What is the minimum wordlength if (a) the binary point scaling (b) the slope-bias scaling (zero bias) is used?

(a) Let p denote the unknown word length and let n denote the length of the fraction part. The precision requirement says that

$$\underbrace{(1\ /\ 2^n)}_{\text{ulp}}/2 \leq 0.01$$

which gives

$$n \geq -\log_2 0.02 \approx 5.64$$

[2]Definition of the SQNR (signal to quantization noise ratio) assumes a sine signal and uniformly distributed quantization noise. Computation of signal powers leads to the formula $SQNR = 6.02p + 1.76$ dB where p is the number of bits. This is the definition of the dynamic range used in audio engineering, for example. Note that also this definition has the "6 dB per bit" property, but the offset is different.

Therefore, we use $n = 6$.

The range of two's complement integers is $[-2^{p-1}, 2^{p-1} - 1]$, and the range of the fixed-point number is obtained by dividing that range by 2^n. Therefore,

$$(2^{p-1} - 1)/2^n \geq 9.2$$

which gives the inequality

$$p \geq \log_2(9.2 \times 2^n + 1) + 1 \approx 10.20$$

Therefore, we use the word length $p = 11$, that is, the format is s11.6.

Check: ulp $= 1/2^6 \approx 0.0156$ and the maximum error is therefore less than 0.0078 (ok). For $n = 5$ the precision requirement would not be satisfied. What comes to range, 01111.xxxxxx $\geq$ 15 (ok). If there were one integer bit less 0111.xxxxxx $\leq$ 01000.xxxxxx = 8, that is, we could not represent any values above or equal to 8.

(b) The precision is equal to the scaling factor s. Therefore, $s/2 \leq 0.01$. We have $V_{int} = \tilde{v}/s$ and get $|V_{int}| \geq (1/0.02)|\tilde{v}|$.

For $|\tilde{v}| = 9.2$, we get $|V_{int}| \geq 460$. To represent the integers in the range [-460,+460], we need at least 10 bits since 10 bits can encode the integers in the range [-512,+511]. We use the full range of integers to maximize precision, and $s \times 511 = 9.2$ gives $s = 0.018$. So, the result is that 10 bits with the slope $s = 0.018$ can be used as a fixed-point representation (check it!).

When we compare these two solutions, we note that the solution in (a) requires 11 bits, but the solution (b) only 10. The latter solution utilizes fully the dynamic range of 10-bit representation and the precision fulfills the requirement. In (a), the values utilize only partially the range of 11-bit representation. ■

Example 7. Let us consider design of an 8-bit representation for values in the range $[-\pi, \pi)$. If we use binary point scaling, we must have two integer bits, because we must be able to represent 3. The range of such representation is unnecessarily large, $[-4, 4 - 2^{-5}]$, and the precision is $2^{-5} = 0.03125$.

If we use the slope-bias scaling with zero bias and $s = \pi/2^7$, we get a representation with the range $\pi \times [-1, (1-2^{-7})]$ and the precision is ulp$= s \approx 0.02454$. Available dynamic range of the 8-bit representation is now fully exploited. ■

Example 8. In neural network quantization, one may use for NN weights so-called asymmetric representation [Nag+21, Sec. 2.2], which can be considered as an example of slope-bias scaling. Note that the zero point z corresponds to $-b/s$

and z must be an integer, as then it is easy to remove bias. In terms of Eq. 9, the stored integer is

$$\begin{aligned} V_{int} &= \text{round}((v-b)/s) \\ &= \text{round}(v/s - b/s) \\ &= \text{round}(v/s + z) \\ &= \text{round}(v/s) + z \end{aligned}$$

In addition, clamp operation is applied to V_{int}, to guaranty that the result fits to the available wordlength. We note that $V_{int} - z = \text{round}(v/s)$, that is, it is easy to eliminate bias by simple subtraction, before use of 2's complement arithmetic. ■

3.3 Modes of arithmetic: integer and fractional

As the fixed-point numbers just scale integers, implementation of the arithmetic can based on integer arithmetic directly (recall discussion in Sec. 2.2). Considering multiplication, it was also mentioned that the bit after the sign bit is needed only in a special case. Based on this observation, there exists multipliers based on the so-called *fractional* arithmetic e.g. in some fixed-point processors. The operation of such multipliers is illustrated in Fig. 6.

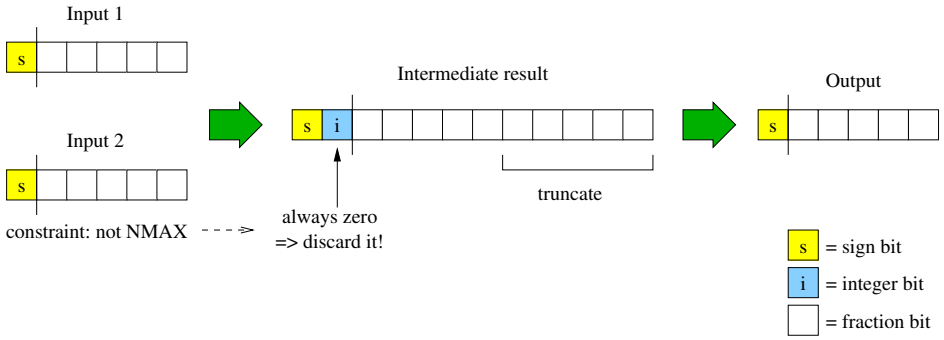


Figure 6: Multiplication in fractional arithmetic.

