An Approximate Computing based Spiking Neural Networks Neuron Model and STDP Learning

Haihang Xia, Haotian Liu, Yuqin Zhao, John Goodenough, Charith Abhayaratne, *Member, IEEE,* Sichen Liu, Sizhao Li, *Member, IEEE,* and Tiantai Deng

***Abstract*—Spiking Neural Networks (SNNs) show great potential in applications such as image processing, robotics, and communications. However, the vast number of neuron models and learning algorithms in large-scale SNNs impose significant hardware and energy overhead, with multiplication remaining the most critical operation. Thus, to address this challenge, this paper presents the hardware design of the Logarithmic Linear Multiply (LLMu) and Logarithmic Linear Segmented Multiply (LLSMu). These two components are specifically designed for neuron models and learning algorithms, achieving high accuracy with low hardware resource utilization and energy consumption. To demonstrate the capabilities of LLMu and LLSMu, we implement two mainstream SNN neuron models—Leaky Integrate-and-Fire (LIF) and Izhikevich—as well as the Spike Timing-Dependent Plasticity (STDP) learning algorithm, and compare their performance with state-of-the-art approaches on FPGA and ASIC platforms. The scope of this work is limited to these models and algorithms. The LLMu- and LLSMu-based implementations exhibit significantly improved energy efficiency over existing methods. Specifically, in the FPGA implementation, the LLSMu-based LIF neuron model achieves a 6.75× improvement, the LLSMu-based Izhikevich neuron model achieves a 2.70× to 3.72× improvement, and the LLMu-based STDP achieves a 21.03× to 48.78× improvement in energy efficiency. In the ASIC implementation, the LLSMu-based Izhikevich neuron model further improves energy efficiency by 5.58× to 5.69×, while the LLMu-based STDP achieves 5.96× and 3.69× improvements compared to prior designs.**

***Index Terms*—Spiking Neural Network, FPGA, ASIC, SNN Accelerator, Approximate Computing, Izhikevich Neuron Model, Leaky Integrate-and-Fire Neuron Model, Spike Timing-Dependent Plasticity**

I. Introduction

SPIKING Neural Networks (SNNs) have been widely explored in applications such as image processing, robotics, motor control, communication, and event-based data processing [1], particularly for energy- and resource-constrained platforms such as edge and embedded devices. Owing to their low power consumption, reduced hardware resource requirements, and unique learning mechanisms, SNNs are increasingly regarded as promising candidates for next-generation neural networks [2].

In SNNs, the neuron model serves as the most basic computational unit, capturing the dynamics of membrane potential by mathematically modeling the voltage difference across a biological neuron's cell membrane [3]. In an effort to achieve a balance between computational efficiency and biological realism, a number of neuron models have been developed, including the Leaky Integrate-and-Fire (LIF), Hodgkin-Huxley [4], and Izhikevich [5] models. Among the various neuron models proposed to date, the LIF and Izhikevich neuron models have been the most widely adopted in SNN applications.

In addition to neuron models, learning algorithms serve as a key mechanism for inter-neuron communication, which is primarily realized through synapses. These synapses function as the fundamental medium for both signal transmission and synaptic plasticity [6]. Among them, Spike-Timing Dependent Plasticity (STDP) stands out due to its simplicity and strong biological plausibility, as it depends only on the relative timing of presynaptic and postsynaptic spikes [7].

However, despite the simplicity of individual neuron models and learning algorithms, large-scale SNNs contain vast numbers of neurons and synapses, which constitute the primary source of energy and hardware resource overhead [8]. Moreover, the asynchronous nature of SNNs leads to irregular synaptic weight access patterns, making them inefficient on conventional von Neumann architectures [9]. Therefore, optimizing neuron models and learning algorithms, together with designing specialized computing hardware for SNNs, is essential to reducing power consumption and hardware resource utilization without compromising performance.

Among these models and algorithms, multiplication is the most computationally intensive operation. Consequently, refining multipliers used in neuron models and learning algorithms can substantially reduce hardware resource utilization and power consumption in SNN deployments.

In this context, to simplify the computations in neuron models without compromising accuracy, Z. Peng *et.al.* leveraged the CORDIC algorithm to perform the square operation in the Izhikevich neuron model [11]. Similarly, J. Wu *et.al.* adopted the CORDIC method to implement multiplication and exponential functions in the LIF neuron model and the STDP learning algorithm [8]. While the CORDIC algorithm offers improved computational speed and reduced power consumption, it requires multiple iterations to achieve high accuracy. This iterative nature results in increased storage demands, higher latency, and a larger hardware footprint. To address these limitations, J.Kim *et al.* use Template-Scaling-Based Exponential Function Approximation (TS-EFA) for modeling the LIF neuron [25]. TS-EFA slightly alleviates the high latency of the CORDIC algorithm with lookup-table-based approximation functions. However, TS-EFA's FPGA imple-

Haihang Xia, Haotian Liu, Yuqin Zhao, John Goodenough, Charith Abhayaratne, Sichen Liu, and Tiantai Deng are with the School of Electrical and Electronic Engineering, The University of Sheffield, S1 3JD Sheffield, U.K.

Sizhao Li is with the College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China.

mentation still suffers from high resource utilization. This is primarily due to its reliance on precomputed results stored in RAM, increasing memory usage, power consumption, and reducing computational efficiency.

Motivated by these limitations, this paper proposes the Logarithmic Linear Multiply (LLMu) and Logarithmic Linear Segmented Multiply (LLSMu) methods with low hardware resource utilization, high throughput, and high energy efficiency. These methods are specifically designed to optimize multiplication operations in SNN neuron models and learning algorithms. Given the high robustness of SNN systems, small computational errors introduced by approximate computing have minimal impact on overall network performance, while significantly reducing hardware resource requirements and power consumption. Unlike conventional iterative methods such as the CORDIC algorithm, the proposed method employs linear approximation to directly estimate the result. Furthermore, by decomposing large multiplications into smaller parallel operations, the LLSMu architecture achieves further improvements in speed and efficiency. A systematic error analysis is conducted to design effective compensation strategies and perform hardware-level optimizations, thereby improving computational accuracy. The proposed method is generalizable to any SNN neuron model or learning algorithm involving multiplication. In this work, the implementation and evaluation focus on three widely adopted components: the LIF and Izhikevich neuron models, and the STDP learning algorithm, with the scope limited to these models and algorithms. The main contributions of this paper include:

1) The traditional linear approximation method is enhanced through systematic error analysis and a segmented parallel computation strategy. Based on this, the proposed LLSMu approach achieves higher accuracy and faster computation compared to conventional linear approximation-based multipliers [13], with a 31.4% reduction in approximation error.
2) The LLSMu method was applied to the Izhikevich neuron model and evaluated on FPGA and ASIC platforms. On the FPGA, it improved throughput by 21.26 to 108.33% and energy efficiency by 2.70× to 3.72× over prior designs. On the ASIC, it reduced area by 71.30% and 76.64% compared to the CORDIC and PWL approaches, respectively.
3) The LLSMu method was applied to the LIF neuron model and evaluated on FPGA and ASIC platforms. On the FPGA, it reduced slice register utilization by 17.47 to 92.69% and improved energy efficiency by 6.75× over prior designs. On the ASIC, it achieved a 5.208× to 15.625× improvement in maximum operating frequency.
4) The LLMu method was applied to the STDP learning algorithm and evaluated on FPGA and ASIC platforms. On the FPGA, it reduced slice register and LUT utilization by 6.56 to 91.68% and 22.92 to 92.41%, respectively, improved maximum operating frequency by 47.96 to 114.25%, and enhanced energy efficiency by 21.03× to 48.78× over prior designs. On the ASIC, it achieved 3.979× and 1.953× higher throughput compared to CLSTDP and ImSTDP, respectively.
5) SNNs of different sizes were built using the LLMu-based STDP and LLSMu based neuron models and subsequently evaluated on real-world rotor dataset. The test accuracy ranged from 87.23% to 98.77%, demonstrating that the errors introduced by approximate computing had minimal impact on the system-level performance.

The remainder of this paper is organized as follows: Section II reviews related neuron models, learning algorithms, and approximate computing techniques. Section III presents the proposed LLMu and LLSMu method, including their applications. Section IV presents the evaluation of LLMu and LLSMu method with their application. Section V presents the hardware architecture of the proposed SNN neuron models and learning algorithms. Section VI reports the FPGA and ASIC implementation results of all proposed designs. Section VII concludes the paper.

II. Related Works

A. SNN Neuron Model

1) Izhikevich neuron model: The Izhikevich model [5] is described as

$$\begin{aligned}\frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I\\ \frac{du}{dt} &= a(bv - u).\end{aligned} \tag{1}$$

The parameters a, b, c, and d define the recovery time scale, sensitivity to membrane potential, membrane recovery value, and recovery increment, respectively. When the membrane potential v reaches the threshold, a spike is generated, with the change in v updated as

$$if \quad v \geq 30\,mV,\ then \begin{cases} v = c \\ u = u + d. \end{cases} \tag{2}$$

2) LIF neuron model: The discrete-time iteration equation of the membrane potential v in the LIF neuron model [8] is given by

$$V[t+1] = e^{-\frac{1}{\tau}} \cdot (V[t] - E_{rest}) + E_{rest} + I. \tag{3}$$

In (3), τ represents the membrane time constant. When v reaches the threshold, the neuron generates a spike and updates v to the resting potential as

$$if\ V > V_{th},\ then\ V = E_{rest}. \tag{4}$$

B. Pair-based trace-STDP Learning Algorithm

Trace-STDP (t-STDP) is a representative form of STDP [12]. Its discretized computation is given by

$$\begin{aligned} x\tau_j[n] &= x\tau_j[n] - \frac{x\tau_j[n-1]}{\tau_j} + s_j[n] \\ \Delta W[n] &= F_j(\omega[n]) \cdot x\tau_i[n] \cdot s_j[n] \\ &\quad - F_i(\omega[n]) \cdot y\tau_j[n] \cdot s_i[n], \end{aligned} \tag{5}$$

where $x\tau_j$ represents the trace of the post-synaptic spike, s_j denotes the spike generation at the current time step of the post-synapse, and τ_j is the time constant for the post-synaptic trace. In addition, $\Delta W[n]$ denotes the weight change, with $F_j(\omega[n])$ and $F_i(\omega[n])$ serving as fixed amplification factors.

