ASPEN: Approximate Spike Encoding Neuron

***Abstract*—Spiking Neural Networks (SNNs) are a neural network architecture that encodes information as a time series of binary pulses, or spikes. SNNs are a promising method to perform energy efficient on-chip neural network execution, as spike encoding removes the need for expensive multiplication operations. SNN neuron updates commonly include a thresholding phase to produce binary spikes, which ensures that SNNs are resilient to computational errors. This paper proposes the Approximate Spike Encoding Neuron, (ASPEN), a spiking neuron design that aims to reduce energy consumption using approximate computing techniques. The first major contribution of ASPEN presents an optimized design for performing sequential weight summation using counter compressors and block-save adder networks. This design leverages the event-driven nature of spiking networks to improve energy-efficiency. The second major contribution of this paper analyzes the effect of approximate computing on neuron level design metrics and overall accuracy. We present a novel technique to approximate adder trees by introducing forced errors to collapse trees to lower depth networks. The proposed designs are implemented using 45nm technology. An SNN implemented with ASPEN achieves 98.42% accuracy on the MNIST digit database with the neuron computation consuming 43.9-48.8% less energy per inference than alternative spiking neuron architectures.**

***Index Terms*—spiking neural networks, deep neural networks, multi-input addition, approximate computing**

I. Introduction

There is a growing desire in a number of fields to implement neural network (NN) algorithms on embedded hardware. The explosion of digital data available in the world has greatly increased the need for high-performance tools to process and analyze that data. Modern NNs have achieved remarkable performance on a number of analysis tasks, and embedded systems are already integrating NNs into a number of developing technologies, from IoT and wearable devices, to medical and industrial scanners.

Spiking neural networks (SNNs) are an emerging architecture commonly referred to as the third generation of NNs. SNNs are a neuromorphic-inspired architecture that aims to emulate how the human brain processes information. Information is passed between neurons using time-varying binary pulses, or spikes, instead of traditional numerical information. This allows SNN computation to avoid the use of expensive multiplication units altogether and instead use simple addition and thresholding operations. SNNs are an active area of research, and have demonstrated cutting edge performance on MNIST [1], and a variety of other benchmarks [2].

SNNs are not without problems, however. To achieve similar levels of performance as NNs, SNNs typically require larger and deeper networks. The spike encoding causes SNNs to be inherently event driven, which causes irregular memory access patterns. Furthermore, since data is time-varying, networks need multiples passes to get accurate results. These factors together require SNNs to have larger memory footprints and perform more, if simpler, computations than traditional network architectures. Running SNNs on commercial hardware platforms, such as GPUs, is often difficult [16] as these systems are optimized for regular memory access. As the demands and performance expectations for NNs grow, addressing these issues is critical for enabling the future adoption of SNN systems.

Approximate computing is a general optimization technique that introduces a cost-accuracy trade-off to circuit designs. This technique has proven effective for improving traditional neural networks [11], [12]. Neural networks have been shown to be resilient to computational approximation [17]. SNNs are a promising target for approximate computing optimizations. Since spikes are generated based on thresholding the result of a computation, an approximate computation can produce the same spiking pattern as an exact computation as long as the error is not large enough to cross the threshold unintentionally. Furthermore, since spiking data is time dependent, minor time-shifts in internal spike trains may still produce correct final results.

In this work, we explore a novel direction for SNN circuit optimization by examining the spiking neuron itself, and present the Approximate Spike Encoding Neuron (ASPEN). The neuron update is a significant computational operation, often requiring summation over a large number of input weights, and therefore this update operation will be the focus of our proposed design. Our contributions are as follows:

- We present a modular weight summation unit design that is optimized for sequential, event-driven inputs. This unit combines counter compression and block-save adder networks to perform efficient multi-input addition.
- We present a technique to approximate arbitrary adder tree networks. This technique strategically chooses signals to fix in order to simplify a tree structure and reduce the tree's depth. This introduces errors with know magnitude and sign. We show a method to introduce overlapping errors with opposite signs that reduces the expected error magnitude.
- We present an energy-efficient approximate spiking neuron design which replaces exact modules in our weight summation unit with approximate modules. We analyze the effect of the approximate neuron on network accuracy by emulating the approximate hardware in a software simulation run on the MNIST dataset.
- We demonstrated superior energy consumption over alternative weight summation architectures at both the neuron and network level.