3.4 Integer MAC unit and FIR filtering

Let us consider now one of the basic building blocks of DSP systems, the multiply-accumulate (MAC) unit implemented using integer arithmetic. Especially, we consider implementation of finite impulse response (FIR) filters with it.

Basic structure of a fixed-point MAC unit is shown in Fig. 7. The multiplier takes in two p-bit fixed-point numbers, and outputs a $2p$-bit result as discussed in Sec. 2.2. The accumulator output is initially reset to zero, and incremented by multiplication results. The result is stored in a $(2p+g)$-bit register, where the extra g bits, so-called *guard bits*, are used to prevent overflow.

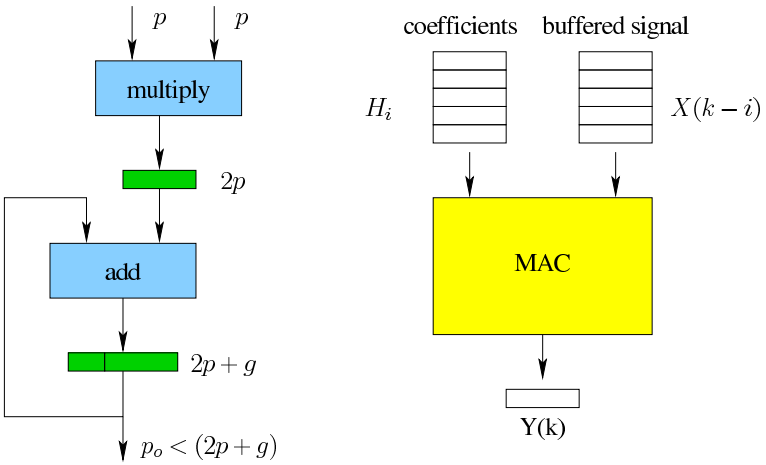


Figure 7: Fixed-point MAC unit and FIR I/O.

In the case of FIR filtering, the first input of MAC can be fed by the fixed-point values representing the coefficients

$$\{h_i : i = 0, 1, ..., N-1\}.$$

Another input of MAC is fed by buffered fixed-point values, which represent the current and past signal samples at time instant k,

$$\{x(k), x(k-1), ..., x(k-(N-1))\}.$$

The accumulator is reset to zero first. Then, the MAC operation is repeated so that, after N clock cycles, the result

$$y(k) = \sum_{i=0}^{N-1} h_i x(k-i) \tag{11}$$

becomes available at the accumulator output.

3.4.1 Coefficient quantization

Under the slope-bias scaling with zero bias, the fixed-point values representing the coefficients h_i are mapped to integers H_i by the rule

$$H_i = \text{round}(h_i/s) = \text{round}(s^{-1}h_i), \tag{12}$$

where s is the slope. Scaling by s^{-1} must be such that the resulting integers can be represented using p bits. However, we may force it with clamp operation:

$$H_i = \text{clamp}(\text{round}(s^{-1}h_i)), \tag{13}$$

where

$$\text{clamp}(x) = \begin{cases} -2^{p-1} & \text{if } x < -2^{p-1} \\ 2^{p-1}-1 & \text{if } x > 2^{p-1}-1 \\ x & \text{otherwise.} \end{cases} \tag{14}$$

Due to the rounding and clamping, the ratio of H_i and H_j may not correspond to the ratio of h_i and h_j.[3] This is called the *coefficient quantization error*.

3.4.2 Scaling issues

We can list some issues for choosing the scaling s^{-1}:

- We would like to use as large s^{-1} as possible with minimal clamping as then we would utilize the dynamic range of p-bit representation well, as the mapped coefficients are spread over the available range of integers.
- However, the coefficients H_i of the fixed-point filter should maintain the ratios of original coefficients h_i. Therefore, it may be so that the largest possible s^{-1} is not the best one from the viewpoint of coefficient quantization noise.

On the other hand, assume that we are free to choose the word lengths on the data path originating from the coefficients. Then:

- If we are tuning the word lengths of the data path, then we would like to use as small s^{-1} as possible. For example, if s^{-1} is halved, then one bit less is sufficient for representing coefficients, multiplication output, and accumulation output.
- However, we want again to avoid quantization noise, and maintain the ratios of original coefficients. Typically, when s^{-1} is made smaller, the quantization noise increases.

[3] In other words, H_i corresponds to $\tilde{h}_i = sH_i$ and H_j corresponds to $\tilde{h}_j = sH_j$. It can be that $\tilde{h}_i/\tilde{h}_j \neq h_i/h_j$.