C. Mitchell Approximate Method

Mitchell's method is a typical logarithmic linear approximation known for its high hardware efficiency [13]. The method operates as follows. Assume that there are two N-bit fixed-point numbers A and B. Suppose the first '1' in A and B is at the position k_a and k_b, where $(N-1) \geq \ k_a, k_b \geq 0$. By extracting the characteristic bit and separating the fractional part, A can be expressed as

$$A = 2^{k_a}\left(1 + \sum_{i=0}^{k_a-1} 2^i a_i\right). \tag{6}$$

Subsequently, the fractional part is denoted by $\widetilde{A}$ as follows:

$$A = \ 2^{k_a} \cdot (1 + \widetilde{A}). \tag{7}$$

Taking the base-2 logarithm of (7) yields (8):

$$\log_2 A = k_a + \log_2 (1 + \widetilde{A}). \tag{8}$$

After this step, k_a is extracted and retained as a characteristic bit. The multiplication can then be transformed into an addition in the logarithmic domain. Performing the algorithmic computation on $A \times B$ yields (9):

$$\log_2 (A \times B) = k_a + \log_2 (1 + \widetilde{A}) + k_b + \log_2 (1 + \widetilde{B}). \tag{9}$$

According to Mitchell's method, (9) can be simplified by approximating the $\log_2 (1 + \widetilde{A})$ and $\log_2 (1 + \widetilde{B})$ as follows:

$$\log_2 (1 + \widetilde{A}) \approx \widetilde{A} \ \Rightarrow \ \log_2 (A) \approx k_a + \widetilde{A}. \tag{10}$$

Due to $(1+\widetilde{A}) < 2$ the curve of $\log_2 (1 + \widetilde{A})$ is close to linear line. The mean error of this step approximation is 5.781%, the deviation of it is 2.524%. Moreover, the maximum error of it is only 8.607%. After these approximations, the logarithms of $A \times B$ is obtained as

$$\log_2 (A \times B) = k_a + k_b + \widetilde{A} + \widetilde{B}. \tag{11}$$

Finally, applying the antilogarithm yields the result of $A \times B$ as follows:

$$\tilde{P} = A \times B \approx \begin{cases} 2^{k_a+k_b}(1 + \tilde{A} + \tilde{B}), & \tilde{A} + \tilde{B} < 1 \\ 2^{k_a+k_b+1}(\tilde{A} + \tilde{B}), & \tilde{A} + \tilde{B} \geq 1. \end{cases} \tag{12}$$

The average relative error in this phase is 3.841%, with a standard deviation of 2.934% and a maximum error of 11.109%.

D. Classical Karatsuba Method

Karatsuba multiplication is a method that reduces the computational complexity of multiplication by achieving sub-quadratic performance, without compromising computational accuracy [15]. Assume that there are two N-bit fixed-point numbers A and B, which are sliced into four fixed-point segments of equal bit width and expressed as

$$A = A_H \cdot 2^k + A_L \tag{13}$$

$$B = B_H \cdot 2^k + B_L, \tag{14}$$

where $k = \frac{N}{2}$, A_H, A_L, B_H and B_L denote the high- and low-bit segment of A and B, respectively. According to (13) and (14), the exact multiplication result of A and B is expressed as

$$\begin{aligned} A \times B &= (A_H \cdot 2^k + A_L) \cdot (B_H \cdot 2^k + B_L) \\ &= \underbrace{A_H B_H}_{m_1} \cdot 2^{2k} + (A_H B_L + A_L B_H) \cdot 2^k + \underbrace{A_L B_L}_{m_0}. \end{aligned} \tag{15}$$

For convenience of subsequent computational expression, $A_H B_H$ and $A_L B_L$ are denoted as m_1 and m_0, respectively. (15) consists four multiplication, $A_H B_H$, $A_L B_L$, $A_H B_L$ and $A_L B_H$. The cross terms can be eliminated by first expressing them collectively as follows:

$$\begin{aligned} m_2 &= \underbrace{(A_H + A_L)}_{a_0} \underbrace{(B_H + B_L)}_{a_1} \\ &= A_H B_H + A_L B_L + (A_H B_L + A_L B_H). \end{aligned} \tag{16}$$

Correspondingly $A_H + A_L$ and $B_H + B_L$ are denoted as a_0 and a_1, respectively. Therefore, $A_H B_L + A_L B_H$ can be written as

$$A_H B_L + A_L B_H = \underbrace{m_2 - m_1 - m_0}_{s_3}. \tag{17}$$

As a result, $A \times B$ can be expressed as follows:

$$A \times B = m_1 2^{2k} + (m_2 - \underbrace{(m_1 + m_0)}_{a_2}) 2^k + m_0. \tag{18}$$

III. Proposed SNN neuron model and Learning algorithm

A. Logarithmic Linear Segmented Multiply

In conventional Mitchell based logarithmic linear approximation, information from the lower bits is typically lost. Separating the computation of the lower-bit components can help reduce approximation errors and improve overall accuracy. Therefore, a Karatsuba-based approach is employed to partition the operands into multiple segments, enabling parallel approximate computations. This strategy maximizes information retention while enhancing parallelism, thereby improving the overall computation speed. Meanwhile, performing a preprocessing step to shift the most significant '1' bit of the operand to the MSB position before subsequent computation can significantly improve accuracy. The preprocessing operation is illustrated in Fig. 1. The number of bits shifted will be recorded, and the result will be shifted back to the right by the same number of bits after the calculation is completed.

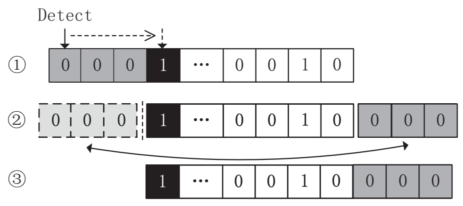


Fig. 1. Preprocess structure of LLSMu.

As shown in Fig. 2, the overall structure of LLSMu is presented. After passing through the preprocess module, the operand is symmetrically partitioned into four smaller-width segments. These segments are then individually summed and

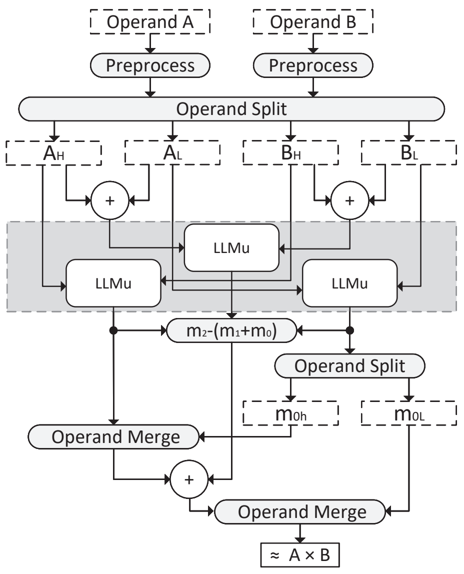


Fig. 2. Simplified logarithmic linear segmented multiply arithmetic operation.

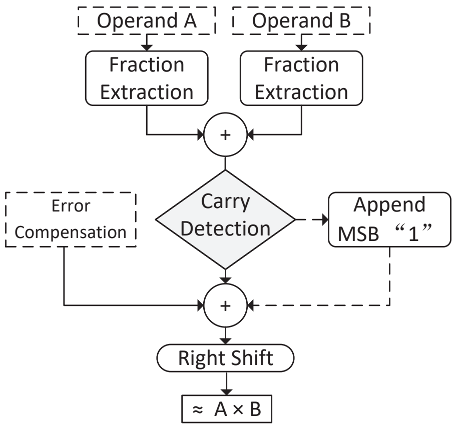


Fig. 3. Simplified logarithmic linear multiply arithmetic operation.

fed into the LLMu module for approximate multiplication. The internal structure of the LLMu module is illustrated in Fig. 3. Upon entering the LLMu module, the fractional parts of the operands are first extracted and summed. A carry detection operation determines whether the sum of the fractional parts exceeds one. This result subsequently influences both the selection of the error compensation constant and the final shift adjustment required for output correction.

The convex nature of the logarithmic function ensures that the linearly approximated multiplication result is consistently lower than the exact value, resulting in a negative error $\tilde{P}-P$. The average error can thus be computed and applied as a constant compensation, which improves overall accuracy without incurring additional hardware resource overhead [14]. Since $\tilde{A}$ and $\tilde{B} \in [0, 1)$, the average error $\tilde{P} - P$ can be obtained by evaluating a double integral over this interval as follows:

$$\begin{aligned}
P &= A \times B = 2^{k_a+k_b}(1+\tilde{A})(1+\tilde{B}) \\
\text{Average}_{\text{Error}} &= \frac{1}{(1-0)(1-0)} \int_0^1 \int_0^1 (\tilde{P}-P)\; d\tilde{A}d\tilde{B} \\
&= -2^{k_a+k_b}\Bigg(\int_0^1 \int_0^{1-\tilde{B}} (\tilde{A}\tilde{B})\, d\tilde{A}d\tilde{B} \\
&+ \int_0^1 \int_{1-\tilde{B}}^1 (1+\tilde{A}\tilde{B}-\tilde{A}-\tilde{B})\, d\tilde{A}d\tilde{B}\Bigg) \\
&\approx -(0.08333) \times 2^{k_a+k_b}.
\end{aligned} \tag{19}$$

Subsequently, the computed $\text{Average}_{\text{Error}}$ is directly added to $\tilde{P}$ as an error compensation term. After processing through three LLMu modules, the values of m_0, m_1, and s_3 are obtained as shown in (18). To minimize the overhead associated with shifting and addition operations, m_0 is symmetrically divided into two k-bit segments: a high-bit part m_{0H} and a low-bit part m_{0L}. The high-bit segment is directly concatenated with m_1, effectively completing both the left-shift of m_1 by k bits and their addition in a single operation. Subsequently, the concatenated result is added to s_3, yielding $m_1 2^k + s_3 + m_{0h}2^{-k}$. Finally, m_{0L} is concatenated with the previous result, completing the final left-shift and addition without introducing additional operations. The complete computation process of LLSMu is summarized in Algorithm 1.

B. LLSMu-based Neuron Model

1) LLSMu-based Izhikevich Neuron Model: Ordinary differential equations (1) need to discretize for hardware implementation. The equations for the membrane potential with n iterations can be obtained as

$$\begin{aligned}
v\,[n+1] =& v\,[n] + \Delta t \cdot (2^{-5} v\,[n]^2 + 2^2 v[n] + v[n] \\
&+ 140 - u[n] + I[n]) \\
u\,[n+1] =& u\,[n] + \Delta t \cdot a(bv[n] - u[n]).
\end{aligned} \tag{20}$$

The proposed design is based on 33-bit fixed-point number (1,10,22), corresponding to 1 sign bit, 10 integer bits, and 22 fractional bits. This design accounts for the possibility that v may occasionally take very small negative values, necessitating a larger integer bit width. Meanwhile, the Izhikevich neuron model typically uses a small time step Δt to accurately capture changes in membrane potential. A 22-bit fractional width is sufficient to support Δt values on the order of 10^{-7}. In addition, fixed-point arithmetic requires significantly fewer hardware resources than floating-point. Splitting $5 \times v[n]$ into $2^2 \times v[n] + v[n]$ allows a multiplication calculation to be replaced by a shift and an addition operation. After this procedure, (20) still contain multiplications by the time constants Δt, parameters a and b, and the constant 2^{-5}. Correspondingly, by approximating these terms with powers of two, their multiplications are replaced by shift operations. Moreover, the coefficient of the $v[n]^2$ is changed from 0.04 to 2^{-5} that is closest 0.04. After all the operations of (20), one multiplication of $v[n]$ remains. Therefore, the finial op-