II. Background and Related Works

A. Spiking Neural Networks Circuits

In recent years, there has been a shift in focus towards designing and building custom hardware units optimized specifically to execute SNN algorithms. One approach has to been to build computers that run SNN algorithms at a massive scale [2], [18]. These systems are focused on providing tools to researchers and professionals to easily test novel spike-based algorithms. These systems have demonstrated impressive performance gains on a variety of baselines and problems, and have definitively proven the value of spike-based computing.

Concurrent development has made progress towards developing small-scale and highly-optimized SNN processor chips [3], [4]. These proposed designs focus on providing spiking networks for real-world embedded systems and edge devices. These systems often focus on designing efficient routing structures to handle the memory access challenges. Some of these concepts and architectures have even been implemented in commercial products [19].

B. Approximate Neural Network Circuits

Neural networks have been shown to be resilient to computational approximations [17]. This has led to a good deal of research focused on building approximate NN circuits. These projects can be divided into two categories based on the method used: 1) Network level approximations, and 2) neuron level approximations. Network level approximations focus on reducing the neural network to a simpler form [17]. Neuron level methods focus on using approximate computing circuits in the actual neuron computation [11], [12], [13]. These methods aim to introduce errors at the expense of accuracy in order to improve performance.

Some work in recent years has attempted to apply network level approximation techniques to spiking neural networks. Some work have directly applied network reduction strategies to SNNs [5], [6]. Other works have attempted to use the unique computational behavior of SNNs, particularly their time-dependent nature, to dynamically adjust the network during inference [4]. To our current knowledge, no previous works have attempted to apply neuron level approximate methods to SNNs beyond simple weight quantization.

C. Multi-Input Addition

Multi-input addition is an old field with a variety of proposed solutions. Thabah *et al.* demonstrate experimentally that Wallace Trees are generally the most efficient multi-input design [7]. There are structural reasons why Wallace trees are not well suited for implementation in SNNs. Wallace trees require a fixed number of inputs that are all present at the same time. This setup is ill-suited for SNNs, as weights summed during a neuron update are variable in number and are accessed irregularly.

Instead, an multi-input design that receives inputs sequentially would be better suited for the spiking behavior of SNNs. The most common accumulator design is a carry-save accumulator [20]. This design feeds an input into a carry-save adder and latches the output to be fed back into the system. Once all inputs have been processed, a final addition is applied to complete carry propagation. This system is compact, but consumes a lot of energy when run multiple times. Input compression is an alternative low cost method to accumulate sequential information [9]. The premise is to find the number of ones in each significance column of the inputs, and perform arithmetic on more compact counted representation. The idea here is to replace a series of medium-cost operations with a lot of low-cost operations and one big-cost operation.

III. Spiking Neural Network Implementation

The first step in developing an SNN accelerator design is choosing a network architecture. This work uses the training method developed by Diehl *et al.* [1]. This network is applied to the MNIST handwriting dataset, and uses a $784\times1200\times1200\times10$ feed-forward structure. This network uses a weight normalization stage to bound all weights between [-1,1]. Normalization allows us to use the same neuron circuit for different layers, and also reduces the inference time required for accurate results significantly. ASPEN runs its SNN through 10 simulated time steps. This network achieves an MNIST accuracy of 98.46%. This model is simple to train and implement in software, but it also confers a number of advantages when converted to a hardware implementation.

The simplest and most common neuron model used is the integrate-and-fire neuron (IF). See Fig. 1 for a schematic view of this neuron model. There are two phases to performing an IF neuron update. The first phase starts by accessing the synaptic weights of all n neurons in the previous layer that have fired in the current time step. It then sums these weights together with the neuron's current membrane potential, V_in, to form the synaptic impulse. In the second phase, this impulse is compared against some threshold value. If the threshold is exceeded, the neuron fires and resets. If not, the impulse is latched as the neuron's new membrane potential. ASPEN's primary contributions concern the first phase.