Evaluation of the severity of quantization error can be based on the mean-squared error computed using

$$\text{MSE} = (1/N) \sum_{i=0}^{N-1} (\tilde{h}_i - h_i)^2, \tag{15}$$

where $\tilde{h}_i = sH_i$.

The maximum magnitude of the filter output (range) can be estimated[4] with

$$\text{RANGE} = 2^{p-1} \sum_{i=0}^{N-1} |H_i|, \tag{16}$$

where p is the number of bits used for the input signal. The number of bits needed at the accumulator output is

$$p_a = \lceil \log_2(\text{RANGE}) \rceil + 1. \tag{17}$$

According to Sec. 2.2, the number of the guard bits g for the case shown in (Fig. 7) can be computed using $g = \lceil \log_2 N \rceil$. Based on the coefficient values, some of those bits may not be used, that is,

$$p_a \leq 2p + g. \tag{18}$$

So, better tuning of the word lengths on the data paths is possible *if the coefficient values are known beforehand*. If nothing is known, we must have word lengths that are sufficient for the worst case, or the programmer must be informed about the limits.

Example 9. Let us assume that we must implement a 3-tap FIR filter, whose coefficients are specified as $h_0 = 0.4$, $h_1 = -0.8$, and $h_2 = 0.4$. Available word length for storing coefficients is $p = 8$ bits, the signal input word length is the same, and the multiplier of the MAC unit, which is non-saturating, has 15-bit output (i.e. $2p - 1$).

First, we try to exploit the whole dynamic range of the 8-bit representation for coefficients, and choose scaling $s^{-1} = 128/0.8 = 160$, which gives $H_0 = H_2 = \text{round}(63.5) = 64$ and $H_1 = -128$. However, wrap-around is possible with a non-saturating multiplier as H_1 corresponds to NMAX. We want to avoid this value, we could try to use for example $s^{-1} = 127/0.8$, which gives $H_0 = H_2 = 64$ and $H_1 = -127$. Now quantization error is introduced to coefficients as $-127/64 \neq -0.8/0.4$.

[4]The largest possible output is obtained by some buffered input signal composed of positive and negative maxima PMAX$= 2^{p-1} - 1$ and NMAX$=-2^{p-1}$. The formula (16) does not take into account that PMAX is not 2^{p-1}, so it is not completely accurate.

Both problems are solved, if we use $s^{-1} = 126/0.8$, which gives $H_0 = H_2 = 63$ and $H_1 = -126$. Now, MSE $= 0$. For the 8-bit input signal, RANGE $= 2^{p-1} \sum_{i=0}^{N-1} |H_i| = 2^7 \times (63 + 126 + 63) = 32256$. The number of accumulator bits used is $p_a = \lceil \log_2(\text{RANGE}) \rceil + 1 = \lceil 14.9773 \rceil + 1 = 16$, which equals $2p$. According to the number of coefficients (3), we expect that 2 guard bits are needed, but actually there is only one extra bit compared to the multiplier word length.

We also note that $s^{-1} = 2/0.8$ would give $H_0 = H_2 = 1$ and $H_1 = -2$. So, it is possible to use just 2 bits (-2 = 10, 1 = 01) for representing the numbers without error in their ratios. One can reduce the word lengths on the MAC data path significantly. For example, 10 bits in the accumulator register would be sufficient.

Finally, note that using MAC does not seem reasonable as implementation could be based on arithmetic shifting (see Sec. 5.1) and two additions/subtractions: $Y(k) = X(k) - 2^1 X(k-1) + X(k-2)$. ■

3.5 Neural network quantization

Neural networks (NNs) can provide high-performance data-based solutions to many signal processing problems. However, original trained NNs operating with floating-point numbers can be complex. In the case of platforms providing just integer arithmetic, one must quantize NNs so that available integer arithmetic can be used. For good introduction, see [Nag+21].

Understanding MAC units is important also in this context. For example, fully connected (FC) layers of NNs can be implemented using MAC arithmetic. In practice, the impact of FC layer weight quantization must be studied using some calibration data: comparison of the outputs of the original and quantized NNs reveals if sufficient performance is obtained.

4 Floating-point representation

Floating-point numbers are composed of three parts: sign bit, exponent and significand. The exponent defines the range of representation, whereas number of bits in the significand part (aka mantissa) defines the available precision. Standard iEEE 754 formats are typically used in processors. For example, in the IEEE 754 single precision format the word length is 32 bits, the significand part takes 23 bits and the exponent 8 bits (Fig. 8). In the double precision format the word length is 64 bits, the significand takes 52 bits and the exponent part takes 11 bits.