Algorithm 1 Logarithmic Linear Segmented Multiply

```
Input: $integer\,A[2N],\ integer\,B[2N],\ constant\,c = 0.08333$
Output: $P(A,B)$
 1: $H(A) \leftarrow A[2N-1:N],\ H(B) \leftarrow B[2N-1:N]$
 2: $L(A) \leftarrow A[N-1:0],\ L(B) \leftarrow B[N-1:0]$
 3: $a_0 \leftarrow H(A)+L(A),\ a_1 \leftarrow H(B)+L(B)$
 4: $k_{lA} \leftarrow$ leading one position$(L(A))$
 5: $k_{lB} \leftarrow$ leading one position$(L(B))$
 6: if $\frac{L(A)}{2^{k_{lA}}} + \frac{L(B)}{2^{k_{lB}}} - 2 < 1$ then
 7:     $m_0 \leftarrow 2^{(k_{lA}+k_{lB})} \times (c + \frac{L(A)}{2^{k_{lA}}} + \frac{L(B)}{2^{k_{lB}}} - 1)$
 8: else
 9:     $m_0 \leftarrow 2^{(k_{lA}+k_{lB})} \times (c/2 + \frac{L(A)}{2^{k_{lA}}} + \frac{L(B)}{2^{k_{lB}}} - 2)$
10: end if
11: $k_{hA} \leftarrow$ leading one position$(H(A))$
12: $k_{hB} \leftarrow$ leading one position$(H(B))$
13: if $\frac{H(A)}{2^{k_{hA}}} + \frac{H(B)}{2^{k_{hB}}} - 2 < 1$ then
14:     $m_1 \leftarrow 2^{(k_{hA}+k_{hB})} \times (c + \frac{H(A)}{2^{k_{hA}}} + \frac{H(B)}{2^{k_{hB}}} - 1)$
15: else
16:     $m_1 \leftarrow 2^{(k_{hA}+k_{hB})} \times (c/2 + \frac{H(A)}{2^{k_{hA}}} + \frac{H(B)}{2^{k_{hB}}} - 2)$
17: end if
18: $k_{a0} \leftarrow$ leading one position$(a_0)$
19: $k_{a1} \leftarrow$ leading one position$(a_1)$
20: if $\frac{a_0}{2^{k_{a0}}} + \frac{a_1}{2^{k_{a1}}} - 2 < 1$ then
21:     $m_2 \leftarrow 2^{(k_{a0}+k_{a1})} \times (c + \frac{a_0}{2^{k_{a0}}} + \frac{a_1}{2^{k_{a1}}} - 1)$
22: else
23:     $m_2 \leftarrow 2^{(k_{a0}+k_{a1})} \times (c/2 + \frac{a_0}{2^{k_{a0}}} + \frac{a_1}{2^{k_{a1}}} - 2)$
24: end if
25: $a_2 \leftarrow m_0 + m_1$
26: $s_3 \leftarrow m_2 - a_2$
27: $H(m_0) \leftarrow m_0[2N-1:N],\ L(m_0) \leftarrow m_0[N-1:0]$
28: $P(A,B) \leftarrow (s_3 + m_1 \& H(m_0)) \& L(m_0)$
```

timization is to replace this multiplication with LLSMu. The proposed Izhikevich neuron model is then expressed as

$$\begin{aligned} v\,[n+1] =& v\,[n] + \Delta t \cdot (2^{-5} LLSMu(v\,[n]\,, v\,[n]) + 2^2 v[n] + \\ & v[n] + 140 - u[n] + I[n]) \\ u\,[n+1] =& u\,[n] + \Delta t \cdot a(bv[n] - u[n]). \end{aligned} \tag{21}$$

2) LLSMu-based LIF Neuron Model: The proposed design is based on 16-bit fixed-point number (1,6,9). Since the membrane potential in the LIF neuron model is relatively small and exhibits larger fluctuations compared to the Izhikevich neuron model, lower precision is sufficient. As a result, a 16-bit width is adequate for its representation. And the unique multiplication in the LIF neuron model is replaced by LLSMu. The proposed LIF neuron model can be written as

$$\begin{aligned} V[t+1] =& LLSMu(e^{-\frac{1}{\tau}}, V[t] - E_{rest}) + E_{rest} + I \\ & if\ V[t] \leq V_{th},\ then\ V[t+1] = E_{rest}. \end{aligned} \tag{22}$$

C. LLMu STDP Learning Algorithm

In the proposed design, both τ_j and τ_i in (5) are set to powers of two, allowing division to be replaced with simple bit-shift operations. As a result, all calculations in (5) are multiplication and addition. The multiplication can likewise be replaced by the LLMu. The LLSMu was not adopted for the STDP implementation because the value variations in STDP are relatively small, with most values remaining below 2. In addition, a 5-bit fractional precision is sufficient to represent the synaptic weight changes. Therefore, 8-bit fixed-point numbers $(1, 2, 5)$ are adequate for this requirement. Since LLSMu is optimized for high bit-width operations by processing segmented operands in parallel, it is less suitable for low bit-width scenarios. After applying the LLMu, the STDP calculation after using LLMu is expressed as follows:

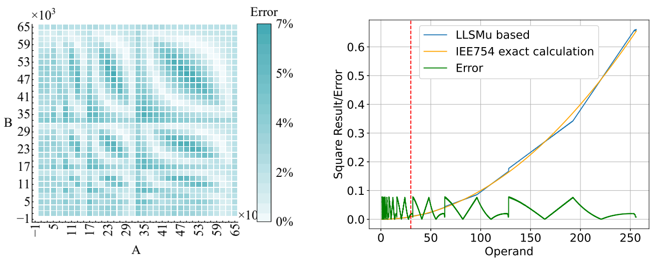
$$\begin{aligned} \Delta W[n] =& LLMu(F_j(\omega[n]), x\tau_i[n]) \cdot s_j[n] \\ & - LLMu(F_i(\omega[n]), y\tau_j[n]) \cdot s_i[n]. \end{aligned} \tag{23}$$

IV. Evaluation of Proposed Designed

The evaluation of the proposed design is conducted progressively across three levels: the multiplier level, the neuron level, and the network level. A Python-based hardware emulator was developed for the LLMu, the LLSMu, the LIF and Izhikevich neuron models, and the STDP learning algorithm. At the network level, three types of networks were constructed and evaluated using different datasets. Among them, the rotor-related dataset was sourced from real experimental data and was successfully used to deploy rotor fault detection in induction motors based on stator current and stray flux signals.

A. Error Analysis of Logarithmic Linear Segmented Multiply

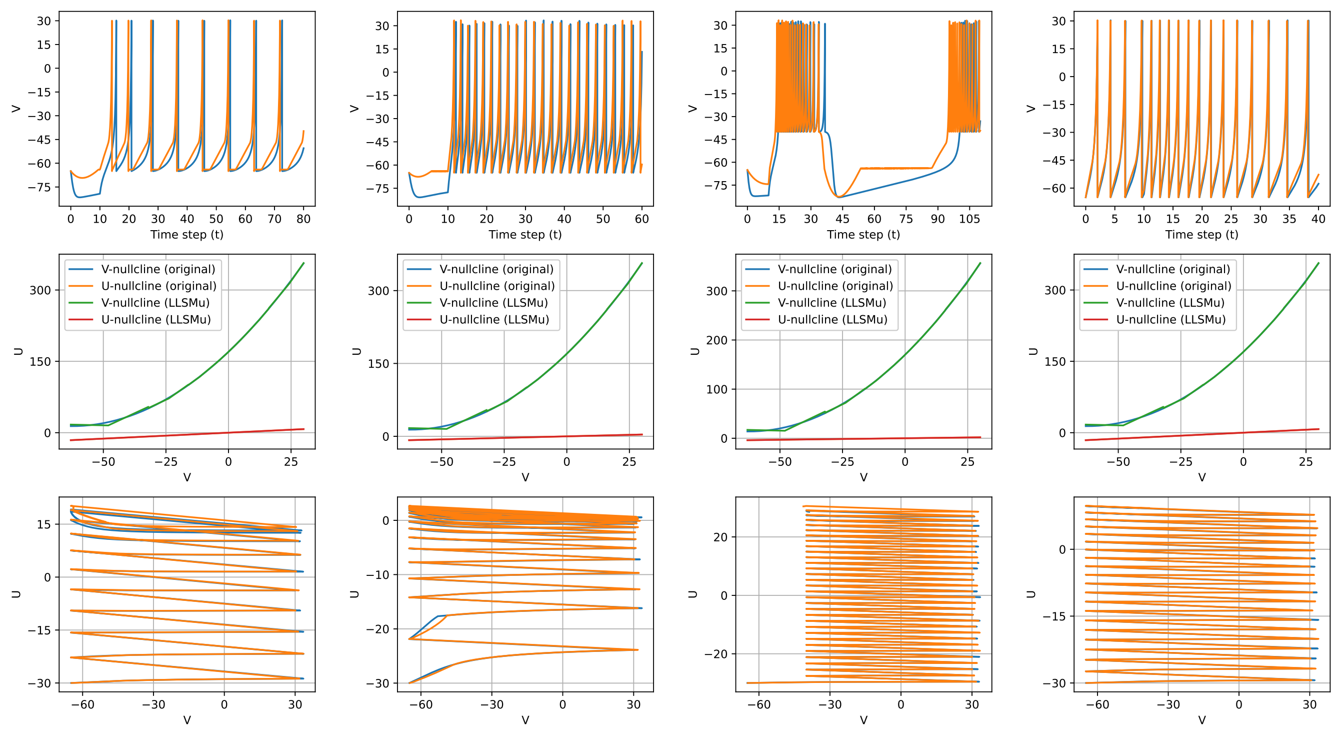
An emulator of the LLSMu has been built, fully replicating the behavior of integer hardware units. Since the bit width is parameterized, the emulator supports testing with arbitrary bit widths. The LLSMu is tested using one million randomly selected operand pairs (A, B), where $A, B \in [0, 2^{16} - 1]$ or $A, B \in [0, 2^{32} - 1]$, corresponding to 16-bit and 32-bit evaluations, respectively. Moreover, the relative error is used, defined as the difference between the approximate result and the IEEE 754 exact result, normalized by the exact result.



(a) Error distribution graph. (b) Square calculation comparison.

Fig. 4. Comparison of LLSMu and exact calculation results. The red line is the dividing line of operand is 30.

The error distribution for the 16-bit evaluation is shown in Fig. 4(a), where lighter-colored areas indicate smaller errors. It can be seen that the area with higher errors is small, so the



(a) Regular Spiking. (b) Fast Spiking. (c) Chattering Spiking. (d) Low Threshold Spiking.

