Posit™ Standard Documentation*
Release 4.12-draft

Posit Working Group

July 21, 2021

*The initial development of posit arithmetic was supported by Singapore's Agency for Science, Technology and Research (A*STAR) and by the DARPA TRADES Program, Contract #HR0011-17-9-0007.

Standard for Posit™ Arithmetic

Sponsor

National Supercomputing Centre (NSCC) Singapore

Abstract

This standard specifies the storage format, operation behavior, and required mathematical functions for posit arithmetic. It describes the binary storage used by the computer and the human-readable character input and output for posit representation. A system that meets this standard is said to be *posit compliant* and will produce results that are identical to those produced by any other posit compliant system. A posit compliant system may be realized using software or hardware or any combination.

Keywords

Arithmetic, binary, exponent, format, fraction, NaR, number rounding, quire, regime

Participants

The following people in the Posit Working Group contributed to the development of this standard:

John Gustafson, *Chair*
Gerd Bohlender
Shin Yee Chung
Vassil Dimitrov
Geoff Jones
Siew Hoon Leong (Cerlane)
Peter Lindstrom
Theodore Omtzigt
Hauke Rehr
Andrew Shewmaker
Isaac Yonemoto

Contents

1 Overview

1.1 Scope

This standard specifies the storage format and mathematical behavior of posit™ numbers, including basic arithmetic operations and the set of functions a posit system must support. It describes how results are to be rounded to a real posit or determined to be a non-real exception.

1.2 Purpose

This standard provides a system for computing with real numbers represented in a computer using fixed-size binary values. Deviations from mathematical behavior (including loss of accuracy) are kept to a minimum while preserving the ability to represent a wide dynamic range. All features are accessible by programming languages; the source program and input data suffice to specify the output exactly on any computer system.

1.3 Inclusions and exclusions

This standard specifies:

- Binary formats for posits, for computation and data interchange
- Addition, subtraction, multiplication, division, dot product, comparison, and other operations
- Composite (fused) functions that are computed exactly, then rounded to posit format
- Mathematical elementary functions such as logarithm, exponential, and trigonometric functions
- Conversions of other number representations to and from posit formats
- Conversions between different posit formats
- Function behavior when an input or output value is not a real number (NaR)

Excluded from the standard are the specific names of the values and operations described here. The lower camelCase naming style is used here, but naming style is excluded from this standard. Implementations may use alternative names and symbols for values and operations that match the behavior described here.[1]

Also excluded are rules for how an implementation should handle and report errors. If a program attempts a posit computation outside the domain that produces a real-valued output, or compares a NaR with a real number, behavior beyond the arithmetic result specified here is up to the implementation designers.

This is a numerical format standard, not a language standard. This standard *enables* a language to provide deterministic rounding as a posit-compliant mode, but the availability of non-deterministic rounding behavior is up to the language designers.

1.4 Requirements vs. recommendations

All descriptions herein are requirements of system behavior, not recommendations. The decision of how to satisfy the requirements and which precisions to support is up to the implementer of this standard, but all functionality must be provided and behave as described for a system to be posit-compliant.

[1] For example, the arc hyperbolic cosine is here shown as **arcCosH**, but it may be called `acosh` in the math library for C so long as it meets this standard's requirement of correct rounding for all inputs. Similarly, a language may express a sum of two posits a and b as $a + b$, though that function is here called **addition**(a, b). Rounding behavior must follow the rules in this document for an implementation to be posit-compliant.

2 Definitions, abbreviations, and acronyms

2.1 Definitions

bit field A contiguous set of bits in a format with a defined meaning. A bit field may extend beyond stored bits $b_{n-1} \cdots b_0$; bits beyond stored bits have value `0`.

exception A special case in the interpretation of the posit format: 0 or NaR.[2]

exponent The power-of-two scaling determined by the exponent bits, in the set $\{0, 1, 2, 3\}$.

exponent bits A two-bit unsigned integer bit field that determines the exponent.

format A set of bit fields and the definition of their meaning.

fraction The value represented by the fraction bits; $0 \leq \text{fraction} < 1$.

fraction bits The bit field after the exponent bits.

fused Rounded only after an exact evaluation of an expression involving more than one operation.

implicit value A value added to the fraction based on the sign: -2 for negative posits, 1 for positive posits. Zero and NaR do not have an implicit value.