ASPEN is designed to be agnostic to the exact memory and routing structure used in a complete SNN circuit, but some discussion of the assumptions we make is helpful for understanding some of the design decisions made in later sections. SNNs have been shown to be robust to small precision weight

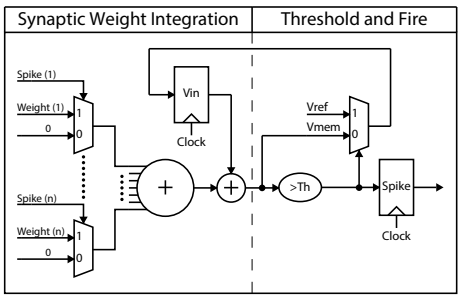

Fig. 1. Standard Integrate-and-Fire Neuron Model

quantization [5]. This allows ASPEN to use q0.8 signed fixed-point numbers for synaptic weights, and q8.8 numbers for the impulse and membrane potentials, with minimal impact on accuracy. This quantization allows this system to store all of its synaptic weights in only 2.4 MB of memory. A unique facet of SNNs memory system is the event driven nature of memory access. These irregular memory access patterns mean that the weights required for a neuron update are received sequentially. ASPEN's weight summation unit is therefore optimized for sequential inputs.

IV. Proposed Design

This section outlines the circuit level design of ASPEN's weight summation system, as well as the approximation methods used.

A. Weight Summation Architecture for Exact Neuron

ASPEN proposes an energy-efficient weight summation circuit that sums a variable number of signed, fixed-point weights that arrive sequentially. The architecture combines parallel counter compression with adder tree networks to perform efficient multi-input addition [9]. This structure decomposes into three modular components: 1) counter compression, 2) parallel block-save addition, and 3) standard binary addition. See Figure 2 for a dot-diagram of the proposed addition unit. Inputs and outputs are marked in black, while internl signals are white and gray. The two most important parameters in this construction are the input width, k, and the counter size, c.

The first stage, counter compression, takes in sequential weight inputs and counts the number of ones in each significance column. ASPEN implements these parallel counters using simple chains of resetable flip-flops [8]. Figure 2 shows examples of positive and negative counter chains for $c = 3$. Each of the k bit positions requires a separate counter, so this stage uses $k * c$ flip-flops. In 3, the results of each counter is represented as a diagonal line of dots. In order to represent the sign bit compressor, the most significant counter result is negated, represented by the dark gray dots in Figure 3. This is the stage that benefits the most from the sequential arrival of the inputs, as counters compression is a very low-energy operation.

The second stage constructs an adder tree network to sum together the properly arranged counter results from Stage 1. This component can be implemented a variety of ways, but ASPEN uses a block-save method that decomposes the counters results into parallel blocks that are used to form pieces of the inputs to Stage 3. The exact dimensions of these blocks are carefully chosen to ensure that the resulting pieces form a disjoint partition of the resulting numbers. This approach runs in parallel and has a modular design that is well suited for the approximation methods outlined in a later section. When c is small, the block sum operation is optimally performed using carry-save adder trees.

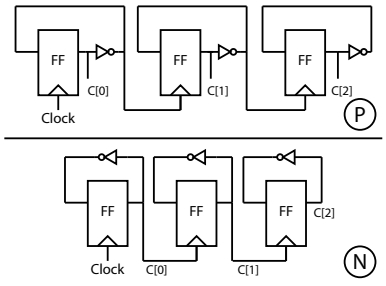

Fig. 2. Counter Chain for c = 3: P is a positive (+1) counter while N is a negative counter (-1).

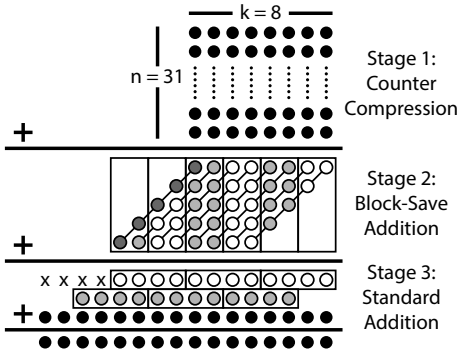

Fig. 3. Weight Summation Unit ($c = 5$): Inputs are marked with black dots. Internal signals are marked with white and gray dots. The 2s complement of the most significant counter result is marked with dark gray dots.