Fig. 5. Comparison of original Izhikevich model with LLSMu-based Izhikevich model. The blue line shows original model and orange line shows the LLSMu-based Izhikevich neuron model. The first row shows the firing patterns. The second row illustrates the nullclines of V and U, while the third row presents the corresponding phase plot. The parameters for neuron model are: (a) a = 2^{-5}; b = 2^{-2}; c = -65; d = 6; I = 35 (b) a = 2^{-3}; b = 2^{-3}; c = -65; d = 2; I = 30 (c) a = 2^{-7}; b = 2^{-4}; c = -40; d = 2; I = 25 (d) a = 2^{-6}; b = 2^{-2}; c = -65; d = 2; I = 40.

average error is only 2.583% which improves 31.4% accuracy compared to Mitchell multiplication. It can also be seen that the peak error is only 7.1% and the error distribution is relatively uniform, resulting in a standard deviation of only 1.83%. Moreover, LLSMu outperforms the algorithm by *Wu* [16] (4.43% average error) in accuracy while being less complex. Additionally, the 32-bit LLSMu test results show an average error of only 2.588%, a standard deviation of 1.86%, and a peak error of 8.3%. The slight loss of accuracy is acceptable given the low complexity. A small standard deviation also proves the stability of LLSMu. Fig. 4(b) shows the comparison of LLSMu calculation results with exact square calculations. The two curves are almost identical. The gap widens for larger operands as the absolute error increases, but the relative error remains small. Moreover, in neuron models and STDP learning, the operand will not exceed 30.

Additionally, the average error and standard deviation of 16-bit and 8-bit LLMu were also evaluated using the same methodology. The 16-bit LLMu achieves an average error of 2.6099% with a standard deviation of 1.85%, and a maximum error of 8.3%. Moreover, the 8-bit LLMu yields an average error of 2.712%, a standard deviation of 1.940%, and a maximum error of 11.1%. Compared to LLMu, LLSMu demonstrates improved accuracy and reduced standard deviation through parallel operations and error compensation.

B. Error Analysis of LLSMu-based Izhikevich Neuron Model

Fig. 5 shows the difference between the original and LLSMu-based Izhikevich model computation results across four representative spiking patterns, where the neuron parameters were selectively configured to demonstrate distinct firing behaviors. Moreover, the parameters a and b were constrained to powers of two to meet hardware design requirements. The figure confirms that the LLSMu-based curves closely align with those obtained from precise computation across all spiking patterns. To more accurately evaluate the performance of LLSMu, the following error metrics are introduced.

Time Error (ERRT): The error in the neuron model may cause the difference in spike timing and lag. With the neuron model parameters as well as the outputs being the same and keeping the neurons of both models synchronized, the time interval between every two spikes in neurons was recorded. The ERRT are calculated as follows:

$$ERRT = \left|\frac{\Delta t_a - \Delta t_o}{\Delta t_o}\right| \times 100\%. \tag{24}$$

In (24), Δt_a denotes the time interval between spikes in the approximate computing neuron model, while Δt_o represents the corresponding interval in the original neuron model.

Normalized Root Mean Square Deviation Error (NRMSD): The NRMSD is used to measure the similarity of spike shapes in approximated and original models. NRMSD is defined as follows:

$$\begin{aligned} RMSD &= \sqrt{\frac{\sum_{i=1}^{n}\left(v_a\left(n\right) - v_o\left(n\right)\right)^2}{n}} \\ NRMSD &= \frac{RMSD}{v_{max} - v_{min}}, \end{aligned} \tag{25}$$

where v_a and v_o are the membrane potential of approximated and original neuron model. v_{max} and v_{min} are the maximum and minimum value in v_o domain.

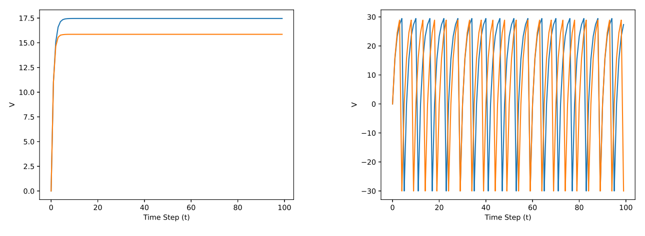
TABLE I
COMPARISON OF NEURON BEHAVIORS BASED ON ERRT AND NRMSD

Neuron	Behavior	ERRT (%)	NRMSD (%)
Izhikevich	Low Threshold Spiking	0.13028	0.5650
	Regular Spiking	0.86890	0.1240
	Chattering Spiking	4.19330	0.6897
	Fast Spiking	0.63818	0.9467
LIF	$\tau = 1$	0	8.2268
	$\tau = 1.5$	16.667	3.02571
	$\tau = 2$	0	0.5387
	$\tau = 2.5$	0	0.0179
	$\tau = 3$	0	0.0045

τ means time constant in LIF neuron model

The smaller the values of ERRT and NRMSD, the closer the model derived from the LLSMu is to the original model. Table I shows that the ERRT is less than 1% for all the patterns except Chattering Spiking which has ERRT more than 1%. The specificity of the chattering Spiking leads to significant error accumulation, resulting in a high but still acceptable error. This indicates that the difference between original and LLSMu-based neuron models is minimal, thereby explaining the near-overlap of the waveforms.

C. Error Analysis of LLSMu-based LIF


(a) $\tau = 1$. (b) $\tau = 1.5$.

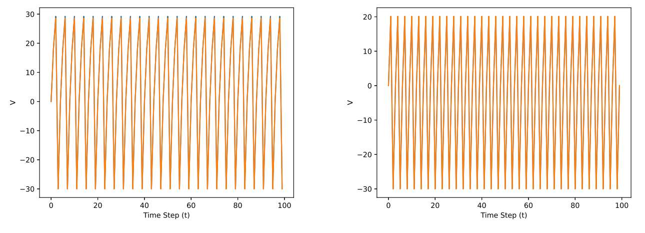

(c) $\tau = 2$. (d) $\tau = 2.5$.

Fig. 6. Comparison of original LIF neuron model with LLSMu-based LIF neuron model. The blue line shows the original model and the orange line shows the LLSMu-based LIF neuron model.

A comparison between LLSMu-based and exact computation results for LIF neurons under different time constants τ is presented. As shown in Fig. 6, when $\tau = 1$, the discrepancy is relatively larger due to the lack of spike generation. As τ increases, the LIF neuron spikes more frequently, and the difference between LLSMu-based and exact implementations becomes negligible.

Table I reports the ERRT and NRMSD values of LLSMu-based LIF neurons under four different τ values. It can be observed that higher error occurs only when τ is small, primarily due to increased spike interval. When τ is sufficiently large, the neuron spikes consistently, resulting in zero ERRT and near-zero NRMSD. In practice, τ is typically chosen to be greater than 2 in SNNs to ensure stable spike generation. Since information in SNNs is encoded in the timing of spikes rather than their amplitude, the LLSMu-induced error does not degrade the performance of LIF neurons when τ is large. Therefore, LLSMu-based LIF neurons performs almost identically to the original in practical scenarios.

D. Error Analysis of LLMu-based STDP Learning Algorithm

An LLMu-based STDP emulator was developed to measure synaptic weight changes between neurons. The weight differences between the exact STDP model and the LLMu-based STDP model were measured under random spike generation from both pre- and post-synaptic neurons, as shown in Fig. 7. To quantitatively evaluate the error, the NRMSD was employed, with the voltage v in the original formula replaced by the synaptic weight change Δw. The resulting NRMSD for the LLMu-based STDP was measured to be 0.761%. Furthermore, Fig. 7 shows that the weight error fluctuates over time but remains consistently low due to the cancellation of alternating positive and negative deviations.

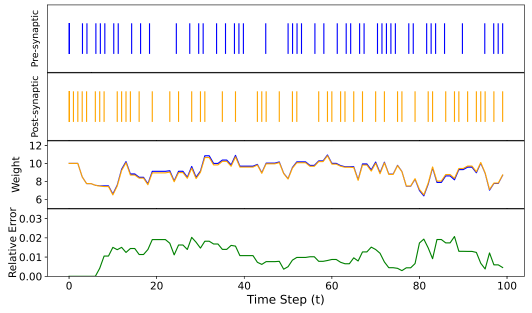


Fig. 7. Comparison of original STDP with LLMu-based STDP. The wight blue line shows the original STDP and the wight orange line shows the LLMu-based STDP.

E. Performance Analysis of LLMu- and LLSMu-based Spiking Neural Network and STDP Learning Algorithm

1) Network Topology: In this study, three different SNN architectures were developed, including a single-layer simple SNN, a two-layer fully connected SNN, and a six-layer deep convolutional SNN (DCSNN).

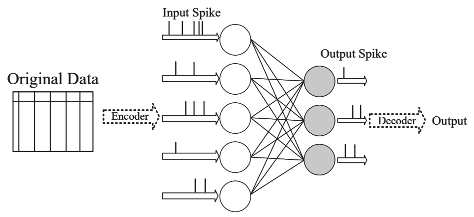


Fig. 8. Structure of 2-layers SNN.

The single-layer network was used for classifying the simple MNIST dataset. Moreover, the architecture of a two-layer fully connected SNN, shown in Fig. 8, is employed for electrical motor diagnostics. The raw data is preprocessed through slicing windowing and Bernoulli rate-based encoding to generate spike maps as input. The DCSNN, built to process more complex task as shown in Fig. 9, consists of three parts. The input part consists of a convolutional layer, batch normalization layer, a max-pooling layer, and an SNN neuron model layer. This stage encodes raw image data into spike-based data through convolutional feature extraction and spiking neuron processing, enabling compatibility with the downstream SNN architecture. The fully connect and output part each contain a fully connect SNN neuron layer.

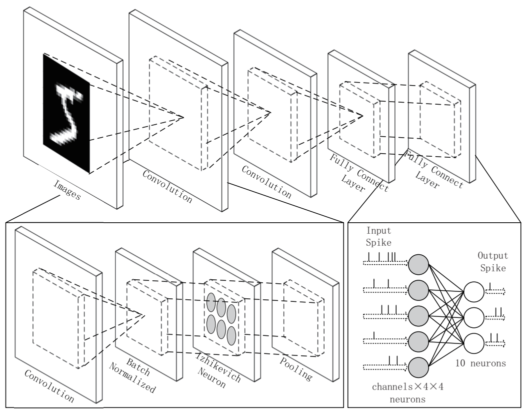
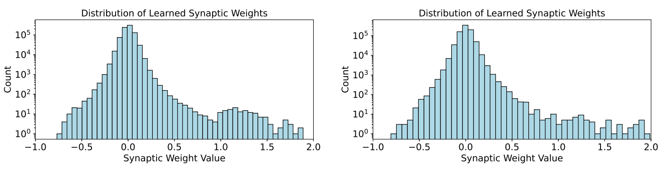


Fig. 9. Structure of 6-layers DCSNN.



(a) Exact STDP. (b) LLMu-based STDP.

Fig. 10. Comparison of original STDP with LLMu-based STDP.

2) Evaluation of LLMu-based STDP Learning Algorithm: A six-layer DCSNN is trained on the Fashion-MNIST dataset using the original and LLMu-based STDP learning algorithm under identical parameter settings, allowing the resulting weight updates to be observed. As shown in Fig. 10, the weight distributions at peak training accuracy are compared. The observed difference is minimal and can be attributed not only to the approximation error introduced by the LLMu method, but also to stochastic factors inherent in SNN training. This demonstrates that the minor approximation errors introduced by the LLMu method do not adversely affect the overall training outcome of the STDP learning algorithm, even in continuous computation scenarios. Additional evaluations were also conducted using various network architectures and datasets to further validate the generality of this result.