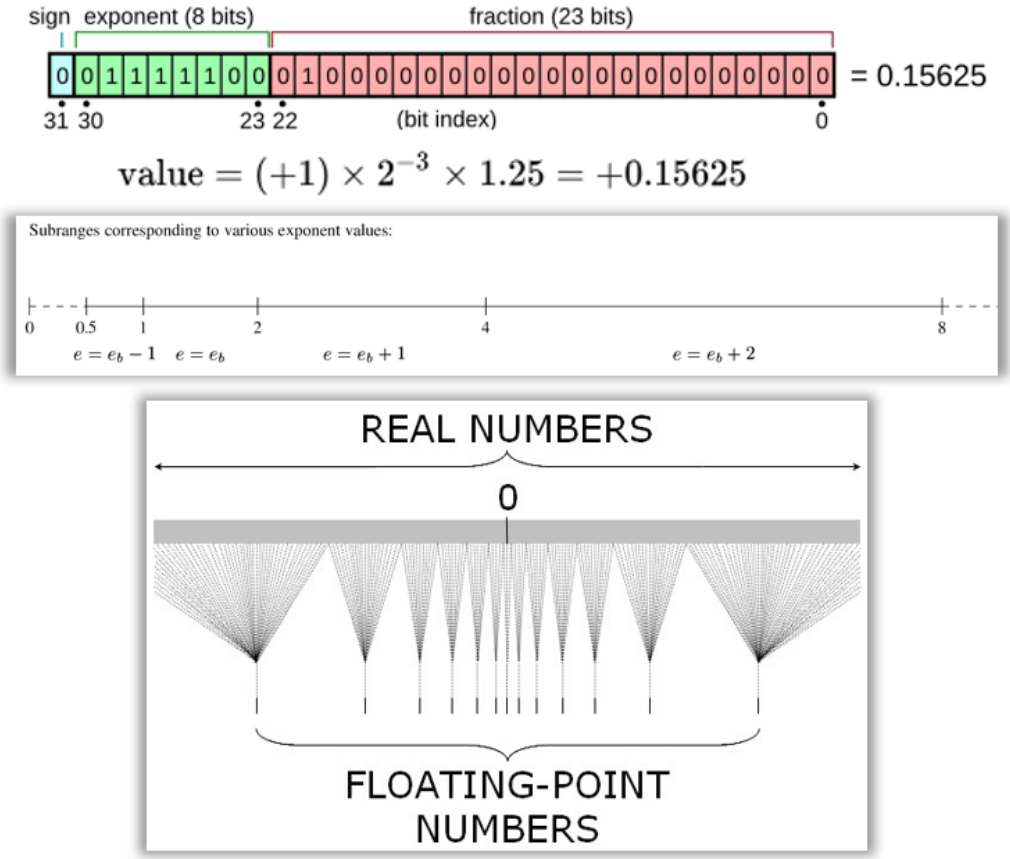


Figure 8: IEEE-754 single precision format above, subranges for different exponent values (in the normal mode). For example, if 8 exponent bits e represent 127, which is the bias e_b of IEEE-754 single precision and the sign $s = 0$, the bit string represents some value in the range $[1, 2)$. In the shown bit string, $e = 124$ and then the number must be in the range $[1/8, 1/4) = [0.125, 0.25)$.

4.1 Modes

A mode refers to a particular usage of the bit fields, which has certain interpretation. In the floating-point formats defined by the IEEE754 standard, there are five modes:

- **normal**: typically the bit string represents a *normalized* number. It means that the significand represents a fraction f ($0 \leq f < 1$), and the decimal value represented by the bit string can be calculated using

$$\tilde{v} = (-1)^s \times (1 + f) \times 2^{e - e_b}, \tag{19}$$

  where s is the value of the sign bit, $e \geq 1$ is the unsigned integer encoded by the exponent part and e_b is the exponent bias[5]. Note the term $(1 + f)$. In the normalized mode, the significand is actually representing this value: '1' corresponds to the so-called *hidden bit* that does not have to be stored.

- **zero**: the zero cannot be represented in the normal mode, and a special encoding is reserved for it, a bit string of zeros (000...0).

- **subnormal**: in the case of subnormals, the exponent field is set to zero, and the decimal value represented can be calculated with

$$\tilde{v} = (-1)^s \times f \times 2^{1 - e_b}. \tag{20}$$

  Note that there is no hidden bit in this case. The subnormal numbers occupy the ranges between zero and the normal mode numbers with the smallest magnitude, $(-2^{1-e_b}, 0)$ and $(0, 2^{1-e_b})$.

- **infinity (Inf)**: In IEEE 754 standard, infinity is encoded by setting all bits of the exponent field to one (111..1) and the significand field to zero.

- **not-a-number (NaN)**: Division by zero, for example, outputs the value in this mode. As in the case of Inf, the exponent field is set 111..1, but the significand field now has a non-zero value.

The essential modes for a custom DSP floating-point format are the normal and zero modes. The subnormal mode can increase the dynamic range significantly, but it increases the complexity of implementation. Inf and NaN are not needed in DSP systems: programmer's job is to take care that related conditions cannot occur.

[5]Recall the biased representation introduced in Sec. 2. The bias is $e_b = 127$ for the single precision format and $e_b = 1023$ for the double precision (excess-127 and excess-1023, resp.).