LSB The least significant bit of a format or a bit field within a format.

maxPos The largest positive value expressible as a posit. It is a function of n.

minPos The smallest positive value expressible as a posit. It is a function of n.

MSB The most significant bit of a format or a bit field within a format.

NaR Not a real. A value that is not mathematically definable as a unique real number.

n The number of bits in a posit format. It can be any positive integer 2 or greater.

pIntMax The largest consecutive integer expressible as a posit. It is a function of n.

posit A real number representable using the format described in this standard, or NaR.

precision The total storage size for expressing any number format, in bits. For a posit, precision is ***n*** bits.

quire A real number representable using the format described in this standard, or NaR. It can represent exact sums and differences of products of posits and has precision $16n$.

quire sum limit The minimum number of posit additions or subtractions that can overflow the quire.

regime The power-of-16 scaling determined by the regime bits. It is a signed integer.

regime bits A posit bit field after the MSB that uses a form of run-length encoding (as opposed to positional notation) to represent a signed integer.

rounded Converted from a real number to a posit value, according to the rules of this standard.

sign The value 1 for positive numbers, -1 for negative numbers, and 0 for 0. NaR has no sign.

sign bit The MSB of a posit or quire bit field.

significand The implicit value plus the fraction; $-2 \leq \text{significand} < -1$ for negative posits, and $1 \leq \text{significand} < 2$ for positive posits.

[2] Posit exceptions do not imply a need to check status flags or handle heavyweight OS or language runtime exceptions.

3 Posit and quire formats

3.1 Overview

3.1.1 Formats

This section defines posit and quire formats, which are used to represent a finite set of real numbers or the exception value NaR. Formats are specified by their precision, n. There is a quire format of precision $16n$ that is used to contain exact sums of products of posits of precision n. Dynamic range and accuracy are determined solely by n. This standard describes example choices for n like 8, 16, and 32. The posit type label is "posit" with the decimal string for n appended. The corresponding quire type label is "quire" with the decimal string for n appended, even though its format has $16n$ bits.[3]

3.1.2 Compliance

An implementation is compliant with this standard if it supports full functionality of at least one precision. If the implementation supports more than one precision, then it must support conversions between them.

3.1.3 Represented data

A posit is NaR or a real number x of the form $k \times 2^m$, where k and m are integers limited to a range symmetric about and including zero. The smallest positive posit, *minPos*, is 2^{-4n+8} and the largest positive posit, *maxPos*, is $1/\mathit{minPos}$, or 2^{4n-8}. Every posit is an integer multiple of *minPos*. Every real number maps to a unique posit representation; there are no redundant representations. Posits can express all integers i in a range $-\mathit{pIntMax} \leq i \leq \mathit{pIntMax}$. Outside that range, integers exist that cannot be expressed as a posit without rounding to a different integer; *pIntMax* is $\lceil 2^{\lfloor 4(n+2)/5 \rfloor - 4} \rceil$.

A quire is NaR or an integer multiple of the square of *minPos*, represented as a 2's complement binary number with $16n$ bits. The quire can represent the exact summation of products of two posits at least $2^{31}-1$ (approximately two billion) times without the possibility of rounding or overflow.[4]

The properties of example and general posit precisions are summarized in Table 1:

Property	**posit8**	**posit16**	**posit32**	**posit**n
fraction length	0 to 3 bits	0 to 11 bits	0 to 27 bits	0 to $\max(0, n-5)$ bits
minPos	$2^{-24} \approx 6.0 \times 10^{-8}$	$2^{-56} \approx 1.4 \times 10^{-17}$	$2^{-120} \approx 7.5 \times 10^{-37}$	2^{-4n+8}
maxPos	$2^{24} \approx 1.7 \times 10^{7}$	$2^{56} \approx 7.2 \times 10^{16}$	$2^{120} \approx 1.3 \times 10^{36}$	2^{4n-8}
pIntMax	$16 = 2^4$	$1024 = 2^{10}$	$8388608 = 2^{13}$	$\lceil 2^{\lfloor 4(n+2)/5 \rfloor - 4} \rceil$
quire precision	128 bits	256 bits	512 bits	$16n$ bits
quire sum limit	$2^{55} \approx 3.6 \times 10^{16}$	$2^{87} \approx 1.5 \times 10^{26}$	$2^{151} \approx 2.9 \times 10^{45}$	2^{23+4n}

Table 1: Properties of posit formats

3.2 Binary interchange format encoding

3.2.1 Posit format encoding

Figure 1 defines the general format for posit encoding, while Figure 2 shows the extreme case where posits are at their smallest or largest magnitude. The four bit fields are:

1. Sign bit S.
2. Regime bit field R consisting of r bits identical to R_0, terminated either by $1 - R_0$ ($r + 1$ bits total length) as shown in Figure 1, or by the LSB of the posit (r bits total length) as shown in Figure 2.

[3] For example, $n = 3$ types are posit3 and quire3, $n = 64$ types are posit64 and quire64, etc.

[4] The product of two posits in precision n is always exactly expressible in a posit of precision $2n$, but the quire obviates such temporary doubling of precision when computing sums and differences of products. It is safe to form exact sums of posits with much more than 2^{31} terms, per the *quire sum limit*.

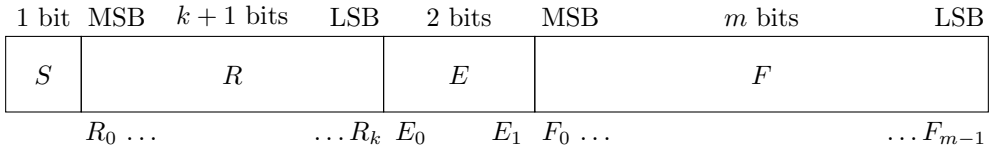

Figure 1: General binary posit format

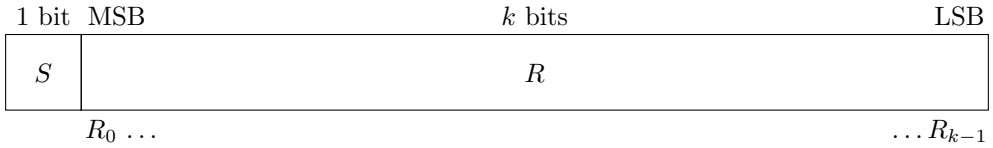

Figure 2: Binary posit format with exponent and fraction fields truncated

3. The exponent bit field E has length 2 bits, but one or both bits may be after the LSB of the posit and thus have value `0`.

4. The fraction bit field F has length $n-5$ bits, but 0 to $n-5$ of those bits may be after the LSB of the posit and thus have value `0`. The number of stored bits is m.

The meaning of each bit field is as follows:

1. S represents an integer s, its literal value, 0 or 1. The *implicit value* is $(1-3s)$.

2. R represents an integer $r = -k$ if R_0 is `0`, or $r = k-1$ if R_0 is `1`.

3. E represents an integer e, its bits treated as a 2-bit unsigned integer. *e.* $0 \le e \le 3$.

4. F represents a fraction f, an m-bit unsigned integer divided by 2^m. $0 \le f < 1$.

The posit value p is inferred from the bit fields (S, R, E, F) as follows:

1. If $S = 0$ and $R = 1-n$ (all other fields contain only `0` bits), then $p = 0$.

2. If $S = 1$ and $R = 1-n$ (all other fields contain only `0` bits), then p is NaR.

3. Otherwise, $p = ((1-3s)+f) \times 2^{(1-2s)\times(4r+e+s)}$.

3.2.2 Quire format encoding

Quire format is a fixed-point 2's complement format of length $16n$, with fields as follows:

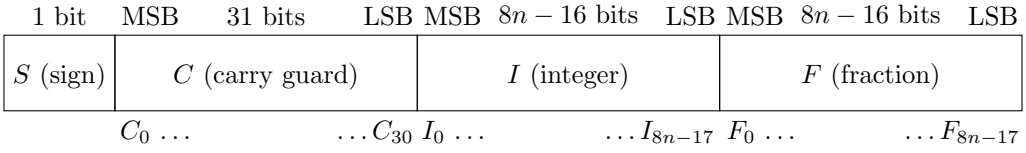

Figure 3: Binary quire format

The quire value q is inferred from the bit fields (S, C, I, F) as follows:

1. If S is `1` and all other fields contain only `0` bits, then q is NaR.

2. Otherwise q is 2^{16-8n} times the 2's complement signed integer represented by all bits concatenated.

4 Rounding

4.1 Definition and Method

Rounding is the substitution of a posit for any real number. Operation results are regarded as exact prior to rounding. The method for rounding a real value x is described by the following algorithm:

Data: x, a real number
Result: Rounded x
if x is exactly expressible as a posit **then**
  return x
else if $|x| > maxPos$ **then**
  return $sign(x) \times maxPos$
else if $|x| < minPos$ **then**
  return $sign(x) \times minPos$
else
  Let u and w be n-bit posits such that $u < x < w$ and open interval (u, w) contains no n-bit posit.
  Let U be the n-bit string associated with u.
  Let v be the $(n+1)$-bit posit associated with the $(n+1)$-bit string U`1`.
  if $u < x < v$ **or** (LSB of U is `0` **and** $x = v$) **then**
    return u
  else
    return w
  end
end

4.2 Fused Expressions

A *fused expression* is an expression with two or more operations that is evaluated exactly before rounding to a posit. Expressions that can be written in the form of a dot product of vectors of length less than 2^{31} can be evaluated exactly in the quire and then rounded to posit format to create a fused expression, if so defined by the rules of the language.[5] Fused expressions (such as fused multiply-add and fused multiply-subtract) need not be performed with the quire to be compliant.

4.3 Program Execution Restrictions

For any language where the order of operations is well-defined, the execution order of operations in posit-compliant mode cannot be changed from that expressed in the source code if it affects rounding. This includes any use of precisions or operation fusing not expressed in the source code. Languages that offer optimization modes that covertly change rounding do so at the cost of bitwise-reproducible results and are non-compliant when used in those modes.

If a language permits mixed data types in expressions, including quires and posits, or posits and other formats for representing real numbers, the language must specify how such expressions are evaluated to be posit-compliant.

[5] If a fused expression is computed in parallel, sufficient intermediate result information must be communicated that the result is identical to the single-processor result. Note that functions in Section 5 which are rounded and have two or more operations in their mathematical definition are fused expressions, such as **rSqrt**, **expMinus1**, and **hypot**.

5 Operations

5.1 Guiding principles for NaR

If an operation produces real-valued output, any NaR input produces NaR output, with the exception of **next** and **prior**. NaR is output when the function is not arbitrarily close to a unique real number for open neighborhoods of complex values sufficiently close to the input values[6], except for functions in Section 5.2.1. A test of NaR equal to NaR returns True. NaR has no sign, so **sign**(NaR) returns NaR.

5.2 Mathematical functions

The following functions shall be supported, with rounding per Section 4. Functions that take more than one posit input must have the same precision for all inputs, and any posit output is in the same precision as the inputs. Conversion routines may be used to make mixed-precision inputs the same precision, per Section 6.1. Conversions may be explicit in source code or implicit by language rules.