3) Evaluation of LLSMu- and LLMu-based SNNs: To evaluate the overall impact of the proposed design on SNN performance, three previously introduced network architectures were implemented with different neuron models and learning algorithms. These architectures were evaluated on the MNIST and Fashion-MNIST datasets, as well as on real-world dataset for rotor fault detection in induction motors using stator current and stray flux signals. The maximum training accuracy of the networks, summarized in Table II, indicate that using LLSMu-based neuron models and LLMu-based STDP algorithms has negligible impact on training accuracy compared to their exact counterparts. The accuracies are nearly identical, and the minor differences observed can be primarily attributed to stochastic factors within the network.

TABLE II
COMPARISON OF PERFORMANCE USING DIFFERENT NEURON MODELS AND LEARNING ALGORITHMS

Learning Algorithm	**Neuron Model**	**Method**	**Accuracy (%)** MNIST 1-layer	F-MNIST 6-layer	Rotor fault 2-layer
Surrogate Gradient Descent	LIF	LLSMu	92.95	97.68	98.13
		Exact	92.94	97.78	98.06
	Izhikevich	LLSMu	90.62	97.88	93.83
		Exact	90.43	98.0	93.91
STDP	LIF	LLSMu	92.95	87.39	98.62
		Exact	92.95	87.30	98.82
	Izhikevich	LLSMu	90.14	88.69	96.02
		Exact	89.94	89.10	95.84
LLMu STDP	LIF	LLSMu	92.94	87.23	98.77
		Exact	92.94	87.56	98.78
	Izhikevich	LLSMu	90.28	88.78	95.80
		Exact	90.07	88.65	95.81

Exact means exact neuron model; LLSMu means LLSMu-based neuron model

V. HARDWARE DESIGN AND IMPLEMENTATION OF LLMU- AND LLSMU-BASED STRUCTURE

A. LLSMu Structure

1) Preprocess Module: The preprocess module scans the input integer from left to right using a series of consecutive XOR gates. When a '1' is detected, its position is recorded, and the input is left-shifted to align the first '1' with the most significant bit (MSB). If no '1' is detected in the input, a control signal is generated to bypass the multiplication process and directly set the output to zero.

2) Logarithms Linear Approximate Computing Module: As shown in block 2 of Fig. 11, the hardware architecture of the LLMu is presented. The input data first passes through a preprocess module to extract the fractional component. The extracted values are then summed, and an overflow flag from the adder is used to determine whether the result exceeds one. Based on the overflow signal, a constant error compensation value is selected and added to the result. Finally, the output is left-shifted to produce the final computation result.

3) Segmented Multiply Module: Fig. 11 illustrates the LLSMu design, which is divided into multiple pipeline stages. The entire LLSMu consists of four components: the approximate multiplier, the error reduction part, the parallel operation, and the adder tree. The first two factors are calculated by a series XOR gate and a shift to be scaled. Next, divide the two new inputs into four parts: AH, AL, BH, and BL.

Subsequently, multiply AH with BH and AL with BL as per the previous formula. At the same time, AH is added to AL and BH is added to BL, after which the sum of the two is multiplied by approximate computing. It is worth noting that, to preserve overflow from addition, the bit width of LLMu in this case is extended by one bit. This process yields values m_0, m_1 and m_2. After this, all factors are only required to do addition, subtraction, and combinatorial operations. The result requires rescaling after two addition and one subtraction operations, as the initial two multiplication factors are shifted.

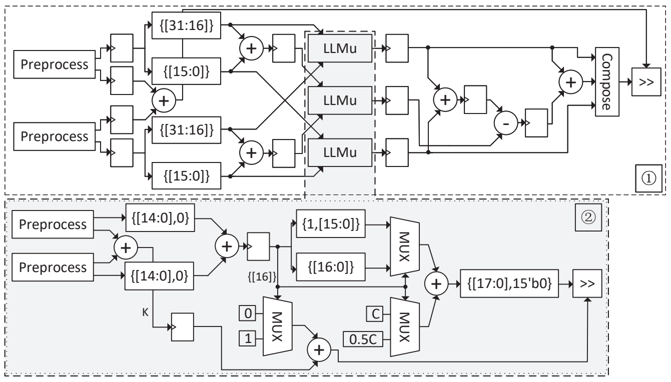


Fig. 11. Block diagram of LLMu and LLSMu. Block 1 illustrates the structure of LLSMu, and Block 2 illustrates the structure of LLMu.

B. Hardware Design of LLMu-based STDP Learning Algorithm

As shown in Fig. 12, the hardware architecture of the LLMu-based STDP implementation is illustrated. The signals S_PR and S_PO, representing the presynaptic spikes s_i and postsynaptic spikes s_j, respectively, are used as control inputs to the multiplexers. These multiplexers determine whether the two polynomial terms involved in the weight update computation are set to zero. Meanwhile, T_PR and T_PO represent $x\tau_i[n]$ and $x\tau_j[n]$, respectively. Since the time constant τ is configured as a power of two, T_PR and T_PO can be efficiently shifted and then processed through a subtractor and an adder to obtain $x\tau_i[n]$ and $x\tau_j[n]$. Afterward, the computed values $x\tau_i[n]$ and $x\tau_j[n]$ are multiplied with F_PR and F_PO, which represent $F_i(\omega[n])$ and $F_j(\omega[n])$, respectively, using the approximate multiplication unit. The difference between the two results yields Δw. Finally, the weight update is completed by adding Δw to the current synaptic weight and storing the result back to the weight register. Furthermore, due to the computational intensity of the STDP algorithm in SNNs, the LLMu-based STDP hardware is implemented using a five-stage pipeline to accelerate weight updates, enhance data throughput, and increase operating frequency.

C. Hardware Design of LLSMu-based Izhikevich Neuron Model

In the hardware design of the Izhikevich neuron model, the two key state variables, u and v, are computed in parallel, as illustrated in Fig. 13. The square of v is calculated using the LLSMu module, followed by a series of addition and subtraction operations to update the values of both u and v. Since

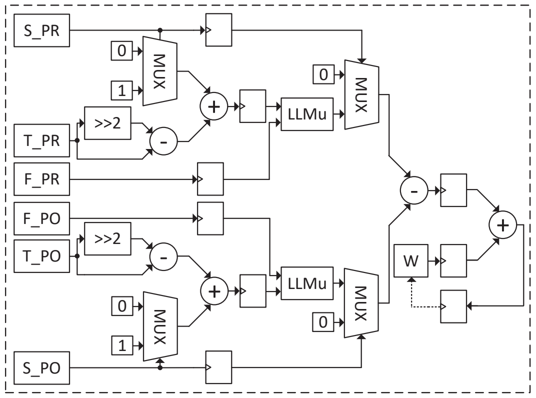


Fig. 12. Block diagram of LLMu-based STDP hardware design.

the parameters a, b, and Δt are approximated as powers of two, their corresponding multiplications are replaced by shift operations to reduce hardware complexity. Finally, two multiplexers are used to check whether v exceeds the threshold. If the threshold is exceeded, v is reset to a predefined value, and u is incremented by d. Because the update of u depends on whether v reaches the threshold, two parallel computation paths are implemented for u, corresponding to each condition. The correct result is then selected by a multiplexer at the output stage to minimize the computation cycle. Furthermore, the whole design is divided into 9-stage pipeline to achieve high operation frequency and data throughput.

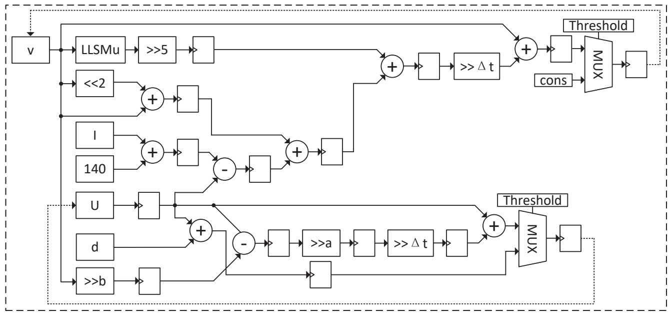


Fig. 13. Block diagram of LLSMu-based Izhikevich neuron model hardware design.

D. Hardware Design of LLSMu-based LIF Neuron Model

The hardware design of the LIF neuron model is shown in Fig. 14. Since $e^{-\frac{1}{\tau}}$ remains constant during computation and may vary depending on the parameter configurations of each task, its precomputed value is directly used as an input. This approach reduces hardware resource consumption while maintaining sufficient flexibility. The subtraction result of $v - E_{\text{rest}}$ and $e^{-\frac{1}{\tau}}$ is fed into the LLSMu approximate computation module, followed by summation with E_{rest} and the input current i to generate the updated membrane potential v. The result is then compared with the threshold value. If the threshold is reached, the multiplexer outputs E_{rest}, and the neuron emits a spike. Otherwise, the result is written back to the internal v register for use in the next iteration. Furthermore, the LIF neuron model is implemented using a 9-stage pipeline to improve data throughput.

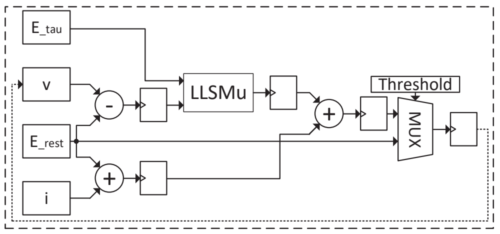


Fig. 14. Block diagram of LLSMu-based LIF neuron model hardware design.

VI. Hardware Implementation Results

The LLSMu-based LIF neuron model, Izhikevich neuron model, and STDP learning algorithm were developed using Verilog HDL and implemented on a Xilinx Zynq UltraScale+ FPGA board. All hardware designs were synthesized and evaluated using Vivado 2023.2. Additionally, these designs were implemented and analyzed under typical process corners (1.00 V, 25 °C) using TSMC's 28 nm technology node. Synopsys Design Compiler, VCS, and PrimeTime PX were employed to perform logic synthesis, dynamic timing simulations, and power analysis for all hardware implementations. Most of the comparative studies on digital implementations of SNN neuron models and STDP learning algorithms report only FPGA-based results. Consequently, ASIC designs for comparison are relatively scarce. For a fair comparison, FPGA and ASIC results are not directly compared. Since SNNs are typically constructed through the repeated implementation of neuron models and learning algorithms, evaluating and comparing the implementation results of a single neuron model and learning algorithm provides valuable insight into the overall network performance when scaled. Moreover, improvements in a single neuron model or learning algorithm will inevitably enhance the overall performance of network deployment.

