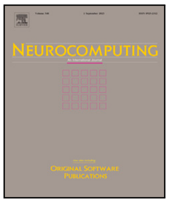
Neurocomputing 597 (2024) 127903

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom



Posit and floating-point based Izhikevich neuron: A Comparison of arithmetic

T. Fernandez-Hart [a,*], James C. Knight [b], T. Kalganova [a]

[a] Brunel University, CEDPS, Department of Electronic and Computer Engineering, Uxbridge, UB8 3PH, UK
[b] University of Sussex, School of Engineering and Informatics, Brighton, BN1 9QJ, UK

ARTICLE INFO

Communicated by Q. Zheng

Keywords:
Floating-point arithmetic
Posit arithmetic
Spiking neural network
Izhikevich neuron model

ABSTRACT

Reduced precision number formats have become increasingly popular in various fields of computational science, as they offer the potential to enhance energy efficiency, reduce silicon area, and improve processing speed. However, this is often at the expense of introducing arithmetic errors that can impact the accuracy of a system. The optimal balance must be struck, judiciously choosing a number format using as few bits as possible, while minimising accuracy loss.

In this study, we examine one such format, posit arithmetic as a replacement for floating-point when conducting spiking neuron simulations, specifically using the Izhikevich neuron model. This model is capable of simulating complex neural firing behaviours, 20 of which were originally identified by Izhikevich and are used in this study. We compare the accuracy, spike count, and spike timing of the two arithmetic systems at different bit-depths against a 64-bit floating-point gold-standard. Additionally, we test a rescaled set of Izhikevich equations to mitigate against arithmetic errors by taking advantage of posit arithmetic's tapered accuracy.

Our findings indicate that there is no difference in performance between 32-bit posit, 32-bit floating-point, and our 64-bit reference for all but one of the tested firing types. However, at 16-bit, both arithmetic systems diverge from the 64-bit reference, albeit in different ways. For example, 16-bit posit demonstrates an $18\times$ improvement in accumulated spike timing error over a 1000ms simulation compared to 16-bit floating-point when simulating regular (tonic) spiking. This finding holds particular importance given the prevalence of this particular firing type in specific regions of the brain. Furthermore, when we rescale the neuron equations, this error is eliminated altogether. Although current Posit Arithmetic Units are no smaller than Floating Point Units of the same bit-width, our results demonstrate that 64-bit floating-point can be replaced with 16-bit posit which could enable significant area savings in future systems.

1. Introduction

Spiking Neural Networks (SNNs) are often considered the next generation of Artificial Neural Network (ANNs) [1]. They are a closer approximation of biological neural networks than current state-of-the-art ANNs such as convolutional neural networks and operate by simulating the dynamics of individual neurons. Unlike ANNs, where neurons are usually continuously active, SNNs exhibit sparse activity in time with neurons only generating brief output ‘spikes’ when they receives sufficient input. These spikes serve as the means of communication between neurons and facilitate the transfer of information within the SNN. Each spike is considered identical and information is encoded only in the time at which they occur [2]. This sparsity means that SNNs share many of the benefits of their biological counterparts such as low power and noise robustness, and are an active area of research [3,4]. While not yet mainstream, SNNs have found success in a variety of applications such as object recognition [5], robotic control [6] and even ChatGPT-like large language models [7].

Typically, SNNs use first-order differential equations to describe each neuron and many sets of equations are available, varying in their level of complexity and biological correctness. The simplest are the phenomenological Integrate-and-Fire (IF) family [8] and, at the other end of the spectrum are biophysically accurate neuron models like the Hodgkin–Huxley (HH) model [9]. In the present study, we use the Izhikevich model, which is less complex than the HH, but still able to replicate more neuron firing behaviours than simpler IF models [10]. Furthermore, Tamura et al. [11] demonstrated that small changes can have a large impact on the Izhikevich model. Their bifurcation analysis of the Izhikevich system showed that, a change in the parameter b

* Corresponding author.
E-mail addresses: Tim.Fernandez-Hart@brunel.ac.uk (T. Fernandez-Hart), J.C.Knight@sussex.ac.uk (J.C. Knight), Tatiana.Kalganova@brunel.ac.uk (T. Kalganova).

https://doi.org/10.1016/j.neucom.2024.127903
Received 26 June 2023; Received in revised form 15 April 2024; Accepted 19 May 2024
Available online 31 May 2024

by as little as 0.07 can change a stable system into a chaotic one making it an interesting testbed for numerical investigations of this sort. Once a neuron model is chosen for the SNN, a numerical simulation proceeds by tracking the evolution of each neuron's state variables through time. With the Izhikevich model, each neuron has a threshold, which if it is crossed, constitutes spike emission, followed by some reset conditions [12].

However, the combination of these complex dynamics and spike-based communication leads to challenges running SNN simulations efficiently on a CPU or GPU [13]. Some Application-Specific Integrated Circuit (ASIC) hardware is available such as the SpiNNaker and Loihi neuromorphic systems, but these are research orientated and not widely available [14,15]. Field Programmable Gate Arrays (FPGA) [16] are another option, but their capacity is limited compared to ASIC, which restricts network size (although they can be linked into multi-board systems, such as [17]). Hence, considerable research has focused on reducing the hardware cost of these networks, while maintaining accuracy in the spike timing and underlying neuron equations. Specifically, there is growing research interest in reduced precision number formats as a system optimisation technique [18,19]. The potential benefits to doing so are twofold. Firstly, reducing hardware complexity can free up resources that can be redeployed to increase model complexity. Secondly, reduced bit-depth increases the speed of computation, which in turn, can reduce energy requirements and allow long running dynamics to be simulated.

Posit numbers represent one exciting form of reduced precision number format that promise the accuracy of Floating-Point (FP) while using fewer bits. Posits were designed to be hardware-friendly and some studies suggest that Posit Arithmetic Units (PAUs) are simpler to implement in hardware than a Floating-Point Unit (FPU) – requiring less silicon and enabling faster, smaller arithmetic units within processors [20–22]. However, these savings are often disputed [23,24] with other studies showing similar hardware and energy usage for an FPU and PAU of a given bit-depth [21] and others showing PAUs to be larger [25–28]. While a few of these studies compare complete systems with integrated FPUs and PAUs [26,27], many others only compare specific operations or use non IEEE-754 compliant FPUs. Furthermore, new research is continuing to reduce the hardware requirements of PAUs [22,29,30].

To the best of our knowledge, only one study has been conducted into the potential use of posit arithmetic for SNN implementations. Silva et al. [31] demonstrated that a single, posit-based Izhikevich neuron was able to replicate the dynamics of 20 different firing types that were originally identified by Izhikevich in his earlier work [12]. However, there are no published studies that explore the impact of reduced precision, posit arithmetic on SNN accuracy and here, we seek to do just this.

We compare the accuracy of equivalently sized posit arithmetic with single precision (32-bit) and half precision (16-bit) FP arithmetic when simulating an Izhikevich neuron while also investigating whether rescaling of the standard Izhikevich equations, to make better use of posit arithmetic's tapered accuracy, is a successful strategy to minimise accuracy loss. The results show that it is possible to simulate tonic firing with 16-bit posit arithmetic while incurring no loss of accuracy, compared to a 64-bit floating-point version. This is an important result, given the common usage of this firing type [32], and the abundance of biological neurons exhibiting these behaviours in certain brain areas (90%–95% of rat basal ganglia [33,34]).

The rest of this paper is organised as follows. Section 1.2 discusses the two arithmetics and gives an example, illustrating their differences. This includes a short introduction to decimal accuracy, which is a measure of arithmetic precision for a given value. Section 2 outlines the methods used, including all formulas and parameter values for both standard and rescaled equation types. Section 3 covers the main results of this study as well as a discussion of possible explanations for the differences found. This section also examines the impact of different scaling factors and posit configurations. Then, we briefly explain some of the issues with using reduced-precision fixed-point arithmetic to simulate Izhikevich neurons in Section 3.4 and bfloat16 in Section 3.5. A wider discussion of the hardware implications is given in Section 3.8. Finally, Section 4 proposes the main conclusions and suggests future research directions.

1.1. Contributions to science

- We not only show that posit arithmetic is capable of matching floating-point at reduced precision, but we quantify the arithmetic errors for both number systems for all 20 firing patterns of the Izhikevich neuron model and we show *when* posits outperform floating-point numbers.
- We demonstrate that rescaling the standard Izhikevich equations can minimise arithmetic errors when using reduced precision floating-point and posit arithmetic and that, with our rescaling, regular spiking (Tonic Spiking) can be simulated using 16-bit posit arithmetic with *no loss of accuracy* compared to 64-bit floating point.
- We investigate the choice of n, es posit parameters as well as the scaling factor and provide deeper insight into how posit arithmetic behaves in these simulations.

1.2. Background

Currently, the predominant format for representing real numbers within a computer is Floating-Point (FP), which is ratified in the IEEE-754 standard [35]. As shown in Fig. 1, floating point numbers represent numbers using sign, exponent and fraction bits. This arrangement has a large dynamic range and it is the most widely used number format in scientific computing. However, it is rather a complex standard, having two representations of zero, permitting 'subnormal' numbers, supporting multiple rounding modes and reserving a great many bit patterns for Not-a-Number (NaN). These increase the format's hardware cost, due to the need to verify all these special conditions during run time.

Posit arithmetic is designed to be a hardware friendly, drop-in replacement for FP although, as it was only introduced in 2017, it is still in its infancy [20]. As shown in Fig. 1, posits have sign and fraction bits like FP. However, unlike FP, the fraction is scaled by two values – the exponent and the regime – rather than just the exponent as in FP. The regime bits are unique to posits, and act like a super-exponent, scaling the exponent and thus the fraction. From a hardware perspective, the main drawback of posit arithmetic is the dynamic size of the regime bits, which alter the location of the exponent and fraction bits. Thus, posit decoding entails extra overhead encoding and decoding values compared to FP, required to determine the boundaries of each of these sections, before any further operations can be performed. However, this overhead could be potentially cancelled out by the less intricate posit standard which only defines a single rounding mode (compared to five for IEEE-754 FP) and two exception values: zero and NaR (Not-a-Real, equivalent to Not-a-Number and representative of $\pm\infty$). This not only reduces costly condition checking but additionally, liberates bit patterns to represent values, which in turn, gives posit arithmetic a higher precision for a given number of bits compared to FP [20].

A posit format is defined by two numbers posit$\langle n, es \rangle$ where n is the total number of bits and es, the maximum number of those bits used for the exponent. The first bit of a posit is always a sign bit, which is immediately followed by the regime bits. The regime bits encode the value r in a run of zeros, or ones, terminated by the opposite bit. A run of ones encodes a positive number, and a run of zeros a negative number, as defined by [36], where k is the number of ones or zeros before the terminating bit. The regime bits are not present in the floating-point standard and are of variable length. In the extreme case, they can expand to fill the whole word, leaving no space for the

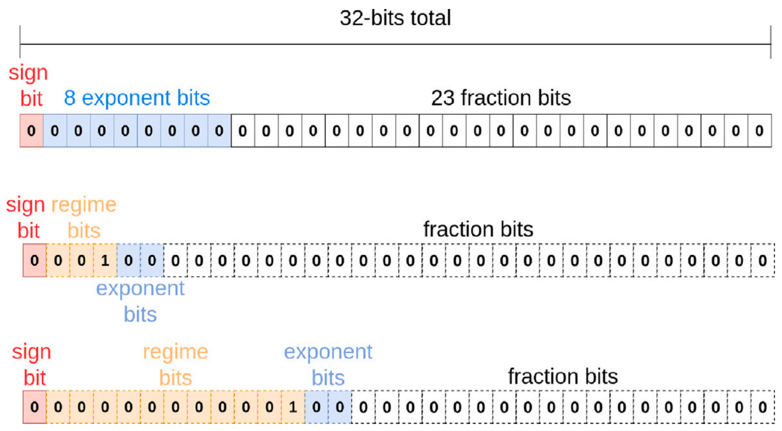


Fig. 1. A comparison of 32-bit floating-point (top) and 32-bit posit (middle and bottom) number structures. 32-bit floating-point structure as defined in the IEEE-754 standard. 1 sign bit, 8 exponent bits and 23 fraction bits. A posit$<32,2>$ is defined as 1 sign bit, at least 2 regime bits (where k is the run of bits before the terminating bit), a maximum of 2 exponent bits and the remaining bits represent the fraction. The bottom diagram illustrates what happens when the regime expands. A posit loses fraction bits first, but can also lose one or all of its exponent bits too. The dashed lines represent the dynamically positioned bits.