Example 10. When designing a DSP circuit for floating-point arithmetic we have an option for using a custom format in order to maximize efficiency. For example, we could implement just the normalized mode and zero mode to simplify hardware. For example, let us consider a design where the word length p=12 bits, and five bits are reserved for the exponent. The decision is that if those bits are 00000, then we have the zero mode, and otherwise we have normalized mode. Assume that the bias is $e_b = 15$. Then, the bit string 100110101010 means that the sign bit is $s = 1$, exponent is $e = 00110_2 = 6$, and the significand fraction is $f = .101010_2 = 0.65625$, and the numeric value represented is $\tilde{v} = -1.65625 \times 2^{-9} \approx 0.0032349$. ■

4.2 Range, dynamic range, and precision

In the floating-point format, ulp is the weight of the least significant bit in the significand part and weighting depends on the value of the exponent. Therefore, large numbers are represented with *less precision* than small numbers (Fig. 8). Adjustments of the exponent in floating-point computations optimize for precision, that is, ulp is always kept as small as possible.

What comes to the dynamic range, fixed-point and floating-point representations have dramatic difference for the same native data word width. For example, 32-bit signed fixed point numbers have dynamic range of 187 dB, whereas the IEEE-754 single precision floating point format (also a 32-bit format) has a dynamic range of 1535 dB. As a result, it is certainly easier to manage computations where both large and small values may occur with floating-point numbers.

Let us now consider a custom floating-point hardware, where just the **normal** and **zero** modes are implemented. Characteristics of the floating-point representation are:

- range: the smallest and largest values are $\pm(1 + f_{\max})2^{e_{\max}-e_b}$
- dynamic range :

$$\frac{(1 + \underbrace{f_{\max}}_{\approx 1})2^{e_{\max}-e_b}}{(1 + \underbrace{f_{min}}_{=0})2^{e_{\min}-e_b}} \approx 2^{e_{\max}-e_{min}+1} = 6.02(e_{\max} - e_{\min} + 1) \text{ dB}$$

- precision (ulp) depends on the exponent: the LSB of the significand has the weight 2^{-n}, where n is the number of bits in the significand. Therefore, for the exponent value e, the precision is

$$\text{ulp}(e) = 2^{e-e_b-n}.$$

Example 11. For the format specified in Example 9, $f_{\max} = .111111_2 = 0.984375$ and $e_{\max} = 11111_2 = 31$. the range is $\pm 1.984375 \times 2^{16} = \pm 130048$. As $e_{\min} = 1$ (not 0 as it represents the zero mode), the dynamic range is approximately $6.02 \times (31 - 1 + 1) = 186.62$ dB.

How many bits would be needed for a fixed-point number with at least the same dynamics? For a signed integer with word length p, the dynamic range is $20 \log_{10} 2^{p-1}$ dB. $20 \log_{10} 2^{p-1} = 186.62$ gives $p \approx 31.997$, that is, the word length must be 32.

The difference in word lengths is huge. Note however that there are only 6 bits in the significand of the floating-point number: it cannot represent some real values as accurately as the 32-bit fixed-point number. ■

If the **subnormal** mode is added, the dynamic range becomes larger. The smallest non-zero absolute value for fraction part f is 2^{-n}, where n is the number of bits in the significand (2^{-n} corresponds to $.00...01_2$). The expression for the dynamic range becomes

$$\frac{(1 + \underbrace{f_{\max}}_{\approx 1}) 2^{e_{\max} - e_b}}{2^{-n} 2^{e_{\min} - e_b}} \approx 2^{e_{\max} - e_{min} + 1 + n} = 6.02(e_{\max} - e_{\min} + 1 + n) \text{ dB}$$

Thus, compared to the case without subnormals, the dynamic range is increased by $6.02n$ dB.

Example 12. If the 12-bit format specified in Example 10 is augmented with subnormal numbers, the dynamic range is increased by about 36 dB as $n = 6$. To achieve associated dynamic range with fixed-point numbers, one would need $32 + 6 = 38$ bits. ■

4.3 Fixed-point vs floating-point studies

In general, floating-point arithmetic has been considered to be power hungrier and much less gate efficient than the fixed-point approach. However, this assumption is questionable in applications if low dynamic range of the fixed-point representation has to be compensated by extra instructions.

For example, [Jan+11] compares 16-bit fixed-point and 12-bit floating-point (with 4-bit exponent) implementations of a MIMO detector. These representations have approximately the same dynamic range (about 90 dB) and it is shown experimentally in the paper that the bit error rates (BER) of the implementations are

similar. However, it is observed that the silicon area of the floating-point processor is smaller. In addition, the floating-point processing consumes approximately 17 per cent less energy per detected bit.

Another comparison of fixed-point and floating-point approaches is done by [Nyl+15], who show similarly that the choice of the floating-point approach can lead to reduced silicon area and power dissipation (see Table 2 in the paper). Considering that it is easier to program with the floating-point format as the scaling of numbers is not as much an issue, we conclude that custom floating-point is a viable option in these embedded designs.

However, note that quantized neural network implementations use fixed-point arithmetic. In this case, low dynamic range is not an issue. Tests done with calibration data are used to confirm sufficient performance.

5 More about arithmetic

5.1 Integer arithmetic

The basis for understanding both fixed-point and floating-point arithmetic is to know about integer operations: sign extension, negation, addition, subtraction, arithmetic shift, and multiplication.