These block sum results, along with an accumulated input, are added together in Stage 3. Importantly, the sign bit from the MSB counter is extended and incorporated with this addition. These extended bits are marked with an $\times$ in Figure 3. This stage first applies a carry-save layer to reduce the three inputs down to two inputs, and then applies a standard binary addition circuit. This final addition can be implemented with any addition method, such as ripple-carry, carry-lookahead, etc. The particular choice of adder design depends on the adder size and desired clock rate. ASPEN uses a simple ripple-carry adder given the small data size. This stage contributes the most to the delay of this system.

In order to sum all n input weights, this system updates and resets at regular intervals. The maximum number of inputs this system can process in a single update is determined by $2^c - 1$, as that is the maximum possible number a counter of size c can represent. At each update, the output of this sub-addition is fed back as an input, and the counters are reset back to zero. This update-accumulate operation continues until all n weights have been summed. This multi-stage process is easy to pipeline, requiring only some internal latches between the various stages.

B. Neuron Approximation Methods

The weight summation design described above performs exact, multi-input addition. One benefit to ASPEN's modular summation design is that it is easy to incorporate approximation modules. ASPEN incorporates two approximation techniques in the Block-Save Addition and the Standard Addition Stages of the multi-input adder described previously.

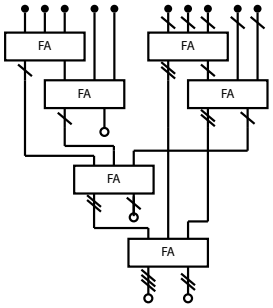

Fig. 4. Exact 5x2 Block Adder Tree. Takes in five 2-bit numbers and outputs a 4-bit sum. Requires 6 full adders. Depth of 4.

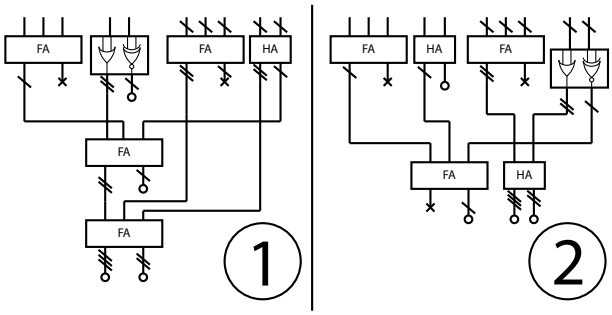

Fig. 5. Approximate 5x2 Block Adders. Design (1) introduces errors at {+1, -2}, and has a Depth of 3. Design (2) introduces errors at {-1, +2, -4}, and has a Depth of 2.

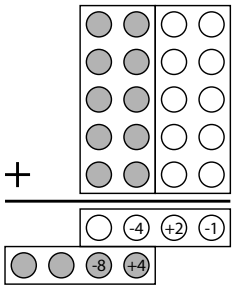

Fig. 6. Block-Save Chain (1,2). The White block is implemented using the (2) network from Figure 5, while the gray block is implemented using the (1) network.

These techniques are analysed to determine their effect on the neuron-level design metrics, including area, power consumption, and delay, and also their effect on overall network accuracy.

The first method applied to ASPEN focuses on approximating the block-save operation in Stage 2 and develops a technique to generate approximate adder trees. Each block-save addition can be implemented optimally using CSA adder networks. An example of such a network for the $c = 5$ case is shown in Figure 4. Each signal has a certain number of dashes along it that indicate its significance. Inputs are marked by black circles, while outputs are marked by white circles. The exact structure of these trees depends on the choice of counter size, but this method generalizes to trees of any size. At a high-level, this method works by fixing carefully selected signals in an adder tree to specific values in order to reduce the depth and complexity of these trees.

First, we examine the signals in the exact adder tree. We reasonably assume random inputs where each bit has an equal chance of being either a zero or one. For a single full or half adder unit given random inputs, the output bits each have an equal probability of begin zero or one. Since an adder tree is entirely comprised of full and half adder units, and both the inputs inputs and outputs of these units have the same distribution, we observe that all signals in an adder tree have a 50% chance of being either zero or one. Therefore, fixing any of these signals to a specific value has a 50% chance of introducing an error with magnitude equal to the significance of that signal. Assuming a signal value of 1 will introduce a positive error, while a value of 0 will introduce a negative error. Components with fixed inputs can often be reduced to simpler forms. For example, a full adder with an input fixed to 0 reduces to a half adder.