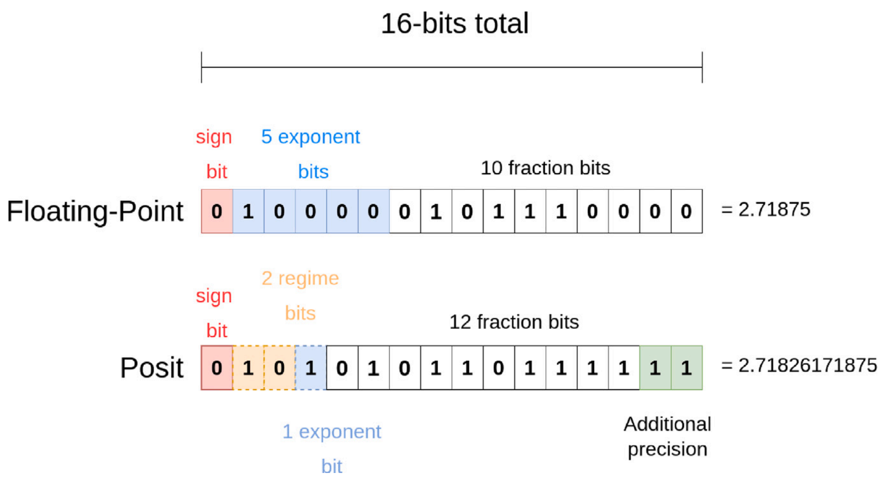


Fig. 2. 16-bit example of Euler's number represented in FP16 and Posit16 arithmetic. The additional 2 fractional bits have been highlighted in green.

exponent or any part of the fraction. Following the regime bits are the exponent bits. Unlike floating-point, the exponent bits do not have a bias. Lastly, we have the fraction bits. These contain an implicit one and function in the same way as in floating point. A number p is decoded using Eq. (1). The value useed is calculated by useed $= 2^{2^{es}}$ (see [20] for more explanation).

$$p = (-1)^s \text{useed}^r 2^e \left(1 + \frac{f}{2^{\text{fs}}}\right) \tag{1}$$

$$r = \begin{cases} -k & \text{if } R_0 = 0 \\ (k-1) & \text{if } R_0 = 1 \end{cases} \tag{2}$$

where s is the sign bit, e is the unbiased exponent value, f is the fraction and fs is the fraction length. R_0 is the first regime bit, if it is 0, it represents a run of 0's terminated with a 1, and if it is 1, it represents a run of 1's terminated with a 0.

Importantly, posits do not underflow or overflow. Instead, they exhibit tapered precision, centred around 0 and greatest in the interval $(-1, 1)$. This has implications for their accuracy. For example, trying to represent Euler's number $e = 2.7182818284590452354$ with 16-bit Floating-Point (FP16) and 16-bit posit$<16,1>$ (posit16) results in the bit patterns shown in Fig. 2. The benefit of dynamically positioned bits means that, here, the posit's regime bits are at their shortest (2 bits minimum) and, when combined with fewer exponent bits, leaves more bits to represent the fraction. These additional fraction bits can be seen to improve the accuracy of the representation.

Decoding Fig. 2 into actual values gives us a posit16 value of 2.71826171875 (Eq. (4)).

$$p = (-1)^0 \times 4^0 \times 2^1 \times \left(1 + \frac{1471}{2^{12}}\right) \tag{3}$$

$$= 2.71826171875 \tag{4}$$

If we compare this result with the FP16 version of e, which has 2 fewer fraction bits, we can see the effect on the accuracy of its representation. FP16 achieves $e = 2.71875$.

$$fp = (-1)^s \times 2^{e-bias} \times \left(1 + \frac{fraction}{2^{10}}\right) \tag{5}$$

$$fp = (-1)^0 \times 2^{16-15} \times \left(1 + \frac{368}{2^{10}}\right) \tag{6}$$

$$= 2.71875 \tag{7}$$

One method to quantify this difference is to calculate the decimal accuracy between the exact representation of a number (x_{exact}) and its representation in the arithmetic being evaluated (x_{repr}):

$$\text{decimal accuracy} = -\log_{10}\left|\log_{10}\left(\frac{x_{\text{repr}}}{x_{\text{exact}}}\right)\right|. \tag{8}$$

This quantifies the number of correct decimal places following rounding [18,20]. Using Euler's number as an example gives a decimal precision of 5.49 for posit$<16,1>$ and 4.13 for FP16, illustrating the advantage of posit's tapered accuracy.

One additional aspect of the posit standard is the *quire* register. This is a large ($16n$) fixed-point register based on the FP Kulisch accumulator [36] and used for minimising rounding and overflow errors during a series of operations. Although, due to its size, it can occupy approximately half the total area of a posit arithmetic unit [27], it has garnered interest in the deep learning community, for minimising dot product and sum errors [37].

2. Methods

We used single-precision (FP32) and half-precision (FP16) floating-point numbers, along with the posit$<32,2>$ (posit32) and posit$<16,1>$ (posit16) formats to run simulations of an Izhikevich neuron over 1000 ms using both standard equations and rescaled equations. Although formally $es = 2$ for all n as defined by the Posit Working Group [36], the previous draft standard defined a 16-bit posit as posit$<16,1>$. Hence, for comparison with previous works [24,30,38,39], unless otherwise stated, posit16 is posit$<16,1>$.

We do not use the quire register in this study as our focus is on using posit arithmetic for reduced precision neural simulations and, unlike deep-learning, this application does not involve long chains of multiply-accumulate operations. As such, we do not anticipate the inclusion of the quire will have a large impact on our results [40]. Moreover, the sheer size of the quire register means that it is often excluded from practical hardware implementation [41].

Following the approach employed by several previous studies [18,42], we used a software implementation to study the errors in numeric simulations of reduced precision FP and posit arithmetic. Although emulation does not give a measure of speed, it does inform us about accuracy of the arithmetic. To compare the membrane voltage of our simulations against a 64-bit floating point baseline, we measure the Normalised Root-Mean-Square Error (NRMSE) of each test. Spike counts and their accumulated timing error were also recorded. We also tested bfloat16, different scaling factors and a variety of posit configurations.

2.1. Model definitions

2.1.1. Standard neuron model

In 2003, Izhikevich presented a neuron model capable of complex spiking patterns, similar to those found in nature (e.g. intrinsic bursting, chattering, low-threshold spiking, fast spiking) but with a reduced computational load in comparison to other models [12]. It consists of a pair of coupled differential equations, modelling the evolution of the membrane voltage driven by an input current (I):

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{9}$$

and of a recovery variable.

$$u' = a(bv - u) \tag{10}$$

Once the membrane voltage crosses a 'threshold voltage' (typically 30 mV), a spike is emitted and predefined reset conditions are applied:

$$v \leftarrow c \tag{11}$$

$$u \leftarrow u + d \tag{12}$$

The behaviour of the model is controlled by four dimensionless variables a, b, c, d which correspond to different model properties [10] and are typically set to the values listed in Table 1 in order to reproduce one of the 20 identified behaviours (See Izhikevich [12]). Three firing patterns (Class 1, Integrator and Accommodation) also require altering Eqs. (9) and (10) as described by Izhikevich [43]. Given its phenomenological fidelity with nature, coupled with its computational efficiency the Izhikevich model is widely used in many SNN implementations and has been used to compare arithmetics in previous studies [44,45]. Hence, it was also selected as the neuron type in this study.

2.1.2. Rescaled neuron model

In addition to the standard Izhikevich model, we developed a rescaled version so that the standard operating range of the state variables would fit within the region where posits are most accurate. This transforms Eq. (9) into Eq. (13). Eqs. (11) and (12) which describe the reset behaviour are unchanged, but the c and d parameters as well as the input current, spike threshold and initial value of u and v, are all rescaled by a factor of 0.01.

$$v' = 4v^2 + 5v + 1.4 - u + I \tag{13}$$

2.2. Testing procedure

We compared IEEE-754 FP32, FP16, posit$<32,2>$ (posit32) and posit$<16,1>$ (posit16), against a 64-bit floating-point (FP64) ground-truth for all 20 firing types outlined above, using both standard and rescaled equations.

Several posit hardware implementations have been proposed [25,27] including full processor designs based on RISC-V CVA-6 [27] and rocket chip cores [41] which can be synthesised and run on an FPGA. Although, prefabricated ASIC hardware supporting posits is not yet widely available, several open-source software simulation libraries are available with the SoftPosit library being the most prominent. Here, we use this library to test the accuracy of posit arithmetic and compare it against other varieties [24,46] and the Berkeley SoftFloat library [47] (via Python wrapper [48]) to simulate FP32 and FP16 formats. We ran our simulations in Python 3.8, on a 64-bit machine using wrappers for SoftPosit and SoftFloat (developed by [46] and [48] respectively) and used the native floating-point type for the reference FP64 simulations. Simulations were run for 1000 ms using the Forward Euler method of numerical integration with time step sizes laid out in Table 1.

Although fast, one limitation with the SoftPosit library is its restrictions in defining a posit's structure. For example, `posit16(1.23)` defines the value 1.23 with a posit$<16,1>$ structure. This configuration adheres to earlier draft standards released by The Posit Working Group. However, recently the standard was updated and now defines 16-bit posits as posit$<16,2>$ [36]. To compare posit$<16,2>$ with posit$<16,1>$ and FP64, we employed a third simulation library, the Universal Numbers Library (UNL) [49]. To verify the consistency of results between SoftPosit and UNL, the NRMSE of all firing types was tested for posit$<16,1>$ and was zero for all firing types (results not shown).

UNL also allows emulation of Google's bFloat16 number format [50]. bFloat16 was introduced to ease storage, communication volumes and computational load of ANNs. It has the same dynamic range as FP32 and does not need any hyperparameter tuning as required by the IEEE-754 FP16 format [51].

2.3. Data analysis

For each test, we recorded the voltage (mV) every time step and, if the spike threshold was crossed, the time step at which this occurred (ms). These recorded values were used to calculate: the membrane voltage error, spike count error, and where the spike count was the same, the accumulated spike timing error.

2.3.1. Membrane voltage error

In line with similar studies [52,53], membrane voltage error was quantified using the Normalised Root Mean Square Error (NRMSE):

$$\mathrm{RMSE} = \sqrt{\frac{\Sigma_{t=1}^{n}(v_{\mathrm{aut}}(t) - v_{\mathrm{fp64}}(t))^2}{n}} \tag{14}$$

$$\mathrm{NRMSE} = \frac{\mathrm{RMSE}}{(v_{\max} - v_{\min})} \tag{15}$$

where v_{aut} is the membrane voltage of the *arithmetic under test*; v_{fp64} is the reference membrane voltage simulated using 64-bit floating-point

Table 1
Izhikevich parameters and initial conditions to produce 20 different firing types. Additionally, for firing types Class 1 and Integrator, Eq. (9) becomes $v' = 0.04v^2 + 4.1v + 108 - u + I$ and for Accommodation, Eq. (10) becomes $u' = a(b(v + 65))$ [12].

Firing type	a	b	c	d	dt	initial u	initial v
Tonic spiking	0.02	0.2	−65.0	6.0	0.25	$b \times v$	−70
Phasic spiking	0.02	0.25	−65.0	6.0	0.25	$b \times v$	−64
Tonic bursting	0.02	0.2	−50.0	2.0	0.25	$b \times v$	−70
Phasic bursting	0.02	0.25	−55.0	0.05	0.2	$b \times v$	−64
Mixed mode	0.02	0.2	−55.0	4.0	0.25	$b \times v$	−70
SFA	0.01	0.2	−65.0	8.0	0.25	$b \times v$	−70
Class 1	0.02	−0.1	−55.0	6.0	0.25	$b \times v$	−60
Class 2	0.2	0.26	−65.0	0.0	0.25	$b \times v$	−64
Spike Latency	0.02	0.2	−65.0	6.0	0.2	$b \times v$	−70
Subthreshold oscillation	0.05	0.26	−60.0	0.0	0.25	$b \times v$	−62
Resonator	0.1	0.26	−60.0	−1.0	0.25	$b \times v$	−62
Integrator	0.02	−0.1	−55.0	6.0	0.25	$b \times v$	−60
Rebound spike	0.03	0.25	−60.0	4.0	0.2	$b \times v$	−64
Rebound Burst	0.03	0.25	−52.0	0.0	0.2	$b \times v$	−64
Threshold Variability	0.03	0.25	−60.0	4.0	0.25	$b \times v$	−64
Bistability	0.1	0.26	−60.0	0.0	0.25	$b \times v$	−61
DAP	1.0	0.2	−60.0	−21.0	0.1	$b \times v$	−70
Accomodation	0.02	1.0	−55.0	4.0	0.5	−16	−65
Inhibition induced spiking	−0.02	−1.0	−60.0	8.0	0.5	$b \times v$	−63.8
Inhibition induced bursting	−0.02	−1.0	−45.0	0.0	0.5	$b \times v$	−63.8

arithmetic at time t; and $v_{\max}$ and $v_{\min}$ are the maximum and minimal values of v_{fp64}. NRMSE quantifies the error between the FP64 voltage reference and the arithmetic under test at each time step during each simulation. Normalising the error allows errors to be directly compared between standardised and rescaled equation types.