TABLE III
Comparison of LLSMu with Dadda Multiplier

	Dadda Multiplier	**Booth-AM** [17]	**AM.Cc** [18]	**LLSMu**
FPGA Platform	Zynq UltraScale+	Virtex-7	Virtex-7	Zynq UltraScale+
Bitwidth	32-bit fixed-point	24-bit fixed-point	32-bit fixed-point	32-bit fixed-point
Slice Registers	**96**	NR	NR	393
Slice LUTs	741	**301**	992	661
Max. Freq.	223.3 MHz	91.0 MHz	**331.13 MHz**	302.4 MHz
Throughput	0.8932 GB/s	0.273 GB/s	**1.325** GB/s	1.210 GB/s
Pipeline	2	NR	NR	6
Latency	4.48 ns	10.99 ns	**3.02 ns**	3.31 ns
Power @100MHz	17 mW	NR	NR	**7 mW**
Power Consumption	165.649 pJ/PMR	48.26 pJ/PMR	**33.040 pJ/PMR**	76.061 pJ/PMR
Energy Efficiency	6.04 GOPS/W	20.72 GOPS/W	**30.27 GOP/W**	13.15 GOPS/W
Relative Error	**0**	9.1%	13%	2.583%
Normalized FoM[1]	-	3.384×	2.997×	**1**×

PMR means not Per-Multiplication Result; OPS means Operation per Second; AM.Cc means Approximate Multiplier used in the referenced work; FoM[1] = Area × Error × Latency × Power Consumption, Area includes Slice LUTs

A. *FPGA Implementation of Logarithmic Linear Segmented Multiply*

Table III compares the implementation results of the LLSMu with the Dadda multiplier and several state-of-the-art approximate multipliers, including the booth approximate multiplier (Booth-AM). Compared to the 32-bit Dadda multiplier and the design in [18], LLSMu reduces LUT utilization by 10.8% and 33.4%, respectively. Although it uses more slice registers than the Dadda multiplier, LUTs occupy significantly more area than slice registers, resulting in a net improvement in hardware resource efficiency. In terms of performance, the maximum operating frequency of LLSMu is only 8.7% lower than that of [18], while being 232.3% and 35.4% higher than those of [17] and the Dadda multiplier. Although [17] and [18] consume 36.6% and 56.6% less energy per multiplication compared to LLSMu, this comes at the cost of 352.3% and 503.3% higher computational error, respectively. To provide a more comprehensive evaluation of overall multiplier performance, a composite figure-of-merit (FoM) [24] is adopted. Results show that the FoM[1] of [17] and [18] are 3.384× and 2.997× worse than that of LLSMu.

B. *FPGA Implementation of LLSMu-based Izhikevich Neuron Model*

1) Hardware Overhead Evaluation of LLSMu-based Izhikevich Neuron Model: Table IV demonstrates that the LLSMu-based Izhikevich neuron model requires only 47.16% LUT compared to the FC-CORDIC based Izhikevich neuron. Despite using 41.05% more slice registers than the FC-CORDIC based Izhikevich neuron, the area of 300 slices only requires 7.50% of the 1000 LUTs. Compared to [19], [23] and [20], the lower usage of LUTs and slice registers in their designs is primarily attributed to the use of additional memory resources.

2) Running Speed Evaluation of LLSMu-based Izhikevich Neuron Model: Compared to [24], the LLSMu-based Izhikevich neuron model achieves a 108.33% improvement in maximum operating frequency using operands of the same bit width. When compared to [21] and [23], the frequency is 10.20% and 14.80% higher, respectively. In contrast, it is only 2.26% and 26.20% lower than that of [22] and [20], both of which use operands with lower bit widths. Therefore, the LLSMu-based implementation achieves the highest throughput, with improvements ranging from 21.26% to 108.33%.

3) Energy Efficiency Evaluation of LLSMu-based Izhikevich Neuron Model: The power of the LLSMu-based Izhikevich neuron model is lower than those in [19] and [28], but higher than that reported in [24]. Since the reported power values across studies correspond to different operating frequencies, a fairer comparison is made by evaluating the energy consumption per membrane potential iteration and the corresponding energy efficiency at each design's maximum frequency. The LLSMu-based Izhikevich neuron model achieves the lowest power consumption and highest energy efficiency among all compared designs. Compared to [21] and [24], it reduces energy consumption per membrane potential iteration by 93.82% and 73.12%, respectively, achieving 16.21× and 3.72× improvement in energy efficiency.

4) Accuracy Evaluation of LLSMu-based Izhikevich Neuron Model: The LLSMu-based Izhikevich neuron model exhibits lower NRMSD than most designs, except [19], which reduces error by fitting numerous curve segments at the cost of increased hardware utilization. To better reflect overall performance, the FoM is also adopted. Compared to this work, the FoM[2] of [21] and [24] are 15.3× and 93.1× worse, respectively.

TABLE IV
OVERALL COMPARISON OF LLSMU-BASED IZHIKEVICH NEURON DESIGN AND PRIOR DESIGNS

Method	**PWL** [19]	**ASIZH** [20]	**CORDIC** [21]	**MNIN** [22]	**PWL** [23]	**FC-CORDIC** [24]	**LLSMu**
FPGA Platform	Virtex-6	ZYNQ-7000	Spartan-6	Virtex-2 pro	Virtex-2 pro	ZYNQ-7000	Zynq UltraScale+
FPGA Technology	40 nm	28 nm	45 nm	90 nm	90 nm	28 nm	16 nm
Bitwidth	NR	15-bit fixed-point	30-bit fixed-point	20-bit fixed-point	20-bit fixed-point	33-bit fixed-point	33-bit fixed-point
Slice Registers	**199**	302	280	408	491	648	914
Slice LUTs	230	**188**	469	459	602	1991	939
DSP Block	**0**	NR	**0**	NR	NR	NR	**0**
Memory	BRAM*	RAM*	NR	NR	158 × 8 b	NR	**0**
Max. Freq.	285.00 MHz	**318.00 MHz**	212.80 MHz	240.00 MHz	204.31 MHz	112.60 MHz	234.58 MHz
Throughput	NR	596.25 MB/s	798.00 MB/s	600.00 MB/s	602.50 MB/s	464.48 MB/s	**967.64 MB/s**
Pipeline	3	8	NR	NR	7	10	9
Latency	**10.5 ns**	25.16 ns	28.20 ns	NR	34.26 ns	88.90 ns	38.37 ns
Power	14 mW	NR	71 mW (212.8 MHz)	NR	NR	**4.6 mW (100 MHz)**	12 mW (100 MHz)
Power Consumption	NR	NR	2002.2 pJ/MPI	NR	NR	460 pJ/MPI	**123.63 pJ/MPI**
Energy Efficiency	NR	NR	0.499 GOPS/W	NR	NR	2.174 GOPS/W	**8.089 GOPS/W**
Error (NRMSD)	**0.022%**	0.160%	0.3951%	0.130%	1.540%†	0.940%	0.124%
Normalized FoM^2	NR	NR	15.3×	NR	NR	93.1×	1×

NR means not reported; MPI means Membrane Potential Iteration; OPS means Operation per Second; BRAM* means BRAM was used but not specified; RAM* means RAM was used but not specified; † means the error type is RMSD; FoM^2 = Area × Error × Latency × Power Consumption, Area includes Slice LUTs and Slice registers

TABLE V
OVERALL COMPARISON OF LLSMU-BASED LIF NEURON MODEL DESIGN AND PRIOR DESIGNS

Model	**LIF** [25]	**LIF** [26]	**LIF** [27]	**AdEx LIF** [28]	**AdEx LIF** [29]	**AdEx LIF** [30]	**LLSMu LIF**
FPGA Platform	Virtex-7	Virtex-7	ZYNQ-7000	Spartan-6	Virtex-2 pro	Zynq UltraScale+	Zynq UltraScale+
FPGA Technology	28 nm	28 nm	28 nm	45 nm	130 nm	16 nm	16 nm
Bitwidth	16-bit fixed-point	6-bit fixed-point	NR	20-bit fixed-point	37-bit fixed-point	10-bit fixed-point	16-bit fixed-point
Neuron Number	512	1	922	1	1	1	1
Slice Registers	296	297	3298	829	388	292	**241**
Slice LUTs	314	**196**	4314	1221	1279	363	355
DSP Block	4	NR	**0**	**0**	**0**	NR	**0**
Memory	1003×64 b	BRAM*	13×36 Kb	NR	**0**	NR	**0**
Max. Freq.	NR	100 MHz	100 MHz	134.40 MHz	190 MHz	**500 MHz**	394.01 MHz
Throughput	NR	312.50 MB/s	NR	336 MB/s	**878.75 MB/s**	625 MB/s	788.02 MB/s
Pipeline	8	NR	NR	NR	NR	NR	9
Latency	NR	NR	NR	NR	NR	NR	**22.84 ns**
Power	NR	NR	180 mW	NR	NR	231 mW (500 MHz)	**34 mW (500 MHz)**
Power Consumption	NR	NR	NR	NR	NR	462 pJ/MPI	**68.526 pJ/MPI**
Energy Efficiency	NR	NR	NR	NR	NR	2.16 GOPS/W	**14.59 GOPS/W**
Error (NRMSD)	$4 \times 10^{-12\dagger}$	NR	NR	**0.434%**	2.755%	1.670%	1.721%
Normalized FoM^3	NR	NR	NR	NR	NR	7.2×	1×

NR means not reported; MPI means Membrane Potential Iteration; OPS means Operation per Second; BRAM* means BRAM was used but not specified; † means MSE error type as used in the original work; FoM^3 = Area × Error × Power Consumption, Area includes Slice LUTs and Slice registers

C. FPGA Implementation of LLSMu-based LIF Neuron Model

1) Hardware Overhead Evaluation of LLSMu-based LIF Neuron Model: Table V compares the FPGA implementation results of the LLSMu-based LIF neuron model with those of state-of-the-art LIF neuron models. The LLSMu-based LIF neuron model achieves the lowest slice register utilization among the works in [25]– [30], with a reduction ranging from 17.47% to 92.69%. For LUT utilization, it achieves a reduction of 2.20% to 91.77% compared to [27]– [30], and is only 13.06% and 81.12% higher than that of [25] and [26], respectively. However, both [25] and [26] employ additional memory, and the operand bit width in [26] is only 37.50% of that in the LLSMu-based LIF neuron model.

2) Running Speed Evaluation of LLSMu-based LIF Neuron Model: The LLSMu-based LIF neuron model achieves a 107.37% to 294.01% increase in maximum operating frequency compared to [26]– [29], and is only 21.20% lower than [30], which may be attributed to the smaller operand bit width used in [30]. In terms of throughput, the proposed model outperforms those in [26], [28], and [30] by 26.08% to 152.17%, and is only 10.32% lower than [29].

3) Energy Efficiency Evaluation of LLSMu-based LIF Neuron Model: Compared to state-of-the-art designs, the LLSMu-based LIF neuron model achieves the lowest power, requiring only 14.53% of that in [30] when operated at the same frequency. Furthermore, for each membrane potential iteration, it consumes just 14.83% of the energy used by [30], achieving a 6.75× improvement in energy efficiency.

4) Accuracy Evaluation of LLSMu-based LIF Neuron Model: The NRMSD of proposed design is 1.287% higher than that of [28], but [28] incurs significantly higher hardware resource utilization. In comparison with [30], both designs exhibit similar resource utilization, and [30] achieves only a marginally lower NRMSD by 0.051%. The FoM is also used to evaluate overall performance. Latency is excluded from FoM^3 due to its absence in most referenced designs. The results show that the FoM^3 of [30] is 7.2× worse than that of this work.