Strategically choosing which signals to collapse can reduce the complexity of the adder network significantly. Of particular interest is any signal choice that reduces the depth of the network, as these operations directly reduce the critical path delay. The choice value to fix a signal too is also important. It is often preferable to alternate the sign of introduced errors. This reduces the maximum possible error magnitude by preventing errors from compounding in the same direction. Furthermore, if two errors are introduced at the same significance level, they have a chance of cancelling out and producing a correct final result.

Two levels of tree reduction are presented in Figure 5. In the first approximate network, marked (1), +1 and -2 errors are introduced, which reduces the depth of the network from 4 to 3. In the second network, marked (2), -1, +2, and -4 errors are introduced, which further reduces the depth of the network down to 2.

Since the results of these block-save adders are used to form pieces of the inputs to the Stage 3 addition, we have to think about the effect of these introduced errors upon this final addition. We can see in Figure 3 that the results of adjacent Stage 2 blocks are offset from each other by some shift, s. For the $c = 5$ case, we observe that $s = 2$. Shifting implies that the magnitude of errors introduced in adjacent blocks are offset by a scalar factor equal to $2^s \rightarrow 2^2 = 4$. This allows us to correspond the introduced errors from adjacent approximate adder networks by simply multiplying the magnitude of the errors by the shifting factor.

If two adjacent blocks use the two networks shown in Figure 5 are placed with configuration (1,2), where network (2) is to the right of network (1), then we see that the errors introduced by network (2) overlap with those introduced by network (1). Particularly, the -4 error from network (2) corresponds to the scaled +1 error from network (1). This effect is demonstrated in Figure 6. This offers a general guidelines for choosing error magnitudes and signs in a large block-save network.

The second approximation method applied to ASPEN is

simply to replace the adder in Stage 3 with an approximate adder. Approximate binary addition circuits are quite common [11], [13]. We chose to use the EvoApprox8b library [10]. They provide an open-source library of basic approximate arithmetic units. We specifically chose design 2JY, a 16-bit signed adder design. This was chosen based on experimental results that showed it produced the best results when implemented on the gscl library.

V. Evaluation

We evaluated ASPEN in both hardware and software. The software simulation was written in C++ and emulates the ASPEN neuron. The simulation emulates the the ASPEN neuron and runs an SNN built with it on the MNIST dataset for 10,000 examples. The hardware for ASPEN was developed at the register-transfer level using Verilog HDL and synthesized using the Cadence Genus synthesis tool on the gscl-45 nm library.

ASPEN focuses its design on the neuron level, and intends to be agnostic to the routing and memory architecture used by a complete SNN design. Therefore, these experiments focus on analyzing the impact of various designs on neuron computation. First, we examine the design metrics for an exact ASPEN neuron. Next, we look at how these baseline metrics change when various approximations are applied. Finally, we show some estimates for the per inference computational cost of using the ASPEN neuron as opposed to other neuron types.

A. Exact ASPEN Neuron

We first evaluate the design of a single exact ASPEN neuron. For this, and all other experiments, we set c = 5. We also compare against a carry-save accumulator (CSA-ACC) with the same parameters. The CSA-ACC has two stages, the accumulation stage and the final addition stage. In Table 1, we show the design metrics for the two stages of the CSA-ACC, as well as Stage 1 of the weight summation unit, Stages 2 and 3 of the weight summation unit, and the complete ASPEN neuron, including both the weight summation unit, and the thresholding and fire circuits.

TABLE I
Exact Neuron Circuit Metrics

Circuit	*Area [μm]*	*Power [mW]*	*Delay [ps]*	*Energy [pJ]*
CSA-ACC: Accumulate	426.1	46.38	223	0.1034
CSA-ACC: Addition	348	38.9	1349	0.525
S1: Counters	572.5	11.47	137	0.0157
S2/3: Block Addition	569.7	59.86	1307	0.7824
Full Neuron	1811.5	102.8	1583	1.627

These results show that the counter compressors in Stage 1 consume an order of magnitude less energy than the other stages, which is beneficial as that is the component most frequently active. They also show that the most significant contributors to power consumption and delay are Stages 2 and 3 in the weight summation unit. These units will be the focus of our approximation efforts.