2.3.2. Spike count and timing

In order to assess the effect of arithmetic error on spike count and timing, we also counted the number of spikes and their absolute accumulated lag/lead times. First, we determined whether each test produced the expected number of spikes (n_{spike}). If it did, we then looked at the timing of each spike (S_i^{aut}), relative to the reference FP64 simulation (S_i^{fp64}):

$$T = \sum_{i}^{n_{\mathrm{spike}}} |S_i^{\mathrm{fp64}} - S_i^{\mathrm{aut}}| \tag{16}$$

where T is the sum of absolute timing difference per spike, across the whole simulation. This method differs slightly from Hopkins et al. [45], in that the final reported result is the accumulated lead/lag, as opposed to the difference for each spike. The main reason for this is brevity as we tested all 20 Izhikevich neuron behaviours whereas Hopkins et al. [45], only tested a few firing types. Moreover, accumulating the timing errors will still allow a comparison between experiments.

3. Results and discussion

3.1. Membrane voltage error

First, we present the NRMSE results for the different arithmetics, bit depths and equation types in Table 3; Figs. 3 and 4 show the 32-bit and 16-bit results respectively.

Considering the standard Izhikevich equations, posit32 had a lower NRMSE than FP32 in every case with the exception of ‘Class 2’. Since the ‘Class 2’ error was much greater than the other errors and it was lower for FP32 it skewed the overall mean which was 0.0115 for FP32 and 0.0119 for posit32. Indeed, with ‘Class 2’ removed from the calculation, the overall mean for FP32 was 2.58×10^{-5} and 3.97×10^{-6} for posit32, which is an error reduction of over 6×. A broadly similar pattern was noted for the rescaled equations. Posit32 had a lower NRMSE in 19 of the 20 firing types. Only ‘Class 2’, had a lower error in FP32. ‘Bistability’ did not demonstrate the expected cessation of firing on its second input spike. This behaviour is very sensitive to timing, and posit32 was not able to demonstrate it. However, with these outliers removed, the overall mean NRMSE for FP32 was 2.32×10^{-5} and 3.93×10^{-6} for posit32, which is an error reduction of 6.8× in the firing types with expected dynamics.

The standard Izhikevich equations for all firing types need to represent a minimum value of −81.8 and a maximum of 30. Since FP32 is a fixed format, it always uses 23 fractional bits to express these values, whereas posit32 will use 27 fractional bits for values around 30, and 23 fractional bits to represent −81.8. Hence, when using the standard equations, posit32 has *at least* the same number of fractional bits as FP32, resulting in additional precision across most of the numerical range encountered in these simulations. Table 2 gives a summary of the number of fractional bits, available for each arithmetic at these values and their decimal accuracy.

Next we consider 16-bit arithmetic. Aside from the ‘Class 2’ Izhikevich neurons – which were equally problematic at all precisions and equation types – the NRMSE was far higher for 16-bit arithmetic compared to 32-bit. Fig. 4 clearly shows that some firing types such as ‘Class 1’ (which has an NRMSE close to 0.2 for all arithmetics) accrue more error than others such as ‘Spike Latency’, ‘Subthreshold’ and ‘Resonator’ whose NRMSE is below 0.05. This pattern broadly correlates with the number of spikes emitted in a simulation, shown by the dotted black line in Fig. 4. However, there were exceptions to this. Most notably, the error was far greater for FP16 with ‘Rebound Burst’ (standard equations). This was due to the neuron continuing to fire regularly following the expected burst of spikes. This is in contrast to the posit16 test which failed to fire the expected burst at all.

Using the standard equations, posit16 had a lower NRMSE in 11 of the 20 firing types. This is reflected in the mean error which is slightly lower for posit16 (0.075) than FP16 (0.0799). Unlike the 32-bit tests, there were no experiments where one arithmetic or firing type was orders of magnitude different from the rest, requiring additional consideration. Rescaling improved the error for FP16 in only 5 firing types and slightly *increases* the mean error. However, with posit16, rescaling resulted in an improvement in 13 firing types compared to the standard equations, and a reduction in the mean error. This clearly demonstrates the use of rescaling to improve simulation accuracy when using reduced precision posit arithmetic. One possible explanation for this is the increase in the number of available fractional bits following rescaling. For example, posit16 has 10 fractional bits to represent the value 30, the same as FP16. However, when we use the rescaled equations, the value 30 becomes 0.3 where posit16 uses 12 fractional bits. Again, it is likely that it is these additional fractional bits that give posit16 the advantage over FP16. A similar situation can be seen for the lower bound number −81.8. At this value, posit16 has only 9 fractional bits compared to 10 used in FP16. However, in the rescaled equations,

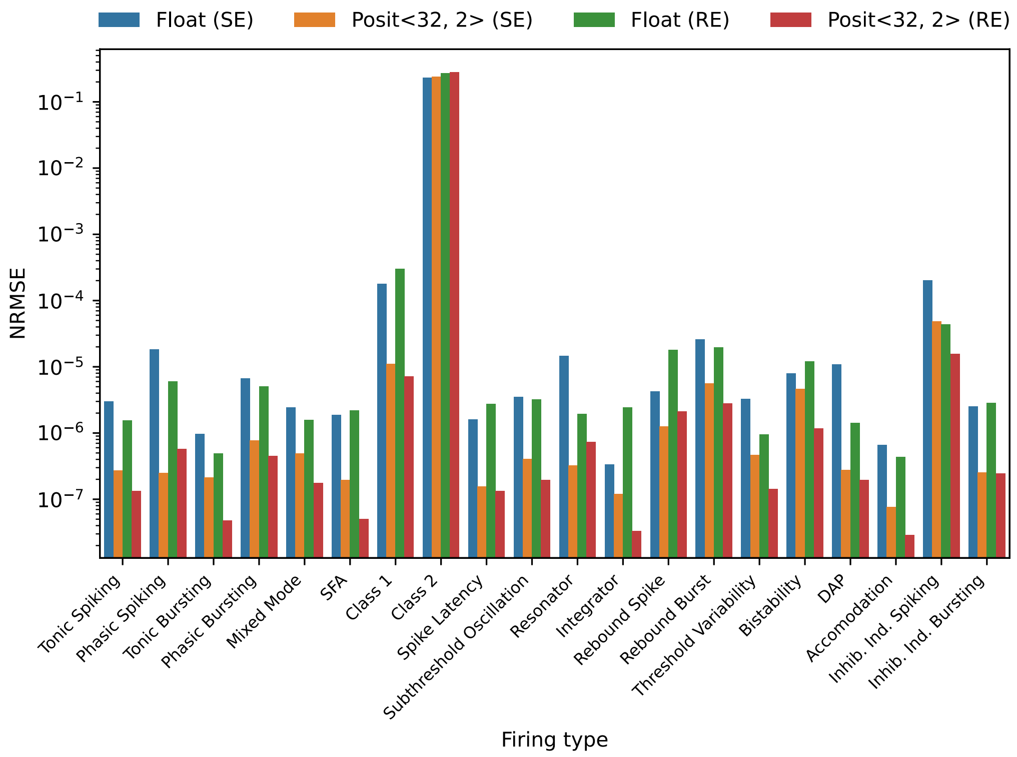


Fig. 3. 32-bit floating-point and posit arithmetics using both the Standard Equations (SE) and Rescaled Equations (RE). Where Inhib. Ind. Spiking is Inhibition Induced Spiking and Inhib. Ind. Bursting is Inhibition Induced Bursting. NRMSE is displayed on a log scale to account for the magnitude of the 'Class 2' error.

Table 2
A table showing the number of bits used to represent the fraction for each arithmetic type and equation. The values were chosen as the minimum and maximum values seen across all firing types. 16-bit posit is posit<16, 1>. Decimal accuracy shows the correct number of decimal places after rounding for each value.

Equation type	Value	Number of bits for fraction		Decimal accuracy	
		FP	Posit	FP	Posit
Standard 32-bit	30	23	26	Exact	Exact
	0	23	0 (exception)	Exact	Exact
	−81.8	23	26	7.79	8.99
Rescaled 32-bit	0.3	23	27	7.76	8.97
	0	23	0 (exception)	Exact	Exact
	−0.818	23	27	8.01	10.80
Standard 16-bit	30	10	10	Exact	Exact
	0	10	0 (exception)	Exact	Exact
	−81.8	10	9	4.18	3.58
Rescaled 16-bit	0.3	10	12	4.15	4.75
	0	10	0 (exception)	Exact	Exact
	−0.818	10	12	4.16	5.44

-81.8 becomes -0.818 and posit16 provides 12 fractional bits compared to FP16's 10.

As well as NRMSE, we can also calculate decimal precision for each arithmetic and equation type. Using the standard equations the threshold value is 30, which is exactly representable in all arithmetics and bit depths. However, in the rescaled equations, the threshold becomes 0.3 where FP32 has 23 fractional bits and a decimal precision of 7.76 digits, whereas posit32 has 26 fractional bits and a decimal precision of 8.97, resulting in more than one correct additional decimal place. Using reduced 16-bit arithmetic, FP16 has 10 bits for the fraction and achieves a decimal precision of 4.15, whereas posit16 has 12 fractional bits and a decimal accuracy of 4.75.

Table 2 compares the number of bits available for each test to represent the fraction for three values. These values were the minimum and maximum seen over all 20 tests for each equation type, plus zero. It shows that when using 32-bits, posit arithmetic has 3 more bits to represent each fraction, giving it a higher accuracy. However, when using 16-bits, FP16 and posit16 both have 10 bits to represent the value 30, but posit16 has only 9 bits to represent -81.6, compared to 10 bits for FP16. This could account for why fewer posit tests could reproduce the correct number of spikes in 16-bit tests using the standard equations. However, overall the NRMSE was very similar to FP. In contrast, when comparing the fraction bits available for the rescaled values, we can see that posit32 has 4 more bits of precision than FP32 and posit16 has 2 more bits than FP16. Thus posit arithmetic always has a higher accuracy when using the rescaled equations.

Indeed, rescaling the equations reduced the NRMSE of posit16-based simulations in 13 out of the 20 firing types – dramatically in some cases. This is in contrast to the FP16 results, where only 5 firing types had an improved NRMSE under the rescaled equations – a likely consequence of almost constant decimal accuracy. 'Tonic Spiking' in particular appears to benefit from using the rescaled equations with posit16 arithmetic.

To illustrate this disparity, Fig. 5 shows the FP16 simulation of 'Tonic spiking' using standard equations. If we compare this to the

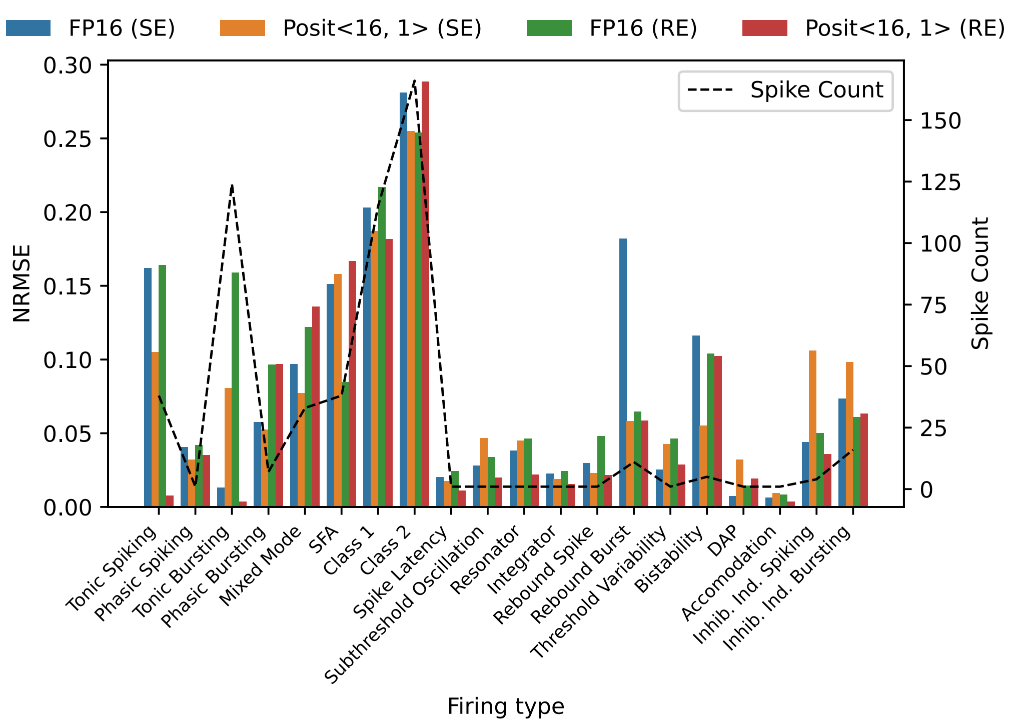


Fig. 4. 16-bit floating-point and posit arithmetics using both Standard Equations (SE) and Rescaled Equations (RE). Where Inhib. Ind. Spiking is Inhibition Induced Spiking and Inhib. Ind. Bursting is Inhibition Induced Bursting. Spike count for each simulation is overlaid with its axis on the right-hand side.