D. FPGA Implementation of LLMu-based STDP Learning Algorithm

1) Hardware Overhead Evaluation of LLMu-based STDP Learning Algorithm: Table VI compares the FPGA implementation results of the LLMu-based STDP with several state-of-

TABLE VI
OVERALL COMPARISON OF LLMU-BASED STDP LEARNING ALGORITHM DESIGN AND PRIOR DESIGNS

Model	**C-STDP** [31]	**C-STDP** [19]	**PSTDP** [32]	**TSTDP** [32]	**STDP [33]**	**R-STDP** [34]	**LLMu t-STDP**
FPGA Platform	Virtex-6	Virtex-6	Spartan-6	Spartan-6	Zynq-7000	Zynq UltraScale+	Zynq UltraScale+
FPGA Technology	40 nm	40nm	45 nm	45 nm	28 nm	16 nm	16 nm
Bitwidth	16-bit fixed-point	NR	18-bit fixed-point	18-bit fixed-point	20-bit fixed-point	16-bit fixed-point	8-bit fixed-point
Slice Registers	292	139	642	1370	122	460	**114**
Slice LUTs	309	192	859	1943	911	1435	**148**
DSP Block	**0**	**0**	NR	NR	16	**0**	**0**
Memory	NR	NR	NR	NR	NR	NR	**0**
Max. Freq.	322 MHz	322 MHz	362 MHz	362 MHz	NR	250 MHz	**535.62 MHz**
Throughput	644 MB/s	NR	**814.50 MB/s**	**814.50 MB/s**	NR	500 MB/s	535.62 MB/s
Pipeline	8	NR	NR	NR	NR	NR	5
Latency	24.84 ns	NR	NR	NR	NR	NR	**9.335 ns**
Power	NR	NR	128 mW	222 mW	NR	205 mW	**<3 mW (100 MHz)**
Power Consumption	NR	NR	353.59 pJ/SOP	613.26 pJ/SOP	NR	820 pJ/SOP	**16.80 pJ/SOP**
Energy Efficiency	NR	NR	2.83 GSOPS/W	1.63 GSOPS/W	NR	1.22 GSOPS/W	**59.51 GSOPS/W**
Error (Type)	NR (NR)	0.336% (NRMSD)	5.914 (NMSE)	0.176 (NMSE)	8.230% (max error)	0.0006 (NMSE)	0.761% (NRMSD)

NR means not reported; SOP means Spike Operations

the-art STDP design. Compared to [31], [19], and [34], the LLMu-based STDP achieves the lowest utilization of slice registers and LUTs, with reductions ranging from 6.56% to 91.68% for slice registers and from 22.92% to 92.41% for LUTs, without using any DSPs or additional memory.

2) Running Speed Evaluation of LLMu-based STDP Learning Algorithm: The LLM-based STDP achieves the highest operating frequency among the designs in [31]– [34] and [19], reaching 156.25% to 226.25% of their reported values. Although its operand bit width is smaller, resulting in a lower throughput, it is still only 34.24% lower than that of [32]

3) Energy Efficiency Evaluation of LLMu-based STDP Learning Algorithm: Compared to [32] and [34], this work achieves the lowest power consumption. However, since the operating frequencies at which their power measurements were obtained are not reported, a fair comparison is made based on energy per weight update and energy efficiency. Under this metric, the LLMu-based STDP reduces energy consumption per weight update by 95.25% to 97.95%, achieving a 21.03× to 48.78× improvement in energy efficiency.

4) Accuracy Evaluation of LLMu-based STDP Learning Algorithm: The NRMSD of the LLMu-based STDP is 0.425% higher than that of [19], but the error remains minimal. Due to differences in error calculation methods and the lack of sufficient data in referenced work, a direct comparison using the FoM is not feasible.

E. ASIC Implementation

1) ASIC Implementation of LLSMu-based Izhikevich Neuron Model: Table VII presents a comparison between the ASIC implementation of the LLSMu approach and representative state-of-the-art CORDIC and PWL approaches. Even with larger operand bit widths, it still achieves a 95.56× improvement in maximum operating frequency and a 143.48× increase in throughput. Moreover, it reduces area by 71.30% to 76.64% and improves energy efficiency by 5.58× to 5.69×. The error is also reduced by up to 0.287%. In addition, the FoM^2 of the two designs in [35] are 455.49× and 268.96× worse than that of this work. While part of these improvements can be attributed to a more advanced technology node, the results also demonstrate the effectiveness of LLSMu in enabling higher frequency, lower power consumption, and reduced chip area.

TABLE VII
ASIC RESULT COMPARISON - IZHIKEVICH

Method	**CORDIC** [35]	**PWL** [35]	**LLSMu**
Node	130 nm	130 nm	28 nm
Voltage	NR	NR	1.0 V
Bitwidth	22-bit fixed-point	22-bit fixed-point	33-bit fixed-point
Frequency	9.1 MHz	9.1 MHz	**869.6 MHz**
Latency	109.89 ns	109.89 ns	**10.35 ns**
Throughput	0.025 GB/s	0.025 GB/s	**3.587 GB/s**
Area	25894 μm^2	21076 μm^2	**6048.126 μm^2**
Power	0.33 mW	**0.3 mW**	5.649 mW
Power Consumption	36.260 pJ/MPI	36.970 pJ/MPI	**6.496 pJ/MPI**
Energy Efficiency	27.580 GOPS/W	27.050 GOPS/W	**153.941 GOPS/W**
Error (ERRT)	1.560%	1.110%	**0.869%**
Normalized FoM^2	455.49×	268.96×	1×

NR means not reported; MPI means Membrane Potential Iteration; OPS means Operation per Second; FoM^2 = Area × Error × Latency × Power Consumption

TABLE VIII
ASIC RESULT COMPARISON - LIF

Method	**QUANTISENC** [36]	**Multiplexing** [37]	**LLSMu**
Node	32 nm	28 nm	28 nm
Voltage	NR	0.9	1.0 V
Bitwidth	8-bit fixed-point	12-bit fixed-point	16-bit fixed-point
Neuron number	1	4096	1
Frequency	0.1 GHz	0.3 GHz	**1.5625 GHz**
Latency	10 ns	320 ns	**5.12 ns**
Throughput	0.1 GB/s	NR	**3.125 GB/s**
Area	2894 μm^2	2160000 μm^2	**1291.37 μm^2**
Power	**0.102 mW**	NR	2.003 mW
Power Consumption	**1.02 pJ/MPI**	3.60 pJ/MPI	1.28 pJ/MPI
Energy Efficiency	**980.39 GOPS/W**	277.78 GOPS/W	780.03 GOPS/W
Error (RMSD)	0.43	NR	1.03
Normalized FoM^2	1.46×	NR	1×

NR means not reported; MPI means Membrane Potential Iteration; OPS means Operation per Second; FoM^2 = Area × Error × Latency × Power Consumption

2) ASIC Implementation of LLSMu-based LIF Neuron Model: Table VIII compares the ASIC implementation of the LLSMu-based LIF neuron model with state-of-the-art approaches. It achieves a 5.21× to 15.63× improvement in maximum frequency, 31.25× higher throughput, and reduces latency and area to 51.20% and 44.62% of [36], respectively. Although its energy efficiency is 20.44% lower, this is likely due to the 50% smaller operand bit width used in [36], which leads to lower energy consumption per membrane potential iteration. Additionally, while [36] achieves a 0.6% lower error than this work, the overall performance evaluation using FoM^2 shows that it still performs 1.46× worse.

TABLE IX
ASIC RESULT COMPARISON - STDP

Method	**CLSTDP** [38]	**ImSTDP** [38]	**LLMu tSTDP**
Node	22 nm	22 nm	28 nm
Voltage	NR	NR	1.0 V
Bitwidth	8-bit fixed-point	8-bit fixed-point	8-bit fixed-point
Frequency	1.000 GHz	1.250 GHz	**1.538 GHz**
Latency	3 ns	**2.40 ns**	3.25 ns
Throughput	0.387 GB/s	0.787 GB/s	**1.538 GB/s**
Area	1771.0 μm^2	1294.5 μm^2	**655.074 μm^2**
Power	1.68 mW	1.33 mW	**0.89 mW**
Power Consumption	3.453 pJ/SOP	2.137 pJ/SOP	**0.579 pJ/SOP**
Energy Efficiency	289.60 GSOPS/W	467.90 GSOP/W	**1727.12 GSOPS/W**
Error (NRMSD)	NR	11.961%*	0.761%
Normalized FoM^2	NR	84.65×	1×

NR means not reported; SOP means Spike Operations; * means not provided originally, but derived via reimplementation; FoM^2 = Area × Error × Latency × Power Consumption

3) ASIC Implementation of LLMu-based STDP Learning Algorithm: Table IX compares the ASIC implementation of the LLMu-based STDP with state-of-the-art approaches. Under the same operand bit width and a less advanced technology node, the LLMu-based design still achieves a 53.80% and 23.04% improvement in maximum operating frequency, along with 3.98× and 1.95× higher throughput compared to CLSTDP and ImSTDP, respectively. Its area is reduced to only 36.99% and 50.60% of that of CLSTDP and ImSTDP. In terms of power efficiency, it outperforms both designs by 5.96× and 3.69×, respectively, while latency is the only metric where it is slightly higher, by 35.40% and 8.33%. Moreover, compared to ImSTDP, this work achieves 11.2% lower error and delivers 84.65× better overall performance, as measured by the FoM^2.

VII. Conclusion

This paper presents two novel approximate computing designs, Logarithmic Linear Multiply (LLMu) and Logarithmic Linear Segmented Multiply (LLSMu), targeted at neuron models and learning algorithms in Spiking Neural Networks. The proposed LLSMu achieves high computational accuracy, with a tested error rate of only 2.583% . Due to its multiplication-replacement structure, the method is broadly applicable to a wide range of SNN neuron models and learning algorithms. The LLSMu was deployed and tested within the Izhikevich and LIF neuron models, as well as the STDP learning algorithm. Both neuron-level and network-level evaluations were conducted, and the results demonstrate that the LLSMu-based implementations maintain the same accuracy and spiking patterns as their exact counterparts. Hardware implementation on the FPGA and ASIC further evaluated the design, showing extremely high energy efficiency, while achieving low hardware resource utilization, and high data throughput. This makes the approach well-suited for energy- and resource-constrained platforms such as edge devices, enabling large-scale SNN deployment with lower hardware and energy requirements, along with improved computational speed.

Acknowledgments

The authors would like to thank Mr. Lingyou Zhou for the valuable writing suggestions that helped improve the quality of this manuscript.