5.2.1 Basic functions of one posit argument

negate(*posit*) returns $-$*posit*.[7]
abs(*posit*) returns **negate**(*posit*) if *posit*<0, else *posit*.
sign(*posit*) returns the posit representing 1 if *posit* > 0, -1 if *posit* < 0, or 0 if *posit* $= 0$.
round(*posit*) returns the integer-valued posit nearest to *posit*, and the nearest even integer-valued posit if two integers are equally near.
ceil(*posit*) returns the smallest integer-valued posit greater than or equal to *posit*.
floor(*posit*) returns the largest integer-valued posit less than or equal to *posit*.
next(*posit*) returns the posit represented by an increment of the *posit* bit string.[8]
prior(*posit*) returns the posit represented by a decrement of the *posit* bit string.[9]

5.2.2 Arithmetic functions of two posit arguments

addition(*posit1*, *posit2*) returns *posit1* $+$ *posit2*, rounded.
subtraction(*posit1*, *posit2*) returns *posit1* $-$ *posit2*, rounded.
multiplication(*posit1*, *posit2*) returns *posit1* $\times$ *posit2*, rounded.
division(*posit1*, *posit2*) returns *posit1*/*posit2*, rounded.

5.2.3 Comparison functions of two posit arguments

All comparison functions return Boolean values identical to comparisons of the posit bit strings regarded as 2's complement integers, so there is no need for separate machine-level instructions. The value NaR has the bit string of the most negative integer, so **compareLess**(NaR, *posit*) returns True if *posit* is real.

compareEqual(*posit1*, *posit2*) returns True if *posit1* $=$ *posit2*, else False.
compareNotEqual(*posit1*, *posit2*) returns True if *posit1* $\neq$ *posit2*, else False.
compareGreater(*posit1*, *posit2*) returns True if *posit1* $>$ *posit2*, else False.
compareGreaterEqual(*posit1*, *posit2*) returns True if *posit1* $\geq$ *posit2*, else False.
compareLess(*posit1*, *posit2*) returns True if *posit1* $<$ *posit2*, else False.
compareLessEqual(*posit1*, *posit2*) returns True if *posit1* $\leq$ *posit2*, else False.

[6] The function may be complex-valued in the neighborhood of the inputs, but still have a real-valued limit. For example, **pow**$(-1,-3) = (-1)^{-3}$ is -1 even though the function is complex-valued in any neighborhood of the second argument. Similarly, **sqrt**$(0) = 0$.

[7] This is the 2's complement of the posit bit string. 2's complement does not affect 0 or NaR, since they are unsigned.

[8] This means that **next**(*maxPos*) is NaR, and **next**(NaR) is $-$*maxPos*. 2's complement integer overflow is ignored.

[9] This means that **prior**($-$*maxPos*) is NaR, and **prior**(NaR) is *maxPos*. 2's complement integer underflow is ignored.