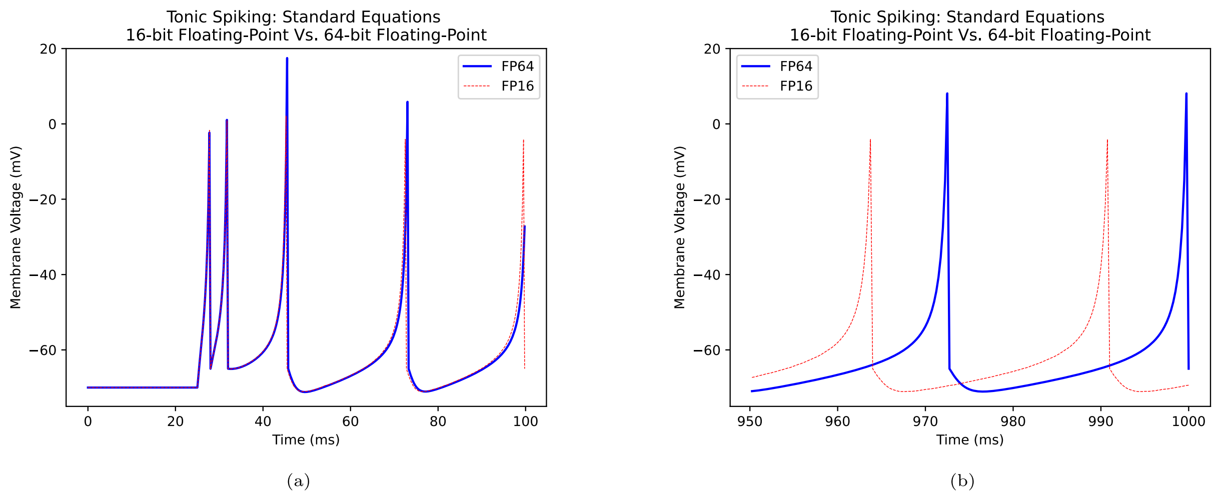


Fig. 5. A comparison of FP16 and FP64 arithmetic simulating tonic spiking over 1000 ms. (a) shows the first 100 ms of the simulation and (b) the final 50 ms. Initially in (a), it can be seen that there is good alignment between the FP16 with the FP64 reference. However, as the simulation proceeds, the lag – caused by accumulated arithmetic error – becomes clearly visible in (b).

posit16 simulation shown in Fig. 6, then the advantages of this arithmetic become apparent. If we again focus on the final 50 ms, while there is a slight error, it is much reduced compared to the error at the same point when using floating-point arithmetic (Fig. 6(b)).

Finally, as shown in Fig. 7, this error reduces to effectively zero for the posit16 simulation when we use the rescaled equations — with only a small difference in voltage amplitude separating the two simulations. However, as shown in Fig. 7(a), rescaling does not reduce the FP16 error.

3.2. Class 2

Fig. 8 shows the ‘Class 2’ firing type using FP32 and standard equations. Izhikevich defines ‘Class 2’ as neuron types which are either quiescent or have a relatively large firing rate [12]. Although Class 2 has the greatest NRMSE, it demonstrates normal firing, and the high NRMSE are likely to be related to the high spike count, each of which lags or leads the FP64 reference. However, if we consider the spike count, Class 2 is very similar to the FP64 reference.

3.3. Spike count and timing

In Table 4 we show the spike count and, if we first consider 32-bit arithmetic, we can see that when using standard equations, posit32 deviated in spike count from the reference only once for the ‘Class 2’ firing type. FP32 did not have this error and fired the correct number of times. While this change in spike number should be a cause for concern and we will discuss it in the next section, we should also consider

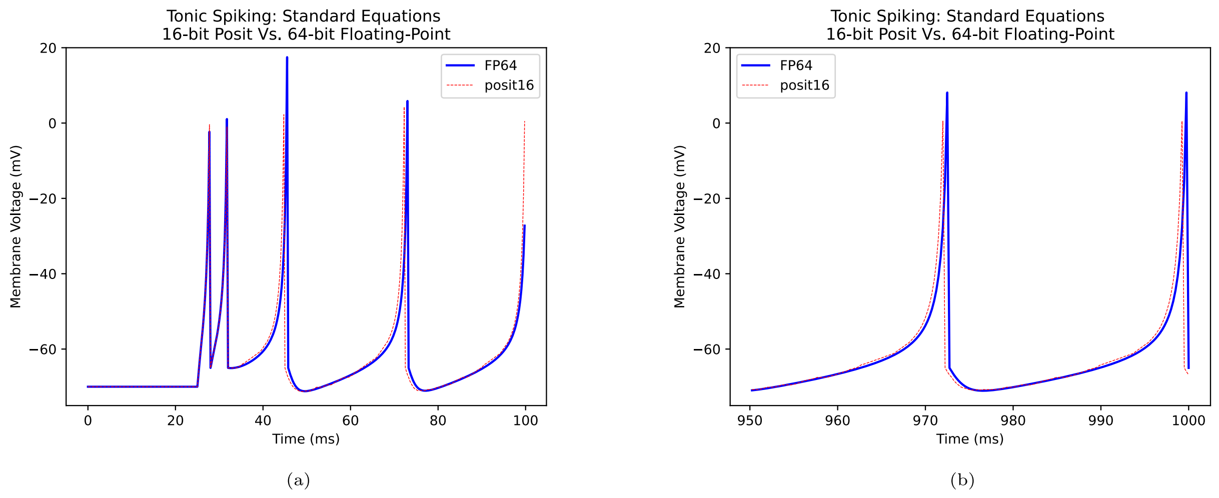


Fig. 6. A comparison of posit16 and FP64 arithmetic simulating tonic spiking over 1000 ms. (a) shows the first 100 ms of the simulation and (b) the final 50 ms. Very little spike timing error can be seen in either (a) or (b) meaning posit16 arithmetic has introduced very few errors during the simulation.

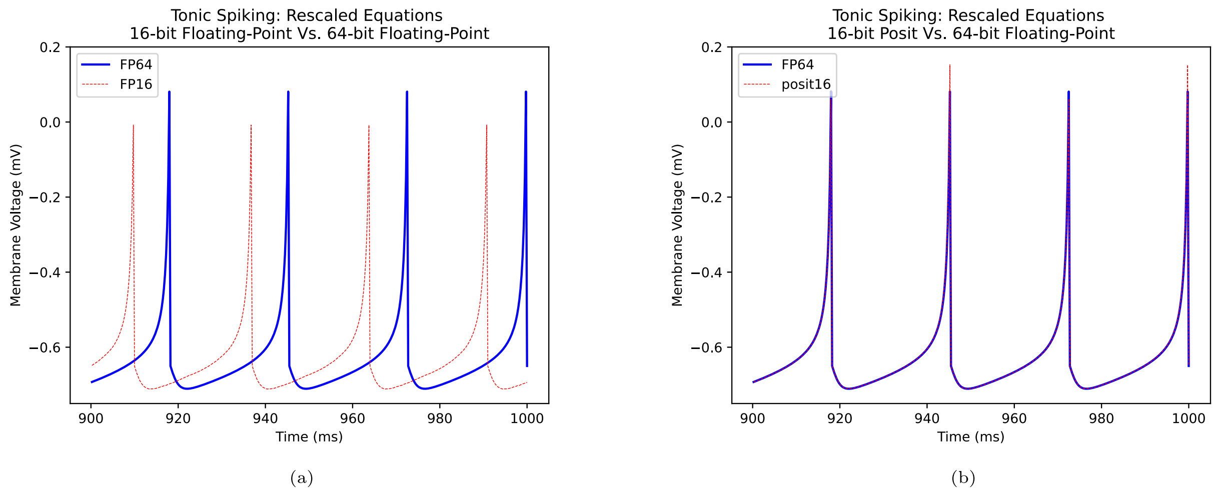
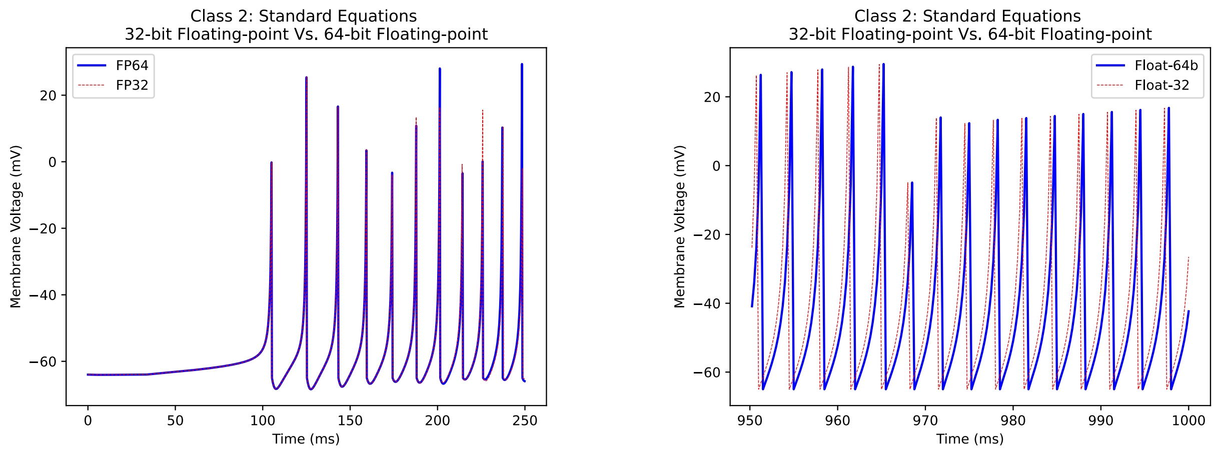


Fig. 7. Both (a) and (b) show the last 50 ms of simulation, where (a) employs FP16 and (b) uses posit16 arithmetic when using the rescaled equations. (a) shows that rescaling does not improve the accuracy in the FP16 case. However, (b) demonstrates the elimination of timing errors in posit16 when using rescaled equations.

Table 3
NRMSE for each firing pattern, arithmetic and equation type. The mean of each arithmetic and equation type is given in the bottom row.