References

[1] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659-1671, Dec. 1997.
[2] Y. Liu *et al.*, "FPGA-NHAP: A General FPGA-Based Neuromorphic Hardware Acceleration Platform With High Speed and Low Power," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 69, no. 6, pp. 2553-2566, Mar. 2022.
[3] J. L. Lobo *et al.*, "Spiking Neural Networks and Online Learning: An Overview and Perspectives," *Neural Netw.*, vol. 121, pp. 88-100, July 2019.
[4] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *J. Physiol.*, vol. 117, no. 4, pp. 500-544, Aug. 1952.
[5] E.M. Izhikevich, "Which model to use for cortical spiking neurons?" *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1063-1070, Sept. 2004.
[6] B. Joo, J. Han and B. Kong, "Energy- and Area-Efficient CMOS Synapse and Neuron for Spiking Neural Networks With STDP," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 69, no. 9, pp. 3632-3642, Sept. 2022.
[7] H.Cho *et al.* "An On-Chip Learning Neuromorphic Autoencoder with Current-Mode Transposable Memory Read and Virtual Lookup Table," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 161-170, Feb. 2018.
[8] J. Wu *et al.*, "Efficient Design of Spiking Neural Network With STDP Learning Based on Fast CORDIC," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 68, no. 6, pp. 2522-2534, June 2021.
[9] P. Michael and P. Thomas, "Deep Learning With Spiking Neurons: Opportunities and Challenges," *Front. Neurosci.*, vol. 12, pp. 409662, Oct. 2018.
[10] E. Z. Farsa *et al.*, "A Low-Cost High-Speed Neuromorphic Hardware Based on Spiking Neural Network," *IEEE Trans. Circuits Syst. II-Express Briefs*, vol. 66, no. 9, pp. 1582-1586, Sept. 2019.
[11] Z. Peng *et al.*,"A High-Accuracy and Energy-Efficient CORDIC based Izhikevich Neuron," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2021, pp. 1-4.
[12] A. Morrison, M. Diesmann and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," *Biol. Cybern.*, vol. 98, no. 6, pp. 459-478, June 2008.
[13] J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4, pp. 512-517, Aug. 1962.
[14] H. Saadat, H. Bokhari and S. Parameswaran, "Minimally biased multipliers for approximate integer and floating-point multiplication," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 37, no. 11, pp. 2623-2635, Nov. 2018.
[15] A. Karatsuba, Y. Ofman*et al.*, "Multiplication of Multidigit Numbers on Automata," *Soviet Physics Doklady*, vol. 7, no. 7, pp. 595, Dec. 1962.
[16] X. Wu *et al.*, "Design of Energy Efficient Logarithmic Approximate Multiplier," in *2023 5th International Conference on Circuits and Systems (ICCS)*, 2023, pp. 129-134.
[17] S. Ullah *et al.*,"Area-Optimized Accurate and Approximate Softcore Signed Multiplier Architectures," *IEEE Trans. Comput.*, vol. 70, no. 3, pp. 384-392, Mar. 2021.
[18] S. Ullah *et al.*,"High-Performance Accurate and Approximate Multipliers for FPGA-Based Hardware Accelerators," *IEEE Trans. Comput-Aided Des. Integr. Circuits Syst.*, vol. 41, no. 2, pp. 211-224, Feb. 2021.
[19] E. Jokar, H. Abolfathi and A. Ahmadi , "A Novel Nonlinear Function Evaluation Approach for Efficient FPGA Mapping of Neuron and Synaptic Plasticity Models," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 2, pp. 454-469, Apr. 2019.
[20] P. V. Sruthi and T. S. Bindiya, "Modeling of Izhikevich Spiking Neurons With High Matching Accuracy Using Comb-Exp Functions and Its Multiplier-Free Realization," *IEEE Trans. Circuits Syst. I: Regular Papers*, pp. 1-0, Apr. 2025.
[21] M. Heidarpur *et al.*, "CORDIC-SNN: On-FPGA STDP learning with Izhikevich neurons," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 66, no. 7, pp. 2651-2661, July 2019.
[22] S. Haghiri *et al.*, "Multiplierless implementation of noisy Izhikevich neuron with low-cost digital design," *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 6, pp. 1422-1430, Dec. 2018.
[23] H. Soleimani *et al.*, "Biologically inspired spiking neurons: Piecewise linear models and digital implementation," *IEEE Trans. Circuits Syst. I: Regular Papers*, vol. 59, no. 12, pp. 2991-3004, Dec. 2012.
[24] J. Wang *et al.*, "A High-Accuracy and Energy-Efficient CORDIC Based Izhikevich Neuron with Error Suppression and Compensation," *IEEE Trans. Biomed. Circuits Syst.*, vol. 16, no. 5, pp. 807-821, Oct. 2022.

[25] J. Kim *et al.*, "Hardware-Efficient Emulation of Leaky Integrate-and-Fire Model Using Template-Scaling-Based Exponential Function Approximation," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 68, no. 1, pp. 350-362, Jan. 2021.

[26] W. Guo *et al.*, "Toward the Optimal Design and FPGA Implementation of Spiking Neural Networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 3988 - 4002, Feb. 2022.

[27] A. Carpegna, A. Savino and S. D. Carlo, "Spiker+: a framework for the generation of efficient Spiking Neural Networks FPGA accelerators for inference at the edge," *IEEE Trans. Emerg. Top. Comput.*, pp. 1-15, Dec. 2024.

[28] M. Heidarpour, A. Ahmadi and R. Rashidzadeh, "A CORDIC Based Digital Hardware for Adaptive Exponential Integrate and Fire Neuron," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 63, no. 11, pp. 1986-1996, Nov. 2016.

[29] S. Gomar, A. Ahmadi, "Digital multiplierless implementation of biological adaptive-exponential neuron model," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 61, no. 4, pp. 1206-1219, Apr. 2014.

[30] S. Xiao *et al.*, "Low-Cost Adaptive Exponential Integrate-and-Fire Neuron Using Stochastic Computing," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 5, pp. 942-950, Oct. 2020.

[31] E. Jokar and H. Soleimani, "Digital Multiplierless Realization of a Calcium-Based Plasticity Model," *IEEE Trans. Circuits Syst. II-Express Briefs*, vol. 64, no. 7, pp. 832-836, July 2017

[32] C. Lammie *et al.*, "Efficient FPGA Implementations of Pair and Triplet-Based STDP for Neuromorphic Architectures," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 66, no. 4, pp. 1206-1219, Apr. 2019.

[33] A. Manoharan, G. Muralidhar and B. J. Kailath, "A Novel Method to Implement STDP Learning Rule in Verilog," in *2020 IEEE Region 10 Symposium (TENSYMP)*, 2020, pp. 1779-1782.

[34] C. Yang, K. Wang and A. Wang, "A Fully Digital Implementation of A Reward-Modulated STDP Synapse," in *2023 5th International Conference on Robotics, Intelligent Control and Artificial Intelligence (RICAI)*, 2023, pp. 894-899.

[35] A. Elnabawy *et al.*, "A Low Power CORDIC-Based Hardware Implementation of Izhikevich Neuron Model," in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2018, pp. 130-133.

[36] S. Matinizadeh *et al.*, "A Fully-Configurable Open-Source Software-Defined Digital Quantized Spiking Neural Core Architecture," *arXiv*, Apr. 2024.

[37] M. Sadeghi *et al.*, "NEXUS: A 28nm 3.3pJ/SOP 16-Core Spiking Neural Network with a Diamond Topology for Real-Time Data Processing," *IEEE Trans. Biomed. Circuits Syst.*, pp. 1-13, Aug. 2024.

[38] D. Zhao *et al.*, "ImSTDP: Implicit Timing On-Chip STDP Learning," *IEEE Trans. Circuits Syst. I-Regul. Pap.*, vol. 72, no. 2, pp. 868-881, Feb. 2025.

Haihang Xia received the M.Sc. degree in Electronic and Electrical Engineering from University of Glasgow, Glasgow, U.K., in 2023. He is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering, University of Sheffield, Sheffield, U.K. His research interests include Spiking Neural Networks, neuromorphic computing, AI acceleration, and digital systems design.

Haotian Liu received the M.Sc. degree in microelectronic system design from the University of Southampton, U.K. He is currently pursuing the Ph.D. degree with The University of Sheffield. His research interests include high-performance computing, sparse network acceleration, and architectural support for parallel programming.

Yuqin Zhao received the B.E. degree in Electrical and Computer Engineering from Tamkang University, Taiwan, in 2021, and the M.Sc. degree in Electronics from Nanyang Technological University, Singapore, in 2022. He is currently pursuing a Ph.D. degree in Electronic and Electrical Engineering at the University of Sheffield, UK. His research interests include FPGA-based acceleration for AI and software-defined radio (SDR), SDR system implementation, and the development of high-level synthesis tools for AI and SDR applications on FPGAs.

John Goodenough received the B.Sc. degree in Engineering Science from the University of Durham, U.K., in 1988, and the Ph.D. degree in VLSI Architecture from the University of Sheffield, U.K., in 1991. He is currently a Professor with the School of Electrical and Electronic Engineering, the University of Sheffield, U.K. He is also an Independent Consultant with textrium.io, U.K., a Technology Advisor with Silicon Catalyst, U.S.A., and a member of the U.K. Government Semiconductor Advisory Panel. He previously held multiple senior positions at Arm, U.K. and U.S.A. (2000–2022), including Distinguished Engineer Architect for Secure SoC, Vice President of Engineering Systems, Vice President of Research Collaboration, and Director of Design Technology and Automation. He was also a board member of SRC Research Scholars Program, U.S.A. (2019-2022). Earlier, he was Director of Technology at Infinite Designs Ltd., U.K. (1995–2000), and a Research Associate and Lecturer at the University of Sheffield, U.K. (1991–1995).

Charith Abhayaratne (Member, IEEE) is a Senior Lecturer with the School of Electrical and Electronic Engineering, The University of Sheffield. He received the BE and PhD in from The University of Adelaide in 1998 and from the University of Bath in 2002, respectively. He was a recipient of the ERCIM Post-Doctoral Fellowship (2002-2004) for his research at the Centre for Mathematics and Computer Science (CWI), The Netherlands, and the National Research Institute for Computer Science and Control (INRIA), France. He has served as an associate editor for IEEE Transactions on Image Processing (2019-2024), IEEE Access (2020-2025) and Elsevier JISA (2019-2025).

Sichen Liu received the B.E. degree from Shandong University of Science and Technology, Qingdao, China, in 2024. He is currently pursuing a Ph.D. degree at the University of Sheffield, U.K. His research interests include Transformer acceleration and sparse network acceleration through hardware–software co-design.

Sizhao Li (Member, IEEE) received the B.S. degree from Jilin University, Changchun, China, in 2007, the M.S. degree from the University of Science and Technology of China, Hefei, China, in 2011, and the Ph.D. degree in electrical engineering from Xiamen University, Xiamen, Fujian, China, in 2018. In 2018, he joined the College of Computer Science and Technology, Harbin Engineering University, Harbin, Heilongjiang, China, as an Associate Professor. He was a Visiting Scholar with the University of Illinois at Urbana–Champaign, Champaign, IL, USA. He is also the Vice Director of the Intelligent Computing and Industrial Internet Security Center, College of Computer Science and Technology, Harbin Engineering University. His interested research areas include high-performance parallel computing, computer architecture, and design of artificial intelligence system.

Tiantai Deng received the B.Eng. degree from Harbin Institute of Technology, Harbin, China, in 2015, and the Ph.D. degree from Queen's University Belfast, U.K., in 2019. He is currently a Lecturer with The University of Sheffield, U.K. Prior his career as an academic, he was a Senior Engineer with HiSilicon, Huawei. His main research interests include hardware acceleration for image processing, deep learning, and high-level design environments.