Sign extension. Addition of bits to a 1's complement or 2's complement integer representation is done by sign bit extension, that is,

$$\underbrace{a_{p-1}a_{p-1}\cdots a_{p-1}}_{\text{copies of the sign bit}}a_{p-2}...a_1a_0$$

represents the same number as $a_{p-1}a_{p-2}...a_1a_0$. For example, the DSP chip bus is often wider than the ADC resolution, and filling of missing bits can be done in this way (after conversion from offset binary representation to 2's complement).

Negation. Negation of a sign-magnitude number is done by complementing the sign bit. In 1's complement, this is done by complementing all bits. Negation of a 2's complement number is illustrated by the conversion of -21 to 8-bit 2's complement format. The steps are:

1. Determine the binary representation of 21: it is $2^4 + 2^2 + 2^0 = 00010101$;
2. Complement all bits: 11101010;

3. Treat as an unsigned integer and add one: $11101011 = -21$. You may check the result with the formula

$$V_{int} = -a_{p-1}2^{p-1} + \sum_{i=0}^{p-2} a_i 2^i$$

which gives the integer value represented by the 2's complement format.

This works also vice versa: $11101011 \overset{(2)}{\rightarrow} 00010100 \overset{(3)}{\rightarrow} 00010101$.

Addition and subtraction. Basic implementation of binary addition operation uses the ripple-carry approach, which can be speeded up by carry lookahead techniques [Cav85]. The binary subtraction method used in most digital processors adds the 2's complement of the negated subtrahend b to the minuend a. According to the previous example, 2's complement of an integer is its 1's complement plus one. Thus, the 1's complement of b is obtained by bit complementation, and the result is added to a with a carry-in of one at the least significant bit (LSB) position (Fig. 9).

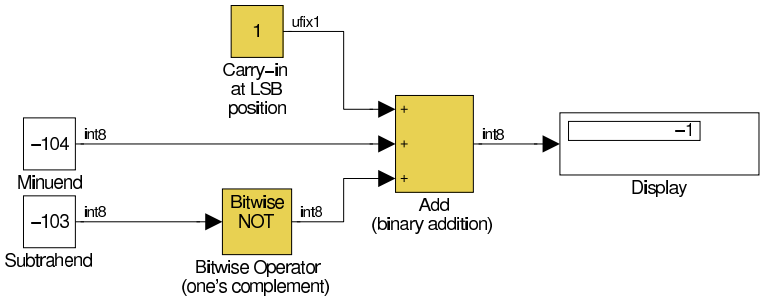


Figure 9: Implementation of binary subtraction (Simulink model).

Arithmetic shift. Multiplying a signed integer by a power of two (2^q) corresponds to the *arithmetic shift* of the number. In arithmetic shift, if $q > 0$, the shift is to the left and for $q < 0$ it is to the right. These operations are illustrated in Fig. 10. Note how sign is extended in the case of the shift to the right. Note also that bits are lost in these operations when the output word length is equal to the input word length.

Multiplication. As discussed in Sec. 2.2, when the multiplicand and multiplier have word lengths equal to p_1 and p_2 bits, the result can always be represented with $(p_1 + p_2)$ bits. This property is referred to as the *law of conservation of bits*

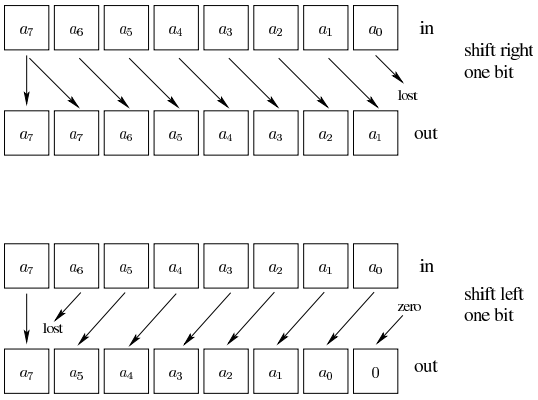


Figure 10: Arithmetic shift to the right and left. To keep the bits shown to be lost, extra bits must be added to the output.

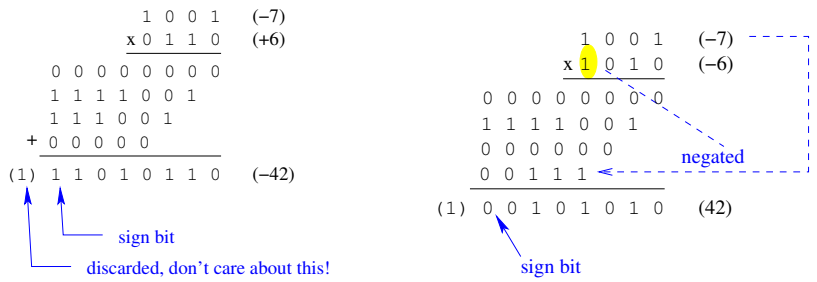


Figure 11: Multiplication by shifting and adding. Note that sign-extended 8-bit multiplicand is 11111001, which is the target of shift operations.