Firing type	Standard - NRMSE				Rescaled - NRMSE			
	32-bit		16-bit		32-bit		16-bit	
	Float	Posit	Float	Posit	Float	Posit	Float	Posit
Tonic Spiking	3.01E−06	2.73E−07	1.62E−01	1.05E−01	1.55E−06	1.34E−07	1.64E−01	7.66E−03
Phasic spiking	1.84E−05	2.51E−07	4.05E−02	3.20E−02	6.00E−06	5.76E−07	4.18E−02	3.50E−02
Tonic bursting	9.69E−07	2.13E−07	1.32E−02	8.05E−02	4.93E−07	4.82E−08	1.59E−01	3.42E−03
Phasic bursting	6.74E−06	7.80E−07	5.75E−02	5.24E−02	5.04E−06	4.53E−07	9.64E−02	9.68E−02
Mixed mode	2.43E−06	4.91E−07	9.68E−02	7.73E−02	1.58E−06	1.76E−07	1.22E−01	1.36E−01
SFA	1.87E−06	1.95E−07	1.51E−01	1.58E−01	2.21E−06	5.01E−08	8.46E−02	1.67E−01
Class 1	1.79E−04	1.11E−05	2.03E−01	1.87E−01	3.03E−04	7.22E−06	2.17E−01	1.82E−01
Class 2	2.30E−01	2.39E−01	2.81E−01	2.55E−01	2.70E−01	2.79E−01	2.54E−01	2.88E−01
Spike Latency	1.62E−06	1.55E−07	2.03E−02	1.76E−02	2.77E−06	1.33E−07	2.44E−02	1.11E−02
Subthreshold oscillation	3.49E−06	4.08E−07	2.80E−02	4.67E−02	3.21E−06	1.97E−07	3.38E−02	1.98E−02
Resonator	1.47E−05	3.25E−07	3.80E−02	4.50E−02	1.93E−06	7.37E−07	4.64E−02	2.20E−02
Integrator	3.36E−07	1.20E−07	2.27E−02	1.88E−02	2.42E−06	3.32E−08	2.42E−02	1.54E−02
Rebound spike	4.22E−06	1.25E−06	2.96E−02	2.30E−02	1.79E−05	2.13E−06	4.80E−02	2.15E−02
Rebound Burst	2.58E−05	5.59E−06	1.82E−01	5.82E−02	1.98E−05	2.80E−06	6.46E−02	5.84E−02
Threshold Variability	3.26E−06	4.69E−07	2.51E−02	4.24E−02	9.53E−07	1.42E−07	4.63E−02	2.85E−02
Bistability	7.96E−06	4.68E−06	1.16E−01	5.52E−02	1.20E−05	1.17E−06	1.04E−01	1.02E−01
DAP	1.08E−05	2.79E−07	7.19E−03	3.21E−02	1.43E−06	1.96E−07	1.43E−02	1.93E−02
Accomodation	6.60E−07	7.67E−08	6.18E−03	9.37E−03	4.38E−07	2.90E−08	8.43E−03	3.52E−03
Inhibition induced spiking	2.03E−04	4.85E−05	4.40E−02	1.06E−01	4.42E−05	1.57E−05	5.00E−02	3.59E−02
Inhibition induced bursting	2.52E−06	2.55E−07	7.33E−02	9.82E−02	2.86E−06	2.46E−07	6.08E−02	6.34E−02
Mean	1.15E−02	1.19E−02	7.99E−02	7.50E−02	1.35E−02	1.39E−02	8.32E−02	6.58E−02



(a) (b)

Fig. 8. Class 2 using FP32 and standard equations. (a) shows the first 250 ms of the simulation and (b) the final 50 ms. A slight timing lag can be seen in (b).

Table 4
Spike count deviations over 1000 ms for each firing type, equation type and arithmetic. '=' signifies no difference from the 64-bit reference.

Firing type	Ref	Standard equation				Ref	Rescaled equations			
	64-bit	32-bit		16-bit		64-bit	32-bit		16-bit	
	Float	Float	Posit	Float	Posit	Float	Float	Posit	Float	Posit
Tonic Spiking	38	=	=	=	=	38	=	=	=	=
Phasic spiking	1	=	=	=	−1	1	=	=	=	=
Tonic bursting	124	=	=	=	=	124	=	=	−2	=
Phasic bursting	7	=	=	=	−7	7	=	=	−1	−1
Mixed mode	33	=	=	=	=	33	=	=	=	−1
SFA	38	=	=	=	=	38	=	=	=	=
Class 1	115	=	=	=	=	115	=	=	=	=
Class 2	166	=	+1	+1	+2	167	=	=	+1	−1
Spike Latency	1	=	=	=	=	1	=	=	=	=
Subthreshold oscillation	1	=	=	=	=	1	=	=	=	=
Resonator	1	=	=	=	−1	1	=	=	+1	=
Integrator	1	=	=	=	=	1	=	=	=	=
Rebound spike	1	=	=	=	−1	1	=	=	−1	−1
Rebound Burst	11	=	=	+103	−11	11	=	=	−11	−11
Threshold Variability	1	=	=	−1	−1	1	=	=	−1	=
Bistability	5	=	=	+19	−3	5	=	=	+19	+21
DAP	1	=	=	=	+2	1	=	=	=	+1
Accomodation	1	=	=	=	=	1	=	=	=	=
Inhibition induced spiking	4	=	=	=	=	4	=	=	=	=
Inhibition Induced bursting	16	=	=	=	=	16	=	=	=	=

spike timing. In Table 5 we show the accumulated spike timing errors and, when using 32-bit arithmetic, this shows there were no spike timing errors for any firing type, again with the exception of Class 2. Similarly to other results, the situation was different when 16-bit arithmetic was used. When using standard equations, posit16 deviated from the reference nine times whereas, FP16 deviated only four times. This often meant that the expected dynamics were not achieved in that test. For example, 'Rebound Burst' should fire 11 spikes following a brief inhibitory input. FP16 fired a burst, but then continued firing afterwards meaning that 114 spikes were emitted over the 1000 ms. This is in contrast to posit16 which did not fire at all. Hence, in this case, both arithmetics failed to produce the expected burst of spikes. When the equations were rescaled, posit16 improved, deviating from the FP64 only seven times whereas FP16 got worse — deviated 8 times from FP64. Furthermore, with rescaled equations, posit16 was the only 16-bit arithmetic capable of reproducing the expected dynamics of the 'Threshold Variability' firing type.

Table 5 shows the mean accumulated timing error. When using 16-bit arithmetic, posit16 had a similar mean value to FP16 across the reproduced firing types when using the standard equations, but a much lower mean error when using the rescaled equations.

3.4. Fixed-point arithmetic

While floating-point and posit arithmetic have been investigated in this paper, fixed-point is another potential arithmetic option. Fixed-point arithmetic is much simpler and faster than either posit or floating-point and easier to implement. However, if care is not taken, it is highly sensitive to potentially catastrophic underflow and overflow errors. Although both Hopkins et al. [45] and Jin et al. [54] were able to achieve a 16-bit fixed-point implementation of an Izhikevich neuron, this was only managed after employing multiple error reducing strategies, which make their implementation incompatible with this work.

Hopkins et al. [45] used higher order ODE solvers namely, RK2, RK3 and Chan-Tsai. While these are more accurate, they entail a higher latency and computational overhead than the much simpler Forward Euler used here. Moreover, they chose only a subset of neuron firing types and a smaller time step. Using smaller time-steps generally improves the stability and accuracy of a solution when using numerical methods of integration. Indeed, the accuracy typically increases monotonically as step size reduces, but this not only increases the computational load, it can also interfere with the required neuron behaviour. For example, some firing types are very sensitive to input

Table 5
Absolute accumulated spike timing error over 1000 ms for each firing type using both standard and rescaled equations, comparing 32-bit and 16-bit arithmetics with the FP64 reference.

Firing type	Standard equations accumulated spike timing error (ms)				Rescaled equations accumulated spike timing error (ms)			
	32-bit		16-bit		32-bit		16-bit	
	Float	Posit	Float	Posit	Float	Posit	Float	Posit
Tonic Spiking	0.00	0.00	166.50	18.50	0.00	0.00	166.50	0.00
Phasic spiking	0.00	0.00	2.50	–	0.00	0.00	0.75	2.50
Tonic bursting	0.00	0.00	0.00	14.25	0.00	0.00	–	0.00
Phasic bursting	0.00	0.00	11.80	–	0.00	0.00	–	–
Mixed mode	0.00	0.00	21.75	11.00	0.00	0.00	116.25	–
SFA	0.00	0.00	132.25	140.25	0.00	0.00	8.50	140.25
Class 1	0.00	0.00	106.50	103.0	0.00	0.00	163.25	69.25
Class 2	31.00	–	–	–	257.50	331.50	–	–
Spike Latency	0.00	0.00	0.19	0.00	0.00	0.00	0.20	0.00
Subthreshold oscillation	0.00	0.00	0.25	0.00	0.00	0.00	0.25	0.25
Resonator	0.00	0.00	5.00	–	0.00	0.00	–	0.50
Integrator	0.00	0.00	0.75	0.50	0.00	0.00	0.75	0.25
Rebound Spike	0.00	0.00	6.20	–	0.00	0.00	–	–
Rebound Burst	0.00	0.00	–	–	0.00	0.00	–	–
Threshold Variability	0.00	0.00	–	–	0.00	0.00	–	5.00
Bistability	0.00	0.00	–	–	0.00	0.00	–	–
DAP	0.00	0.00	0.00	–	0.00	0.00	0.10	–
Accomodation	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Inhibition induced spiking	0.00	0.00	2.00	15.00	0.00	0.00	7.50	3.00
Inhibition Induced bursting	0.00	0.00	78.00	78.00	0.00	0.00	40.00	54.00
Mean	1.55	0.00	33.36	34.59	12.88	16.58	42.00	21.15

spike timing such as bistability; If there is too much lag/lead error, bistability does not stop firing on the second input, as it should. This behavioural dependence on the timestep size has been previously described in Heidarpur et al. [55] and Skocik and Long [32] and also explains why Hopkins et al. [45] and Jin et al. [54] only investigated a subset of firing types. Hopkins et al. [45] also used mixed precision throughout, opting for 32-bit fixed-point numbers for all constants and pre-computed several of these to control any rounding error. They admit that even after using all these strategies, using round-to-nearest rounding scheme, did not produce any spiking behaviour at all. Indeed, to reduce their errors to an acceptable level they had to employ stochastic-rounding. Which, although relatively simple to implement in hardware, inevitably increases the complexity of the design. Jin et al. [54] also achieved a spiking Izhikevich neuron using only 16-bits on the SpiNNaker system. However, in their work they needed to use two scaling factors and to rewrite the equations (Eqs. (9) and (10)) to better fit with the ARM instruction set. Having gone beyond simply rescaling, it becomes difficult to make comparisons with other Izhikevich implementations. Moreover, they did not assess their results for accuracy, making their work hard to compare against and difficult to replicate on non-ARM hardware. Nor did they attempt to reproduce all 20 firing patterns. Hence, while it has been previously demonstrated that 16-bit Izhikevich models are *possible* using 16-bit fixed-point arithmetic, it has not been shown that this is possible without either, changing the equations considerably, or using a much more complicated system to ameliorate the arithmetic errors. Nor has it been shown that at reduced precision, all firing types are possible. Hence, we have not considered fixed-point in this work.

3.5. BFloat16 arithmetic

Bfloat16 was originally conceived to reduce storage and computational load for deep learning applications. Since then, it has also been used to increase efficiency in both distributed and other scientific computing applications [56]. The format features 8 exponent bits, mirroring the FP32 format and 7 mantissa bits, providing enough dynamic range to accurately represent error gradients during backpropagation [51]. Previous studies have shown that for ML workloads, bfloat16 outperforms both posit16 and FP16 providing a better trade-off between silicon area and accuracy Mishra et al. [19].

However, despite its capabilities in other areas, bfloat16 proved incapable of faithfully replicating the dynamics associated with any of the 20 firing types tested, whether in their standard or rescaled forms.

3.6. Exploring $n < 16$ and the effect of different es values

Here we explore $n < 16$ bit simulations in posit arithmetic. Fig. 9 uses the rescaled equations to demonstrate tonic firing for posits from 8-bit to 15-bits. For each simulation $es = 0$ and the bottom right plot, in red, is the reference FP64. Each simulation was run for 1000 ms, but for clarity, each plot focuses on the final 100 ms of the simulation, where firing frequency has converged. The most obvious effect of reducing bit width is to increase the firing frequency. This is possibly a consequence of having fewer values in-between the simulation extremes, especially around zero where u is particularly sensitive $-0.00306 \leq u \leq 0.00198$. For example, posit$<15, 0>$ has 25 values in this interval, posit$<12, 0>$ has 5 and posit$<8, 0>$ has just 3. In posit$<8, 0>$ the smallest representable positive value *minpos* is 0.015625. For posit$<15, 0>$ *minpos* is 0.00012207. This coarseness in *minpos* also causes dU to be larger for smaller values of n and therefore u to change more rapidly, approaching $u = -0.042$, the point at which dV rapidly increases causing spike emission (See Fig. 11). Hence, as n decreases, *minpos* increases which affects how quickly u reaches the required value to cause spike emission.

The effect of changing *es* is not as clear cut as that of changing n. As stated in Section 1.2 *es* defines exponent size. This is used during posit decoding/encoding, controlling the value useed $= 2^{2^{es}}$ (Eq. (1)). If we focus on generating spiking behaviour, Fig. 10 shows that spikes are generated with posit$<8, 0>$ but, as soon as we add an *es* bit (posit$<8, 1>$), no spikes are emitted at all. At first glance, this appears to suggest that there are not enough fraction bits remaining to generate the required precision. However, even if we start using posit$<10, 1>$ the expected spiking behaviour is still not seen. Indeed, it only recommences when we start to use posit$<12, 1>$ for $es > 0$.