5.2.4 Elementary functions of one posit argument

sqrt(*posit*) returns $\sqrt{posit}$, rounded.
rSqrt(*posit*) returns $1/\sqrt{posit}$, rounded.
exp(*posit*) returns e^{posit}, rounded.
expMinus1(*posit*) returns $e^{posit} - 1$, rounded.
exp2(*posit*) returns 2^{posit}, rounded.
exp2Minus1(*posit*) returns $2^{posit} - 1$, rounded.
exp10(*posit*) returns 10^{posit}, rounded.
exp10Minus1(*posit*) returns $10^{posit} - 1$, rounded.
log(*posit*) returns $\log_e(posit)$, rounded.
logPlus1(*posit*) returns $\log_e(posit + 1)$, rounded.
log2(*posit*) returns $\log_2(posit)$, rounded.
log2Plus1(*posit*) returns $\log_2(posit + 1)$, rounded.
log10(*posit*) returns $\log_{10}(posit)$, rounded.
log10Plus1(*posit*) returns $\log_{10}(posit + 1)$, rounded.
sin(*posit*) returns $\sin(posit)$, rounded.
sinPi(*posit*) returns $\sin(\pi \times posit)$, rounded.
cos(*posit*) returns $\cos(posit)$, rounded.
cosPi(*posit*) returns $\cos(\pi \times posit)$, rounded.
tan(*posit*) returns $\tan(posit)$, rounded.
tanPi(*posit*) returns $\tan(\pi \times posit)$, rounded.
arcSin(*posit*) returns $\arcsin(posit)$, rounded. $\mathbf{round}(-\pi/2) \le \mathbf{arcSin} \le \mathbf{round}(\pi/2)$.
arcSinPi(*posit*) returns $\arcsin(posit) / \pi$, rounded. $-1/2 \le \mathbf{arcSinPi} \le 1/2$.
arcCos(*posit*) returns $\arccos(posit)$, rounded. $0 \le \mathbf{arcCos} \le \mathbf{round}(\pi)$.
arcCosPi(*posit*) returns $\arccos(posit) / \pi$, rounded. $0 \le \mathbf{arcCosPi} \le 1$.
arcTan(*posit*) returns $\arctan(posit)$, rounded. $\mathbf{round}(-\pi/2) \le \mathbf{arcTan} \le \mathbf{round}(\pi/2)$.
arcTanPi(*posit*) returns $\arctan(posit) / \pi$, rounded. $-1/2 \le \mathbf{arcTanPi} \le 1/2$.
sinH(*posit*) returns $(e^{posit} - e^{-posit})/2$, rounded.
cosH(*posit*) returns $(e^{posit} + e^{-posit})/2$, rounded.
tanH(*posit*) returns $(e^{posit} - e^{-posit})/(e^{posit} + e^{-posit})$, rounded.
arcSinH(*posit*) returns $\text{arcsinh}(posit)$, rounded.
arcCosH(*posit*) returns $\text{arccosh}(posit)$, rounded. $0 \le \mathbf{arcCosH}$.
arcTanH(*posit*) returns $\text{arctanh}(posit)$, rounded.

5.2.5 Elementary functions of two posit arguments

hypot(*posit1*, *posit2*) returns $\sqrt{posit1^2 + posit2^2}$, rounded.
pow(*posit1*, *posit2*) returns $posit1^{posit2}$, rounded.[10]
arcTan2(*posit1*, *posit2*) returns the argument t of *posit1* + i *posit2*, $-\pi < t \le \pi$, rounded.[11]
arcTan2Pi(*posit1*, *posit2*) returns **arcTan2**(*posit1*, *posit2*)$/\pi$, rounded.

5.2.6 Functions of three posit arguments

fMM(*posit1*, *posit2*, *posit3*) returns *posit1* $\times$ *posit2* $\times$ *posit3*, rounded.[12]

5.2.7 Functions of one posit argument and one integer argument

compound(*posit*, *integer*) returns $(1 + posit)^{integer}$, rounded.
rootN(*posit*, *integer*) returns $posit^{1/integer}$, rounded. If *integer* is even, $\mathbf{rootN} \ge 0$.

[10] See Section 5.1 for situations that generate NaR. For example, x^y is not arbitrarily close to a single real number for any complex-valued neighborhoods of $x = y = 0$, so $\mathbf{pow}(0, 0)$ is NaR.

[11] The apparent discontinuity in $\text{arcTan2}(x, y)$ for $x \le 0, y = 0$ is spurious since it results from jumping between "branches" of a multivalued function. It should return $\mathbf{round}(\pi)$ if $x < 0$, $y = 0$. arcTan2(0,0) must be NaR since the function is not arbitrarily close to a unique real value for any complex-valued open neighborhoods of the inputs.

[12] Because multiplication is commutative and associative, any permutation of the inputs will return the same rounded result.

5.3 Functions not yet required for compliance

Special functions such as error functions, Bessel functions, gamma and digamma functions, beta and zeta functions, etc. are not presently required for a system to be posit compliant. They may be required in a future revision of this standard.