[Lap+97]. Multiplication by paper-and-pencil method illustrates the basic shift-and-add implementation, where we extend the sign of the multiplicand first by p bits and then add arithmetically shifted copies of it. This is straightforward for non-negative multiplier (Fig. 11(a)).

With a negative multiplier, one has to use negations of both multiplier and the result to get the correct result. Another approach is to note that $a_{p-1}a_{p-2}...a_1a_0$ represents the value $v = -a_{p-1}2^{p-1} + \sum_{i=0}^{p-2} a_i 2^i$. Therefore, we can perform ordinary shift-add procedure for other bits than the sign bit of the multiplier. If the sign bit a_{p-1} is one, we add negated multiplicand with weighting 2^{p-1} (Fig. 11(b)). There exists also special logic designs that can handle uniformly both positive and negative multipliers; see [Cav85] for more information.

5.2 Signal quantization (rounding methods)

Addition, multiplication and other arithmetic operations tend to increase the number of bits needed to represent results without loss of precision. It is usually necessary to reduce the precision of the results, for example in order to transfer the results from accumulator registers to memory as in the case of MAC unit. This is called *signal quantization* or *rounding*; recall Sec. 3.4.1, where we discussed *coefficient quantization*, which is performed for filter coefficients at the design time.

Let us denote the step size separating two adjacent fixed-point quantizer output values with Δ. The rounding methods are:

1. **round toward floor (truncation)**[6] simply discards least significant bits, and the output value is always smaller than or equal to the input value. Thus, negative bias is introduced to the signal: the quantization error e is uniformly distributed over the range $-\Delta < e \leq 0$ and thus the expected values $\mathrm{E}\{e\} = -\Delta/2$.

2. **round-to-nearest**: this is the conventional type of rounding that we use in everyday arithmetic. The standard way of implementing it is to add a constant equal to one-half of the least significant bit (LSB), and truncate the result[7]. There is a slight bias problem ($\mathrm{E}\{e\} > 0$) as the values at midpoint of two nearest output values are always rounded up. Typically it is neglected, but it can be troublesome in some applications.

3. **convergent rounding** addresses the bias problem. The rounding of midpoint values depends on the LSB of the value: if it is zero, the number is rounded down, otherwise up. Some processors support this rounding scheme.

4. **magnitude truncation**[8] rounds always towards zero, so the absolute value of the quantizer output is always smaller or equal to the quantizer input. For two's complement integer representation, truncation arithmetic (method 1) can be converted to magnitude truncation by adding one to the result of truncation if it is negative. For positive values, $-\Delta < e \leq 0$ and for negative, $0 \leq e < \Delta$.

Noise introduced by signal quantization is called the *roundoff* noise. Effects of this noise are analyzed by modelling the quantizer as a noise source, a random

[6] Simulink blocks offer also 'round toward ceiling' operation, which rounds in opposite direction and introduces positive bias.

[7] The operation rounds midpoint values toward positive infinity. Matlab offers another rounding mode, where negative midpoint values are rounded toward negative infinity.

[8] Called 'fix' in Matlab.

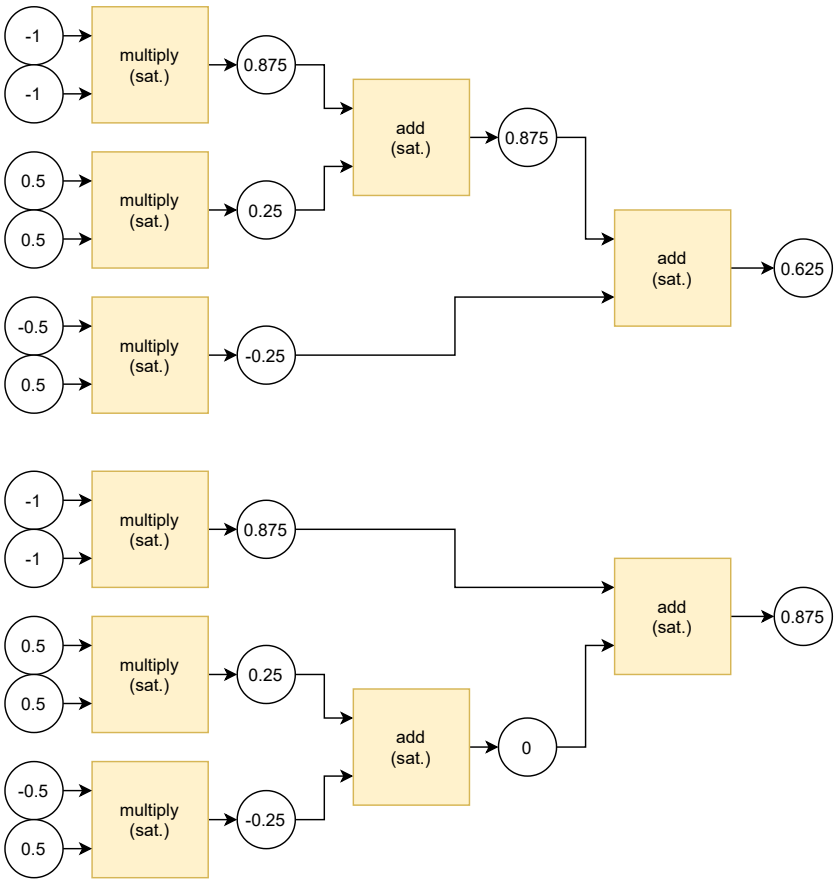
process with some assumptions like uniform distribution, whiteness, wide-sense stationarity, and uncorrelatedness with other signals. The quantization noise gets amplified through the *noise transfer functions*; see [IJ02, p. 419, pp. 836-859].