An explanation can be found by considering the trajectories of posit$<12, 1>$ and posit$<10, 1>$ in the phase plane of the rescaled equations (Fig. 11). The simulation begins at $u = -0.14$, with $v = -0.7$ and increasing. After two spikes, the posit$<10, 1>$ simulation appears to stop abruptly (Fig. 11(a)). Whereas, in Fig. 11(b) using posit$<12, 1>$ arithmetic, the simulation enters the limit cycle of regular firing, close to the FP64 reference. When using posit$<10, 1>$ the dynamics cease when $v = -0.6484375$, $u = -0.01953125$. At this point the next

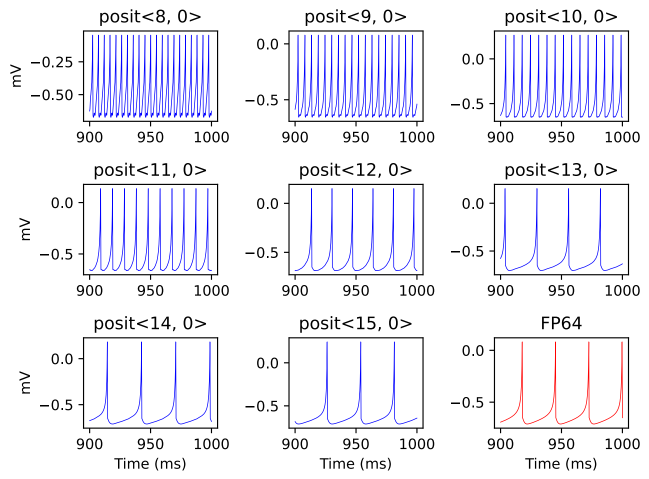


Fig. 9. Tonic firing for various bit-depths of posits with $es = 0$ for each. Time is on the x-axis in simulation steps ($dt = 0.25$). For clarity, we show only a section of the full simulation, where spike timing has converged and become regularly spaced.

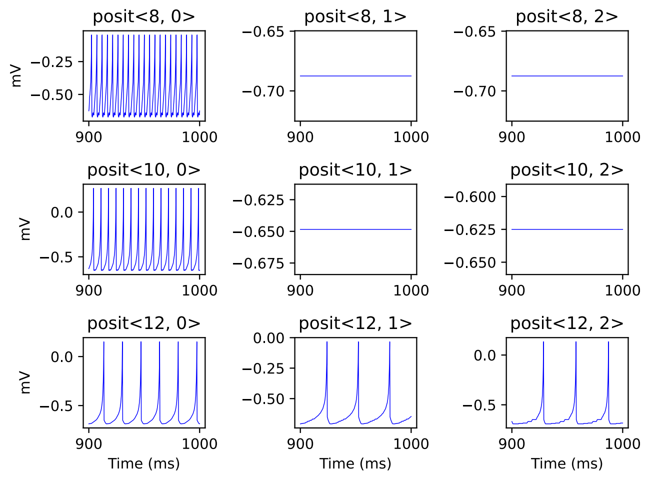


Fig. 10. Here we show the effect of changing both the bit width and es values on tonic spiking with the rescaled equations. Posit<8, es> are along the top, posit<10, es> the middle and posit<12, es> along the bottom row. From left to right the es values are 0, 1 and 2. Although at a much higher frequency, spiking behaviour occurs in all cases if $es = 0$. However, when $es > 0$ spiking is only seen in 12-bit posits.

representable value is $v = -0.640625$. The gap between these two is 0.0078125, but the size of the voltage update using the forward Euler's integration method (given by $0.25 \times \frac{dV}{dt}$) at this point is 0.000976562, which is less than half the required distance. Hence this is rounded away.

When using posit<12, 0>, the next representable point is −0.65234375, a gap of 0.00390625, with $\frac{dV}{dt} = -0.00390625$, allowing the simulation to proceed without stagnating. It may also be possible to avoid this by changing the timestep size, input current, complexity of the integration method or using stochastic rounding.

Additionally, we observed further interdependence between n and es. For example, as shown in Fig. 12, attempting to use posit<15, 1> or posit<15, 2> results in exact spike timing (albeit with some error in the spike amplitude, particularly in the $es = 2$ case). Subsequently, reducing n further to $n = 13$, introduces some error in spike timing which is minimised by setting $es = 2$ (Fig. 13). This result means that the fraction size alone should not be the only consideration when optimising such a system. Changing es and keeping n constant, changes the distribution of representable values within a given numerical interval. These results make designs such as PERI [26], which can switch es values at runtime,

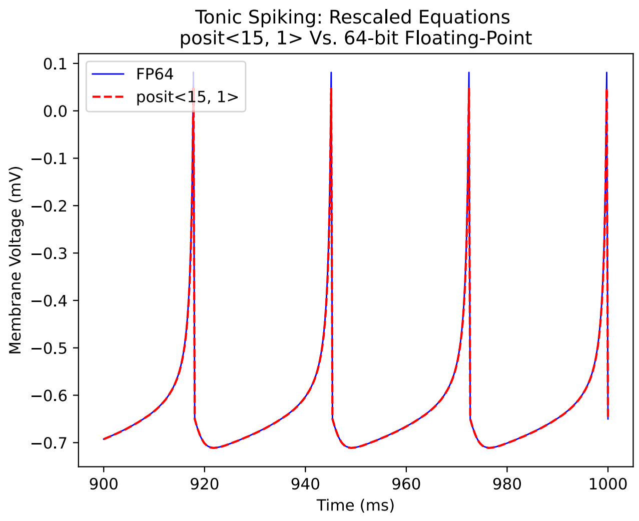


Fig. 11. Phase plots for the rescaled tonic firing equations with different arithmetics. (a) shows posit<10, 1> and (b) shows posit<12, 1>. In both plots the reference FP64 is also shown.

an intriguing prospect and highlight the need for further research into the use of posit arithmetic in scientific applications.

3.7. The effect of scaling

Under the assumption that the membrane voltage remained in the interval (−100 mV, 100 mV), regardless of firing type, an initial scaling factor of 0.01 was chosen. This served to keep the membrane voltage of the rescaled equations in the interval (−1.0, 1.0) where, as explained in Section 1, posits have the greatest accuracy. This region corresponds to where the number of regime bits is minimised and thus, the bits available to the fraction is maximised (the minimum regime length is two bits). From Table 2 the maximum value is 30, and the minimum value is −81.8. To keep the regime bits at their minimum, the largest representable value x, must be $x < useed$ for posit<16, 1> $useed = 4$.

Given, $-81.8 \times 0.05 = -4.09$ while this very upper bound value requires a 3-bit regime, the majority of the values used during a simulation will be within the minimal 2-bit regime. For this reason, the scaling factor 0.05 is at the upper bound of the 2-bit regime length, maximising precision. Smaller scaling factors will not gain any additional fraction bits, and larger scaling factors will lose fraction bits. Accordingly, we tested five scaling factors 0.005, 0.01, 0.05, 0.1 and 0.15 for four firing types (tonic spiking, tonic bursting, phasic spiking and phasic bursting)

As can be seen in Fig. 14, the best choice of scaling factor to produce the lowest NRMSE is firing type specific.

3.8. Implications for hardware design

In this paper, we demonstrate that it is possible to simulate Izhikevich neurons at reduced precision using posit arithmetic while minimising, and in some cases eradicating, any associated arithmetic errors. The implications for hardware are, however, harder to ascertain. In many respects, with fewer exception values, one rounding mode and an absence of subnormal numbers, posits are simpler than FP but, due to the dynamic sizing of the regime bits, posit decoding and encoding is significantly more complex than in the floating-point case. Published comparisons between Posit Arithmetic Units (PAU) and Floating-Point Units (FPU) often assert that, for a given word size, a PAU is typically larger than an FPU. However, such direct comparisons ignore nuances between the two systems. While FPUs are defined solely by their size, posits are characterised by both their size n and their es value, providing a wider and more versatile parameter space, but complicating

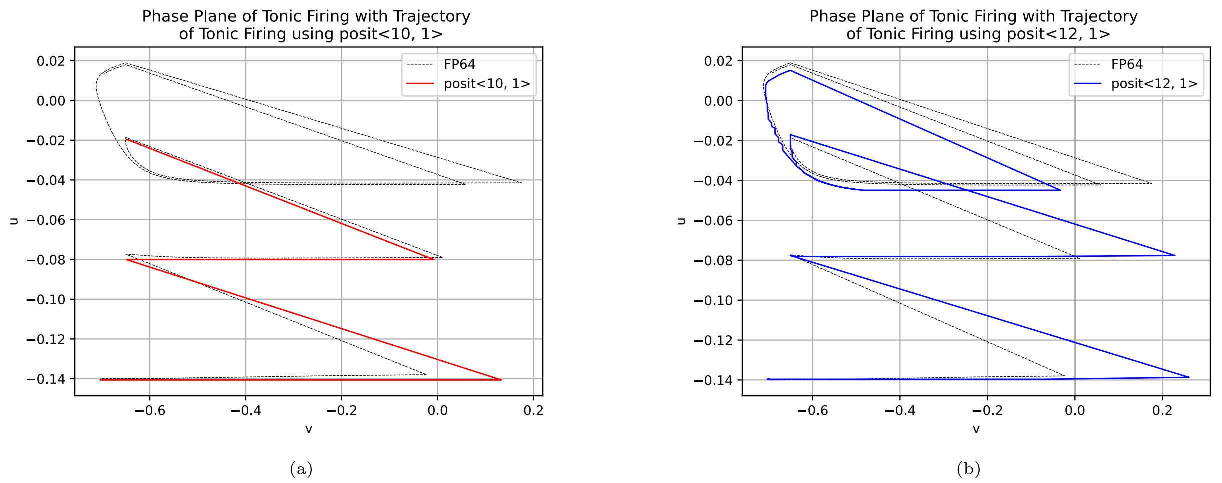


Fig. 12. This figure shows the final 100 ms of a 1000 ms simulation using posit<15, 1> arithmetic compared to FP64. We can see that posit<15, 1> is capable of exact spike timing at the end of 1000 ms simulation. Similarly to posit<16, 1> the only error is in the amplitude of the membrane voltage.

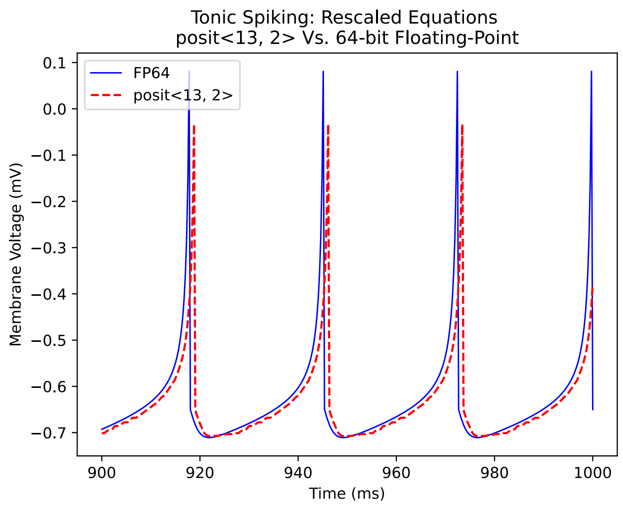


Fig. 13. This figure shows the final 100 ms of a 1000 ms simulation using posit<13, 2> arithmetic. This posit configuration is almost capable of exact spike timing, but leads the FP64 reference by 1.25 ms.

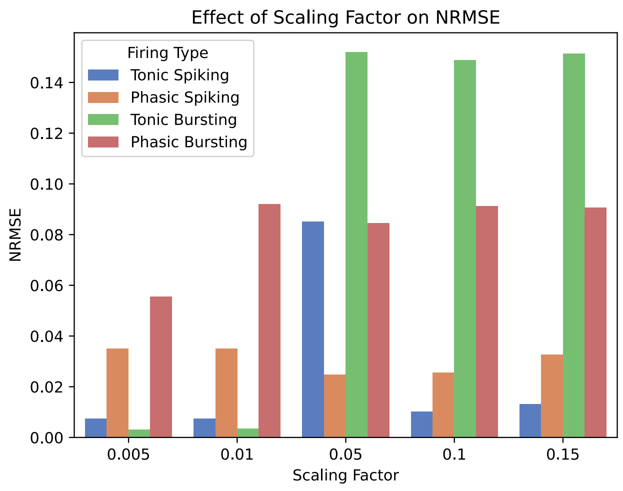


Fig. 14. Posit<16, 1> simulating the firing types tonic spiking, tonic bursting, phasic spiking and phasic bursting with scaling factors 0.005, 0.01, 0.05, 0.1 and 0.15.

comparisons. Some studies investigate multiple posit configurations, acknowledging the additional flexibility. For instance, Zhang et al. [38] reported posit<32, 8>, posit<16, 5> and posit<8, 4>, against FP32, FP16 and FP8.