5.4 Functions that do not round correctly for all arguments

Computing environments that support versions of functions in Section 5.2 that do not round correctly for all inputs must supply the source code for such functions, and use a notation for them that is distinct from the notation for the function that rounds correctly for all inputs.

5.5 Functions involving quire arguments

With the exception of **qToP** which returns a posit format result, these functions return a quire format result.[13] If any quire operation overflows the carry bits of a quire, the result is NaR in quire format.

pToQ(*posit*) returns *posit* converted to quire format.
qNegate(*quire*) returns $-$*quire*.
qAbs(*quire*) returns **qNegate**(*quire*) if *quire* < 0, else *quire*.
qAddP(*quire*, *posit*) returns *quire* $+$ *posit*.
qSubP(*quire*, *posit*) returns *quire* $-$ *posit*.
qAddQ(*quire1*, *quire2*) returns *quire1* $+$ *quire2*.
qSubQ(*quire1*, *quire2*) returns *quire1* $-$ *quire2*.
qMulAdd(*quire*, *posit1*, *posit2*) returns *quire* $+$ (*posit1* $\times$ *posit2*).
qMulSub(*quire*, *posit1*, *posit2*) returns *quire* $-$ (*posit1* $\times$ *posit2*).

qToP(*quire*) returns *quire* rounded to posit format per Section 4.1.

Other functions of quire values may be provided through software in the source code, but are not required for compliance. They may be required in a future revision of this standard.

[13] These quire functions can be used to compute the real and imaginary parts of complex number products, exact sums up to length 2^{23+4n}, exact dot products and scaled sums of vectors up to length $2^{31}-1$, exact determinants of 2-by-2 matrices, exact discriminants of quadratic equations, exact residuals of solutions to systems of linear equations, and higher-precision arithmetic for addition, subtraction, multiplication, division, and square root of values expressed as un-evaluated sums of posit value lists. The posit obtained by rounding a quire can be subtracted from that quire and the quire again converted to a second posit, and that process repeated to produce tuples representing higher precision.

6 Conversion operations for posit format

6.1 Conversions between different precisions

Converting a posit format to higher precision is exact, by appending `0` bits. Conversion to a lower precision is rounded, per Section 4.[14] In the function notation used here,

p*m***To***n*(*posit*) returns the n-bit posit form of an m-bit posit *posit* by these conversion rules.

6.2 Quire conversions

A posit compliant system needs only to support rounding from quire to posit and conversion of posit to quire in the matching posit precision, per Section 5.5.

6.3 Conversions between posit format and decimal character strings

Table 2 shows examples of the minimum number of significant decimals needed to express a posit such that the real number represented by the decimal form will round to the same posit.

Precision	**posit8**	**posit16**	**posit32**	**posit64**
Decimals	2	5	10	21

Table 2: Examples of minimum decimals in a base-ten significand to preserve posit value

6.4 Conversions between posit format and integer format

Supported posit sizes must provide conversion to and from all integer sizes supported in a computing environment. In converting a posit to an integer, if the posit is out of integer range after rounding or is NaR, the integer is returned that has its MSB = `1` and all other bits `0`. In converting an integer to a posit, the integer with its MSB = `1` and all other bits `0` converts to NaR; otherwise, the integer is rounded, per Section 4.

6.5 Conversions between posit format and IEEE 754™ Standard float format

Supported posit sizes must provide conversions to and from the IEEE 754 Standard float formats that are supported in the computing environment, if any. In converting a posit to an IEEE 754 float of any type, posit zero converts to the "positive zero" float, and NaR converts to quiet NaN. Otherwise, the posit value is converted to a float per the float rounding mode in use. In converting a float to a posit, all forms of infinity and NaN convert to NaR. Otherwise, the real number represented by the float is rounded, per Section 4. The "negative zero" and "positive zero" floats convert to posit zero. □

[14] Note that precision conversion does not require decoding a posit into its bit fields.