5.3 A note on saturation arithmetic

Saturating arithmetic, which is a tool for managing overflows, can be applied to multipliers and adders. It must be note that saturating operations are not associative[9], that is, the end results may depend on the order of operations, if saturation occurs. The same holds also for operations involving rounding. Due to this dependence, some standards defining computations may specify exact order in which the operations must be performed. A **bit-exact algorithm** is a numeric algorithm, whose correct implementations provide numerically identical results for a predefined set of test vectors.

Example 11. Let us consider computation of the sum of three multiplications with saturating fixed-point arithmetic and 4-bit word length. In s4.3 format, the minimum and maximum presentable values are -1 and 0.875. The order of additions has effect on the final result:

[9] A binary operation op is associative, if $(a \text{ op } b) \text{ op } c = a \text{ op } (b \text{ op } c)$.



Thus, to have a predictable output for any inputs, one must specify the order of pairwise additions. ■

5.4 Floating-point arithmetic: addition operation

Let us consider addition of two normalized floating-point numbers in order to understand what happens in the floating-point hardware. The operation requires that exponents of the operands are the same. The procedure is:

1. The exponent of the smaller value is increased by shifting the significand to the right. So, difference of the exponents is calculated, which then controls the shifting operation.

2. Addition of signed significands is performed. To do that, the sign-magnitude representation may be mapped to 2's complement representation, and the result is mapped back to sign-magnitude.

3. The result is put to a normalized form by proper rounding and shifting operations. Here, sign-magnitude representation is handy as one can simply count the trailing zeros of the magnitude part and remove them. With 2's complement representation, handling of negative values would be awkward here.

We see that choice of particular forms for representation in the IEEE standard takes into account hardware design. 2's complement was convenient for fixed-point representations, but here we have other issues to take into account.

Example 12. Consider the addition of $A = 76.5$ and $B = -138.75$ represented in IEEE half precision format. They are represented by the bit strings

	sign (s)	exponent (e)	significand (f)
A	0	10101 (21_{10})	0011001000
B	1	10110 (22_{10})	0001010110

Operations are:

1. The exponent of A must be increased by one. Thus, taking into account the hidden bit, the significand of A is .1001100100 after shifting. The significand of B is 1.0001010110. Corresponding exponent is 22_{10}.

2. The sign+significand of A are 00.1001100100 in 2's complement fixed-point format and B is 10.1110101010 (check it!). We add these numbers

$$\begin{array}{r} 00.1001100100 \\ 10.1110101010 \\ \hline 11.1000001110 \end{array}$$

   The result is negative so the sign bit will be 1. Negated result in 2's complement is 00.0111110010, and so the significand is 0.0111110010.

3. There are two zeros on the left, and we must shift by two bits to normalize the result. As a result we get 1.1111001000, and the corresponding exponent is $22 - 2 = 20$.

The result is

	sign (s)	exponent (e)	significand (f)
$A + B$	1	10100 (20_{10})	1111001000

Check: $d = -1 \times (1 + 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-7}) \times 2^{20-15} = -62.25$. ■

References

[Cav85] J. J. F. Cavanagh. *Digital Computer Arithmetic: Design and Implementation*. Singapore: McGraw-Hill, 1985.

[Gol91] D. Goldberg. "What every computer scientist should know about floating-point arithmetic". In: *ACM Computing Surveys* 23 (1991), pp. 5–48.

[IJ02] Emmanuel C. Ifeachor and Barrie W. Jervis. *Digital Signal Processing: a Practical Approach. 2nd edition*. Harlow, England: Pearson Education Ltd., 2002.

[Jan+11] J. Janhunen et al. "Fixed-and floating-point processor comparison for MIMO-OFDM detector". In: *IEEE Journal of Selected Topics in Signal Processing* 5.8 (2011), pp. 1588–1598.

[KG05] Sen M. Kuo and Woon-Seng Gan. *Digital Signal Processors - Architectures, Implementations, and Applications*. New Jersey: Prentice-Hall, 2005.

[Lap+97] P. Lapsley et al. *DSP Processor Fundamentals: Architectures and Features*. Piscataway, NJ: IEEE Press, 1997.

[Nag+21] Markus Nagel et al. *A White Paper on Neural Network Quantization*. 2021. arXiv: `2106.08295 [cs.LG]`.

[Nyl+15] T. Nyländen et al. "Low-power reconfigurable miniature sensor nodes for condition monitoring". In: *International Journal of Parallel Programming* 43 (2015), pp. 3–23.