Zhang et al. [38] synthesised their ASIC designs using the STM−28 nm library and each PAU grew in size as *n* was increased. This is not unexpected, but only one *es* value was used for each *n*. Chaurasiya et al. [21] gave more insight into effect on PAU size that *es* has, regardless of *n*. They synthesised their adder and multiplier using the 90 nm-CMOS Faraday library for two values of *es*. They found that for all values of *n*, increasing *es* reduces the resulting area usage. This relationship has also been described in [39]. However, the same correlation was not seen in FPGA designs by Chaurasiya et al. [21] who synthesised their designs for $1 \le es \le 4$ for $n = 16$ and $n = 32$ for a Zynq-7000 with Vivado 2017.4. No clear effect on PAU size for was seen for changes in *es* for a given *n* in this case.

Further complications arise as many studies focus only on a basic PAUs with limited functionality, such as standalone adders, multipliers [25], division units, cosine units [22], or Multiply Accumulate (MAC) units [38]. Additionally, discrepancies arise in studies with regard to IEEE-754 compliance, with variations in the inclusion of rounding modes, subnormal numbers, and exception values. For example, the Xilinx FPU 7.1 IP core [57] does not support subnormal numbers and, similarly to posits, recognises only three exception values: zero, infinity, and NaN. Although, several authors assert the additional overhead of extra rounding modes and subnormal number inclusion is minimal [58,59].

Chaurasiya et al. [21] are one of the few authors who compare a full IEEE-754 compliant FPU with a posit equivalent for both adders and multipliers. They report their posit<16, 1> adder required 391 LUT and the FP16 required 356 LUT. Similarly, their posit<16, 1> multiplier required 218 LUT compared to FP16 212 (all on a Zynq-7000 FPGA). Although the PAU is larger in both the adder and multiplier, they are not dissimilar. Additionally, as argued by multiple authors [20,25,38], the increase in accuracy often seen with posit arithmetic enables the replacement of longer word size FPU with a smaller word size PAU.

While these preliminary studies shed light on simple operations, they lack insights into the performance of more realistic and complex algorithms. To address this limitation, some studies incorporate PAUs into complete CPUs, such as PERCIVAL [27], CLARINET [60], PERI [26] and POSAR [41]. PERCIVAL is one of the most thorough and includes all posit operations including fused operations and the *quire* register into a RISC-V CVA6 core [27].

As discussed previously, the addition of a quire register significantly increases the overall size of a design (approximately half the total area in PERCIVAL). However, even when excluding the quire, [27] found that the PAU was still $1.32\times$ larger than a IEEE-754 compliant FPU. Nevertheless, [27] found the extra cost of the quire to be justified, when used to reduce the Mean Square Error of the generic matrix-matrix multiplication (GEMM) benchmark. Indeed, they show that in this benchmark, a 32-bit posit was up to four orders of magnitude more accurate than FP32.

Similarly, our results show that for some firing patterns, it is possible to achieve the same accuracy using posit$<16,1>$ compared to FP64. This relationship between increased accuracy while using fewer bits, often described as exchanging an n-bit FPU for a m-bit PAU, where $m < n$ has been described previously [21]. It should also be stressed that while most studies show PAUs are larger than their FPU counterparts some, more recent studies do not. Esmaeel et al. [22] developed a second order IIR notch filter including a modified Booth Multiplier for both posit32 and FP32. They report that the posit implementation was not only more accurate, but was faster and required 35.68% less area.

This highlights that posit research is still in its early exploration phase with many optimisations yet to be discovered. Recent advances such as Half-Unit Biased (HUB) [61], Fixed-Posit [29], and blended IEEE-754 floating-point/posit transprecision MAC units have recently been proposed [62]. Moreover, innovations such as two's complement posit decoding which removes the need to take the two's complement of any negative posits by changing the significand's MSB, simplifying any posit implementations while being mathematically equivalent, will likely continue to optimise future PAU implementations [63].

4. Conclusion

The objective of this study was to examine the feasibility of employing posit arithmetic as an alternative to FP for running SNN simulations using Izhikevich neurons. Additionally, we aimed to explore the effect on accuracy of reducing the bit depth in both posit and FP number systems; and for posits, the effect of changing es in relation to n. Typically, reducing the bit-depth has several advantages such as speed and power efficiency, but is often deleterious to accuracy. Hence, the impact of rescaling the equations, as a potential approach to mitigating this reduced accuracy was also investigated. This is the first study to establish quantitatively how posit arithmetic differs from FP in this context. Our research shows that there is very little difference between number systems at 32-bit either in terms of membrane voltage or spike timing. However, when the bit-depth was reduced to 16-bit, while errors become detectable, generally posit arithmetic was more accurate than FP16. Most notably, we showed the advantage of rescaling the Izhikevich equations when using posit arithmetic, especially for tonic firing. These results suggest that designers of future SNN accelerators may want to consider including a reduced precision PAU rather than an FPU. This could allow for more efficient utilisation of silicon resources in such systems.

While this study provides several important positive contributions, it suffers from a number of limitations. Notably, only one neuron type was tested. Reducing or increasing the complexity of the equations will likely have an impact on the performance of both arithmetic systems. A further limitation is the use of a single ODE solver and keeping the update step size to the original values defined by Izhikevich. Both of these variables have previously been shown to have a significant impact on SNN accuracy [44]. Additionally, this study only explored one mitigation technique — rescaling by a constant factor. It might be expected that different scaling factors could affect accuracy to differing extents, as was found in Klöwer et al. [18]. Other mitigation techniques are also possible but were not explored here, such as using mixed precision with a larger bit-depth for variables which require a higher precision or intermediate results, which can be outside of the optimal range for a particular arithmetic. This might be addressed by reordering or precomputing various values, and this was not considered in this work. Another interesting avenue for future work would be to further explore the relationship between n, es, scaling factor and firing type. It would also be interesting to compare these arithmetics when used in a network context.

Nevertheless, this study offers some important contributions and highlights posit arithmetic as an interesting avenue of future SNN research. We therefore suggest future works attempt to establish the importance of ODE solver choice and time-step size on accuracy while running posit-based SNN simulations.

CRediT authorship contribution statement

T. Fernandez-Hart: Writing – review & editing, Writing – original draft, Visualization, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **James C. Knight:** Writing – review & editing, Writing – original draft, Supervision, Investigation, Conceptualization. **T. Kalganova:** Writing – review & editing, Writing – original draft, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: T. Fernandez-Hart reports financial support was provided by Sundance Multiprocessor Ltd. James C Knight reports financial support was provided by Engineering and Physical Sciences Research Council. Tim Fernandez-Hart reports financial support was provided by Brunel University London. Tatiana Kalganova reports a relationship with SUNDANCE that includes: funding grants.

Data availability

No data was used for the research described in the article.

Acknowledgements

TFH was part funded by Sundance Multiprocesssor Ltd., UK. and an EPSRC Doctoral Training Partnerships (DTP) grant. JK was funded by the EPSRC (grants EP/V052241/1 & EP/S030964/1)

References

[1] Q.T. Pham, T.Q. Nguyen, P.C. Hoang, Q.H. Dang, D.M. Nguyen, H.H. Nguyen, A review of SNN implementation on FPGA, in: 2021 International Conference on Multimedia Analysis and Pattern Recognition, MAPR, 2021, pp. 1–6, http://dx.doi.org/10.1109/MAPR53640.2021.9585245.

[2] W.C. Abraham, O.D. Jones, D.L. Glanzman, Is plasticity of synapses the mechanism of long-term memory storage? npj Sci. Learn. 4 (1) (2019) 9, http://dx.doi.org/10.1038/s41539-019-0048-y.

[3] W. Guo, M.E. Fouda, A.M. Eltawil, K.N. Salama, Neural Coding in Spiking Neural Networks: A Comparative Study for Robust Neuromorphic Systems, Front. Neurosci. 15 (2021) http://dx.doi.org/10.3389/fnins.2021.638474.

[4] C. Frenkel, M. Lefebvre, J.-D. Legat, D. Bol, A 0.086-mm^2 12.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28nm CMOS, IEEE Trans. Biomed. Circuits Syst. (2018) 1, http://dx.doi.org/10.1109/TBCAS.2018.2880425.

[5] Y. Cao, Y. Chen, D. Khosla, Spiking deep convolutional neural networks for energy-efficient object recognition, Int. J. Comput. Vis. 113 (1) (2015) 54–66, http://dx.doi.org/10.1007/s11263-014-0788-3.

[6] Z. Bing, C. Meschede, F. Röhrbein, K. Huang, A.C. Knoll, A survey of robotics control based on learning-inspired spiking neural networks, Front. Neurorobot. 12 (2018) 35, http://dx.doi.org/10.3389/fnbot.2018.00035.

[7] R.-J. Zhu, Q. Zhao, J.K. Eshraghian, SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks, 2023, http://dx.doi.org/10.48550/arXiv.2302.13939, URL https://arxiv.org/abs/2302.13939.

[8] T.P. Vogels, L.F. Abbott, Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons, J. Neurosci. 25 (46) (2005) 10786–10795, http://dx.doi.org/10.1523/JNEUROSCI.3508-05.2005.

[9] L. Hodgkin, A. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. 117 (4) (1952) 500–544.

[10] E. Izhikevich, Simple model of spiking neurons, IEEE Trans. Neural Netw. 14 (6) (2003) 1569–1572, http://dx.doi.org/10.1109/TNN.2003.820440.

[11] A. Tamura, T. Ueta, S. Tsuji, Bifurcation analysis of Izhikevich model, in: International Symposium on Nonlinear Theory and its Applications, 2008, pp. 424–427.

[12] E. Izhikevich, Which model to use for cortical spiking neurons? IEEE Trans. Neural Netw. 15 (5) (2004) 1063–1070, http://dx.doi.org/10.1109/TNN.2004.832719.

[13] J.C. Knight, T. Nowotny, Larger GPU-accelerated brain simulations with procedural connectivity, Nat. Comput. Sci. 1 (2021) 136–142, http://dx.doi.org/10.1101/2020.04.27.063693.

[14] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S.H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, H. Wang, Loihi: A neuromorphic manycore processor with on-chip learning, IEEE Micro 38 (1) (2018) 82–99, http://dx.doi.org/10.1109/MM.2018.112130359.

[15] S.B. Furber, F. Galluppi, S. Temple, L.A. Plana, The SpiNNaker project, Proc. IEEE 102 (5) (2014) 652–665, http://dx.doi.org/10.1109/JPROC.2014.2304638.

[16] R. Wang, A. van Schaik, Breaking Liebig's Law: An Advanced Multipurpose Neuromorphic Engine, Front. Neurosci. 12 (2018) http://dx.doi.org/10.3389/fnins.2018.00593.

[17] S.W. Moore, P.J. Fox, S.J. Marsh, A.T. Markettos, A. Mujumdar, Bluehive - A field-programable custom computing machine for extreme-scale real-time neural network simulation, in: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, pp. 133–140, http://dx.doi.org/10.1109/FCCM.2012.32.

[18] M. Klöwer, P.D. Düben, T.N. Palmer, Number Formats, Error Mitigation, and Scope for 16-Bit Arithmetics in Weather and Climate Modeling Analyzed With a Shallow Water Model, J. Adv. Modelling Earth Syst. 12 (10) (2020) e2020MS002246, http://dx.doi.org/10.1029/2020MS002246.

[19] S.M. Mishra, A. Tiwari, H.S. Shekhawat, P. Guha, G. Trivedi, P. Jan, Z. Nemec, Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithms, in: 2022 32nd International Conference Radioelektronika, 2022, pp. 1–6, http://dx.doi.org/10.1109/RADIOELEKTRONIKA54537.2022.9764927, URL https://ieeexplore.ieee.org/document/9764927.

[20] J.L. Gustafson, I.T. Yonemoto, Beating Floating Point at its Own Game: Posit Arithmetic, Supercomput. Front. Innov. 4 (2) (2017) 71–86, http://dx.doi.org/10.14529/jsfi170206.

[21] R. Chaurasiya, J. Gustafson, R. Shrestha, J. Neudorfer, S. Nambiar, K. Niyogi, F. Merchant, R. Leupers, Parameterized posit arithmetic hardware generator, in: 2018 IEEE 36th International Conference on Computer Design, ICCD, IEEE, 2018, pp. 334–341, http://dx.doi.org/10.1109/ICCD.2018.00057.

[22] A.A. Esmaeel, S. Abed, B.J. Mohd, A.A. Fairouz, POSIT vs. Floating point in implementing IIR notch filter by enhancing radix-4 modified booth multiplier, Electronics 11 (1) (2022) 163, http://dx.doi.org/10.3390/electronics11010163.