B. Approximate ASPEN neuron

To analyze the impact of the approximate designs described previously, we focus in on Stage 2 and 3 of the summation unit, as these are the components altered under ASPEN's approximation methods. Examining the diagram in Figure 3, we see that the least significant block is trivially computed, so an approximation is not necessary here. The second, b_1, and third, b_2, least significant blocks are non-trivial, however. Approximating Blocks at higher significance levels cause major drops in accuracy, likely due to impacts to the sign of the result, and so will always be left exact. The specific configuration for these two blocks are marked in Table 2 by the pair (b_2,b_1), with the value in each position corresponding to the type of adder tree used to implement that block. A value of 0 corresponds to an exact adder tree (Fig, 3), while a value of 1 or 2 correspond to the approximate trees shown in Figure 5. Experiments are also marked if they are implemented using an approximate Stage 3.

Accuracy numbers are based on running the MNIST dataset through a simulation of a full SNN network built using neurons with the specified approximation method. For comparison, a purely software SNN with the same structure and floating point weights achieves 98.46% accuracy on the same dataset. Energy reduction is compared against the exact case.

The first item in Table 2 corresponds to an exact neuron, and therefore shows the effect of weight quantization on accuracy. This effect is very small, as MNIST accuracy drops only 0.13% from the software baseline. From the exact circuit, we see progressively lower energy consumption with slightly worsening accuracy. The (1,2) w/S3 circuit has the largest reduction in energy at -16.34% and comes with only a 1.84% reduction in accuracy compared to the exact circuit accuracy. Notably, the effect of the Stage 3 approximation is less significant than expected, especially compared against the Stage 2 approximations. One interesting effect of the Stage 3 approximation is that it apparently improves accuracy slightly over the exact Stage 3 adder in some cases.

C. Network Analysis

The above experiments looked at ASPEN at the neuron level. We now want to examine the effect of using this neuron on SNN inference energy consumption. We estimate the total inference energy by analyzing a network's spiking activity and applying the energy contributions from each component of a neuron. As a neuron receives spikes, different components are

TABLE II
Approximate Integration Circuit Metrics

Block Config.	*Area [μm]*	*Power [mW]*	*Delay [ps]*	*Energy [pJ]*	*Energy Red. [%]*	*Acc [%]*
Exact	597	62.27	1405	0.875	0	98.33
(0,1)	580.1	59.83	1314	0.786	-10.17	98.31
(1,2)	564.1	57.09	1314	0.75	-14.29	96.47
(0,0) w/S3	578.6	59.95	1435	0.86	-1.71	98.42
(0,1) w/S3	567.8	57.99	1362	0.79	-9.71	97.9
(1,2) w/S3	546.7	55.17	1326	0.732	-16.34	96.49

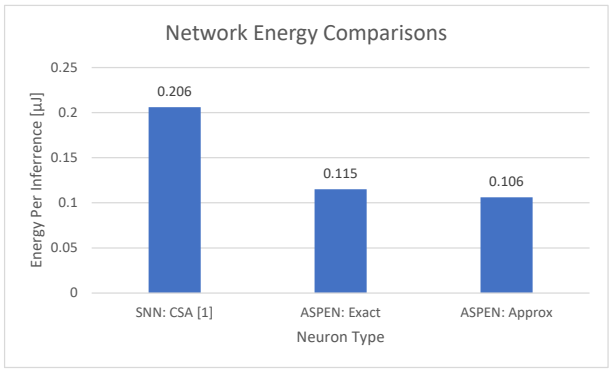

Fig. 7. Total energy per inference on the MNIST dataset. Tested using 1) carry-save accumulator on an SNN, 2) exact ASPEN neuron on an SNN, and 3) approximate ASPEN neuron on an SNN

activated at varying rates. The first stage of the summation unit, counters compression, is activated every time an input is received. Stages 2 and 3 of the summation unit are activated once per update, the frequency of which depends on the counter size. For $c = 5$, the number of inputs per update is $2^5 - 1 = 31$. The neuron thresholding and spike firing occurs once per network time step, after all spikes from the previous layer have been added together. We can use the SNN simulation to find the spiking activity of this network, and use this activity pattern to estimate the total computation energy consumption of an SNN built with a specific neuron architecture.