[23] F. de Dinechin, L. Forget, J.-M. Muller, Y. Uguen, Posits: the good, the bad and the ugly, in: Proceedings of the Conference for Next Generation Arithmetic 2019, ACM, 2019, pp. 1–10, http://dx.doi.org/10.1145/3316279.3316285.

[24] L. Forget, Y. Uguen, Comparing posit and IEEE-754 hardware cost, 2021, p. 13, HAL-03195756v3, URL https://hal.archives-ouvertes.fr/hal-03195756v3.

[25] M.K. Jaiswal, H.K.-H. So, PACoGen: A Hardware Posit Arithmetic Core Generator, IEEE Access 7 (2019) 74586–74601, http://dx.doi.org/10.1109/ACCESS.2019.2920936.

[26] S. Tiwari, N. Gala, C. Rebeiro, V. Kamakoti, PERI: A Configurable Posit Enabled RISC-V Core, ACM Trans. Archit. Code Optim. 18 (3) (2021) 25:1–25:26, http://dx.doi.org/10.1145/3446210.

[27] D. Mallasén, R. Murillo, A.A.D. Barrio, G. Botella, L. Piñuel, M. Prieto-Matias, PERCIVAL: Open-Source Posit RISC-V Core With Quire Capability, IEEE Trans. Emerg. Top. Comput. 10 (3) (2022) 1241–1252, http://dx.doi.org/10.1109/TETC.2022.3187199.

[28] J. Hou, Y. Zhu, S. Du, S. Song, Enhancing Accuracy and Dynamic Range of Scientific Data Analytics by Implementing Posit Arithmetic on FPGA, J. Sign. Process. Syst. 91 (10) (2019) 1137–1148, http://dx.doi.org/10.1007/s11265-018-1420-5.

[29] V. Gohil, S. Walia, J. Mekie, M. Awasthi, Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications, 2021, http://dx.doi.org/10.48550/arXiv.2104.04763.

[30] R. Murillo, A.A. Del Barrio, G. Botella, M.S. Kim, H. Kim, N. Bagherzadeh, PLAM: A Posit Logarithm-Approximate Multiplier, IEEE Trans. Emerg. Top. Comput. 10 (4) (2022) 2079–2085, http://dx.doi.org/10.1109/TETC.2021.3109127.

[31] V.H.L. Silva, J.F. Chaves, R.M. Gomes, B.A. Santos, Posit-Based Spiking Neuron in an FPGA, Vol. 21, 2021, p. 4.

[32] M.J. Skocik, L.N. Long, On the Capabilities and Computational Costs of Neuron Models, IEEE Trans. Neural Netw. Learn. Syst. 25 (8) (2014) 1474–1483, http://dx.doi.org/10.1109/TNNLS.2013.2294016.

[33] D.E. Oorschot, Total number of neurons in the neostriatal, pallidal, subthalamic, and substantia nigral nuclei of the rat basal ganglia: A stereological study using the cavalieri and optical disector methods, J. Comp. Neurol. 366 (4) (1996) 580–599, http://dx.doi.org/10.1002/(SICI)1096-9861(19960318)366:4<580::AID-CNE3>3.0.CO;2-0.

[34] B. Sen-Bhattacharya, S. James, O. Rhodes, I. Sugiarto, A. Rowley, A.B. Stokes, K. Gurney, S.B. Furber, Building a Spiking Neural Network Model of the Basal Ganglia on SpiNNaker, IEEE Trans. Cogn. Dev. Syst. 10 (3) (2018) 823–836, http://dx.doi.org/10.1109/TCDS.2018.2797426.

[35] IEEE standard for floating-point arithmetic, in: IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019, pp. 1–84, http://dx.doi.org/10.1109/IEEESTD.2019.8766229.

[36] J. Gustafson, G. Bohlender, S.Y. Chung, V. Dimitrov, Standard for Posit^{TM} Arithmetic (2022), 2022, URL https://posithub.org/docs/posit_standard-2.pdf. (Accessed 01 February 2023).

[37] A.Y. Romanov, A.L. Stempkovsky, I.V. Lariushkin, G.E. Novoselov, R.A. Solovyev, V.A. Starykh, I.I. Romanova, D.V. Telpukhov, I.A. Mkrtchan, Analysis of Posit and Bfloat Arithmetic of Real Numbers for Machine Learning, IEEE Access (ISSN: 2169-3536) 9 (2021) 82318–82324, http://dx.doi.org/10.1109/ACCESS.2021.3086669.

[38] H. Zhang, J. He, S.-B. Ko, Efficient Posit Multiply-Accumulate Unit Generator for Deep Learning Applications, in: 2019 IEEE International Symposium on Circuits and Systems, ISCAS, 2019, pp. 1–5, http://dx.doi.org/10.1109/ISCAS.2019.8702349.

[39] H. Zhang, S.-B. Ko, Design of Power Efficient Posit Multiplier, IEEE Trans. Circuits Syst. II 67 (5) (2020) 861–865, http://dx.doi.org/10.1109/TCSII.2020.2980531.

[40] N.-M. Ho, D.-T. Nguyen, H.D. Silva, J.L. Gustafson, W.-F. Wong, I.J. Chang, Posit Arithmetic for the Training and Deployment of Generative Adversarial Networks, in: 2021 Design, Automation & Test in Europe Conference & Exhibition, 2021, pp. 1350–1355, http://dx.doi.org/10.23919/DATE51398.2021.9473933.

[41] S.D. Ciocirlan, D. Loghin, L. Ramapantulu, N. Tapus, Y.M. Teo, The accuracy and efficiency of posit arithmetic, in: 2021 IEEE 39th International Conference on Computer Design, ICCD, IEEE, 2021, pp. 83–87, http://dx.doi.org/10.1109/ICCD53106.2021.00024.

[42] A. Dawson, P.D. Düben, Rpe v5: an emulator for reduced floating-point precision in large numerical simulations, Geosci. Model Dev. 10 (6) (2017) 2221–2230, http://dx.doi.org/10.5194/gmd-10-2221-2017.

[43] E. Izhikevich, Figure1.m, 2004, URL http://www.izhikevich.org/publications/figure1.m. (Accessed 01 February 2023).

[44] M. Hopkins, S. Furber, Accuracy and Efficiency in Fixed-Point Neural ODE Solvers, Neural Comput. 27 (10) (2015) 2148–2182, http://dx.doi.org/10.1162/NECO_a_00772.

[45] M. Hopkins, M. Mikaitis, D.R. Lester, S. Furber, Stochastic rounding and reduced-precision fixed-point arithmetic for solving neural ordinary differential equations, Phil. Trans. R. Soc. A 378 (2166) (2020) 20190052, http://dx.doi.org/10.1098/rsta.2019.0052.

[46] C. Leong, SoftPosit, 2022, URL https://gitlab.com/cerlane/SoftPosit. (Accessed 23 September 2022).

[47] J. Hauser, Berkeley SoftFloat, 2018, URL http://www.jhauser.us/arithmetic/SoftFloat.html. (Accessed 21 January 2023).

[48] C. Leong, SoftFloat-Python, 2019, URL https://gitlab.com/cerlane/SoftFloat-Python/-/tree/master. (Accessed 02 February 2023).

[49] E.T.L. Omtzigt, J. Quinlan, Universal Numbers Library: Multi-format Variable Precision Arithmetic Library, J. Open Sour. Softw. 8 (83) (2023) 5072, http://dx.doi.org/10.21105/joss.05072.

[50] Google, The bfloat16 numerical format | Cloud TPU | Google Cloud, 2023, URL https://cloud.google.com/tpu/docs/bfloat16. (Accessed 18 April 2023).

[51] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D.T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, P. Dubey, A Study of BFLOAT16 for Deep Learning Training, 2019, http://dx.doi.org/10.48550/arXiv.1905.12322, URL http://arxiv.org/abs/1905.12322. (Accessed 14 December 2023).

[52] M. Heidarpur, A. Ahmadi, M. Ahmadi, M. Rahimi Azghadi, CORDIC-SNN: On-FPGA STDP Learning With Izhikevich Neurons, IEEE Trans. Circuits Syst. I. Regul. Pap. 66 (7) (2019) 2651–2661, http://dx.doi.org/10.1109/TCSI.2019.2899356.

[53] M. Hayati, M. Nouri, S. Haghiri, D. Abbott, Digital Multiplierless Realization of Two Coupled Biological Morris-Lecar Neuron Model, IEEE Trans. Circuits Syst. I. Regul. Pap. 62 (7) (2015) 1805–1814, http://dx.doi.org/10.1109/TCSI.2015.2423794.

[54] X. Jin, S.B. Furber, J.V. Woods, Efficient modelling of spiking neural networks on a scalable chip multiprocessor, in: 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), 2008, pp. 2812–2819, http://dx.doi.org/10.1109/IJCNN.2008.4634194.

[55] M. Heidarpur, A. Ahmadi, M. Ahmadi, Time Step Impact on Performance and Accuracy of Izhikevich Neuron: Software Simulation and Hardware Implementation, in: 2020 IEEE International Symposium on Circuits and Systems, ISCAS, 2020, pp. 1–5, http://dx.doi.org/10.1109/ISCAS45731.2020.9180632.
[56] J. White, K. Adámek, J. Roy, S. Dimoudi, S.M. Ransom, W. Armour, Bits Missing: Finding Exotic Pulsars Using bfloat16 on NVIDIA GPUs, Astrophys. J. Suppl. Ser. 265 (1) (2023) 13, http://dx.doi.org/10.3847/1538-4365/acb351.
[57] Xilinx, Floating-Point Operator v7.1 LogiCORE IP Product Guide, 2020, URL https://docs.xilinx.com/v/u/en-US/pg060-floating-point. (Accessed 21 November 2023).
[58] F. De Dinechin, B. Pasca, Designing custom arithmetic data paths with FloPoCo, IEEE Des. Test Comput. 28 (4) (2011) 18–27, http://dx.doi.org/10.1109/MDT.2011.44.
[59] J.-M. Muller, N. Brisebarre, F. Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres, Handbook of Floating-Point Arithmetic, ISBN: 978-0-8176-4704-9, 2010, http://dx.doi.org/10.1007/978-0-8176-4705-6.
[60] N.N. Sharma, R. Jain, M.M. Pokkuluri, S.B. Patkar, R. Leupers, R.S. Nikhil, F. Merchant, CLARINET: A quire-enabled RISC-V-based framework for posit arithmetic empiricism, J. Syst. Archit. 135 (2023) 102801, http://dx.doi.org/10.1016/j.sysarc.2022.102801.
[61] R. Murillo, J. Hormigo, A.A.D. Barrio, G. Botella, HUB Meets Posit: Arithmetic Units Implementation, IEEE Trans. Circuits Syst. II (2023) 1, http://dx.doi.org/10.1109/TCSII.2023.3307488.
[62] L. Crespo, P. Tomás, N. Roma, N. Neves, Unified posit/IEEE-754 Vector MAC Unit for transprecision computing, IEEE Trans. Circuits Syst. II 69 (5) (2022) 2478–2482, http://dx.doi.org/10.1109/TCSII.2022.3160191.
[63] R. Murillo, D. Mallasén, A.A. Del Barrio, G. Botella, Comparing Different Decodings for Posit Arithmetic, in: J. Gustafson, V. Dimitrov (Eds.), Next Generation Arithmetic, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2022, pp. 84–99, http://dx.doi.org/10.1007/978-3-031-09779-9_6.

Tim Fernandez-Hart has a B.Sc. (Hons) in Genetics, an M.Sc. in Industrial Biotechnology, a M.Sc. in Medical Ultrasound and received a B.Sc. with first-class honours in Mathematics and Statistics from The Open University, UK in 2021. He is currently working towards a Ph.D. in Computer Systems Research at Brunel University, London, UK. His research interests include AI, neuromorphic engineering, spiking neural networks, event-based sensors and computer arithmetic.

James Knight received his BEng degree in Electronic Engineering from the University of Warwick in 2006. After working in industry for several years, in 2013, he received an MPhil in Advanced Computer Science from the University of Cambridge and, in 2016, a Ph.D. in Computer Science from the University of Manchester. His doctoral work focussed on using the SpiNNaker neuromorphic supercomputer to simulate large-scale computational neuroscience models with synaptic plasticity. James has worked at the University of Sussex since 2017, first as a PostDoctoral researcher and now as a EPSRC Research Software Engineering fellow, focussing on using GPU hardware to accelerate brain-inspired AI.

Tatiana Kalganova received the B.Sc. (Hons.) and Ph.D. degrees. She is currently a Professor in intelligent systems and the Electronic and Computer Engineering Postgraduate Research Director with Brunel University London, Uxbridge, U.K. She has over 30 years of experience in the design and implementation of applied intelligent systems.