Figure 7 contains the network energy per inferrence estimates. We also provide an alternative neuron baseline implemented using a standard carry-save accumulator for weight summation. We test the energy consumption on both an exact and an approximate version of ASPEN, specifically (1,2) w/S3.

ASPEN demonstrates superior performance over the baseline CSA weight accumulation method, showing -43.9% lower energy consumption for an exact ASPEN neuron. The approximate neuron shows greater energy reduction, down to -48.8%. Many components, such as the counter compressors, of the ASPEN neuron are shared between the exact and approximate versions. These components form a significant portion of the SNN's energy consumption, but are not effected by our approximate methods. This explains why the difference in energy consumption between the exact and approximate neuron differs at the neuron and network level.

VI. Conclusion

Spiking neural networks are a promising alternative to standard network architecture. This paper presented ASPEN, a neuron design method to develop efficient approximate spiking neurons for SNNs. ASPEN is designed to be modular and parameterizable, so that a variety of different approximation strategies can be easily integrated and tested. We presented a technique to approximate adder trees, and demonstrated the effect of this technique on ASPEN. We present a range of approximate ASPEN neurons, and measured their relative effectiveness. We demonstrate $\times$1.78-1.95 better energy consumption using ASPEN compared to alternative networks and neuron types on a standard benchmark dataset.

References

[1] P. Diehl, D. Neil, et al., "Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing", *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8, 2015.

[2] M. Davies, N. Srinivasa, et al. "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning", *IEEE Micro*, vol. 38, no. 1, pp. 82-99, 2018.

[3] S. Yin, et al. "Algorithm and Hardware Design of Discrete-Time Spiking Neural Networks Based on Back Propagation with Binary Activations", *IEEE Biomedical Circuits and Systems Conference*, 2017.

[4] S. Sen, S. Venkataramani, A. Raghunathon, "Approximate Computing for Spiking Neural Networks", *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 193-198, 2017.

[5] E. Stromatias, D. Neil, M. Pfeiffer, et al, "Robustness of Spiking Deep Belief Networks to noise and reduced bit precision of neuro-inspired hardware platforms", *Frontiers in Neuroscience*, vol. 9, pp 222, 2015.

[6] N. Rathi, P. Panda and K. Roy, "STDP-Based Pruning of Connections and Weight Quantization in Spiking Neural Networks for Energy-Efficient Recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 668-677, 2019.

[7] S. Thabah, M Sonowal, P. Saha, "Experimental Studies on Multi-Operand Adders", *International Journal on Smart Sensing and Intelligent Systems*, vol. 10, pp. 327-340, 2017.

[8] P. Umarani, "A High Performance Asynchronous Counter using Area and Power Efficient GDI T-Flip Flop", *Indian Journal of Science and Technology*, vol. 8, no. 7, pp. 622-628, 2015.

[9] Chi-Hsiang Yeh and B. Parhami, "Efficient pipelined multi-operand adders with high throughput and low latency: designs and applications," Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers, vol. 2, pp. 894-898, 1996.

[10] V. Mrazek, R. Hrbacek, Z. Vasicek and L. Sekanina, "EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 258-261, 2017.

[11] Y. Kim, et al., "Energy Efficient Approximate Arithmetic for Error Resilient Neuromorphic Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2733-2737, 2015.

[12] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1-7, 2016.

[13] Y. Kim, Y. Zhang and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems," *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 130-137, 2013.

[14] P. Simard, et al., "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition*, pp. 958-963, 2003.

[15] *How google translate squeezes deep learning onto a phone*, [online] Available: research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html.

[16] J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau and A. Veidenbaum, "Efficient simulation of large-scale Spiking Neural Networks using CUDA graphics processors," *2009 International Joint Conference on Neural Networks*, pp. 2145-2152, 2009.

[17] S. Venkataramani, A. Ranjan, K. Roy and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 27-32, 2014.s

[18] S. B. Furber, F. Galluppi, S. Temple and L. A. Plana, "The SpiNNaker Project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652-665, 2014.

[19] *Akida Neural Processor SoC*, Brainchip, [online] Available: brainchipinc.com/akida-neuromorphic-system-on-chip.

[20] S. Kannappan and S. Mastani, "A Survey on Multi-Operand Addition", *Acta Technica Corveniensis - Bulletin of Engineering*, vol. 13, pp. 65-68, 2020.