Design, Optimization, and Evaluation of Triplet-Based STDP in a Verilog SNN

Logan Unger, Dakota Barnes, Patrick Truesdale, and Nik Belle

***Abstract*—Spike-timing dependent plasticity (STDP) is a commonly used local learning rule for spiking neural networks (SNNs), but the typical pair-based rule only uses the relative timing of isolated pre-post spike pairs when updating synaptic weights. Pfister and Gerstner (2006) show that using this pair-only assumption does not account for frequency dependent plasticity and asymmetric responses to spike triplets observed in biological synapses. They propose a triplet STDP rule that introduces additional slow spike traces whose accumulation depends on recent firing history. In this work, we implement a two-layer spiking neural network with triplet STDP in Verilog and compare it against a pair-based STDP baseline on a binary classification task. We design input spike patterns that vary in rate and burst structure to evaluate how triplet STDP's additional trace, particularly using an asymmetric depression term to amplify weight decreases at high activity synapses, enables successful classification in situations where pair-based STDP fails. These results demonstrate that the additional trace of triplet learning translates into practical learning advantages. We further investigate the hardware cost of triplet STDP through ASIC synthesis on SkyWater 130 nm and evaluate four targeted optimizations including LUT-based STDP, nearest-neighbor trace simplification, approximate segmented adders, and spike-gated clock enables. We quantify the trade-offs in varying degrees of accuracy for reductions in area, power, and critical-path delay.**

I. Introduction

In the standard STDP learning rule, synaptic potentiation (LTP) or depression (LTD) depends on whether the postsynaptic spike occurs shortly after or before a presynaptic spike. Pfister and Gerstner argue that the pair-only view is not sufficient to explain experimental data from the hippocampus [1]. They identify two major shortcomings leading to their work as (1) frequency dependence, where repeating the same pre and post timing at different repetition rates produces weight changes that pair-only rules cannot predict and (2) triplet asymmetry, where pre-post-pre and post-pre-post patterns have different effects on depression and potentiation, which pair-based STDP would treat as the same. Their proposed triplet rule addresses both by adding slow trace variables that accumulate in proportion to recent firing activity, offering rate-dependent and history-dependent modulation of weight updates that pair-based rules are not able to capture.

The triplet STDP model that Pfister and Gerstner propose addresses the limitations using spike traces, a small set of local state variables that decay over time and are incremented on spike events. The model tracks presynaptic traces (r_1, r_2) and postsynaptic traces (o_1, o_2) where the subscript 1 denotes a fast-decaying trace, while the subscript 2 denotes a slow-decaying trace. The pair-based component of the update stays the same with potentiation depending on whether a presynaptic spike recently preceded a postsynaptic spike and depression depending on the reverse ordering. The third term incorporates slow traces to scale potentiation and depression additionally when r_2 or o_2 are not zero, highlighting that another spike happened recently. Since slow traces accumulate in proportion to firing rate, this formulation provides frequency-dependent plasticity, which is absent from pair-based rules.

In prior coursework, we implemented a pair-based STDP spiking neural network in Verilog that classifies 5x5 binary images of the numbers "1" and "0" using two output neurons with lateral inhibition. In this project, we extend the design to implement Pfister and Gerstner's triplet STDP rule and evaluate whether triplet learning can succeed in scenarios where the pair-based model fails. We construct input spike patterns that vary in rate and burst structure to create conditions where STDP's pair-based updates are insufficient for learning. However, with the triplet rule's third term, there is enough additional information for successful classification.

We compare classification performance between pair-based and triplet implementation under identical network conditions apart from the additional triplet term. Beyond functional evaluation, we quantify the hardware cost of adding triplet STDP through static ASIC synthesis and explore four optimization strategies that reduce area, power, and critical-path delay while preserving varying levels of classification accuracy.

II. Methodology

Our starting point was the pair-based STDP spiking neural network from homework 4. In initial experiments, we observed that the 2-bit weight and 2-cycle shift-register design was very sensitive to the choice of input spike train. This fragility motivated our investigation, as we wanted to understand the conditions under which pair-based STDP breaks down and whether triplet learning could offer more successful learning in the same conditions. To allow ourselves more freedom when designing the spike patterns, and to encode low frequency repetition easier, we extended the spike trains from 20 bits to 40 bits per pixel. This window allowed us to craft patterns with different burst structures and input spike rates while having enough clock cycles for meaningful trace accumulation.

Thank you to Peng Li our professor with the Department of Electrical and Computer Engineering, University of California, Santa Barbara

A. Triplet STDP Implementation

To implement triplet-based STDP, we started with the pair-based STDP SNN from homework 4 and brought it through several modifications to implement the new learning rule.

The most fundamental change made to the design was replacing the binary shift-register spike history that tracked two previous cycles with continuously decaying trace variables, like the ones Pfister and Gerstner presented. In the pair-based model, spike history only tracked whether each neuron fired one or two clock cycles ago. This representation was strictly binary, denoting if a spike happened or not, and only retains two cycles of history. The pair-based STDP update applies fixed potentiation/depression of 1 or 2, considering only isolated pre-post or post-pre pairs.

For triplet STDP, we replaced these shift registers with four sets of 4-bit trace variables. As described in the paper, we use two presynaptic traces per input synapse ($r_1[i]$ and $r_2[i]$, for $i = 0, \ldots, 24$) and two postsynaptic traces per output neuron (o_1 and o_2). The fast traces (r_1, o_1) decay exponentially by shifting the trace right by one bit per cycle, halving the value. The slow traces (r_2, o_2) decay linearly by 2 per cycle. On each spike event, the corresponding traces are incremented by a configurable amount, saturating at a maximum of 15. This trace-based representation encodes recency and frequency of spikes in a varying amplitude instead of a binary flag, which is required for the triplet rule to modulate weight updates beyond what pair-based timing captures.

The weight update rule follows the triplet STDP formulation described in the paper. On a postsynaptic spike, potentiation is computed as:

$$\Delta w^+ = r_1[i] \cdot A_2^+ + \lfloor r_1[i] \cdot o_2/16 \rfloor \cdot A_3^+ \tag{1}$$

The first term is the standard pair-based LTP gated by the fast presynaptic trace and the second is the triplet term where the slow postsynaptic trace o_2 scales potentiation when a recent prior postsynaptic spike has occurred (forming a post-pre-post triplet). On a presynaptic spike, depression is computed as:

$$\Delta w^- = o_1 \cdot A_2^- + \lfloor o_1 \cdot r_2[i]/16 \rfloor \cdot A_3^- \tag{2}$$

The fast postsynaptic trace gates typical LTD and the slow presynaptic trace provides triplet modulation for pre-post-pre spike orderings. We then normalize the product of the two 4-bit slow traces into the same scale as the pair terms by dividing by 16 to ensure it doesn't dominate the equation. The raw potentiation and depression values are computed in 12 bits and are right shifted by a configurable scale factor (we divide by 4) before they are applied as a signed delta to the weight with clamping to the valid range. This controls how aggressive the weights are updated so that they do not saturate.

We kept the pair-based amplitude parameters symmetric ($A_2^+ = A_2^- = 1$) to give a controlled comparison of potentiation and depression to the pair-based design from homework 4. We can then attribute any difference in learning outcomes to the triplet terms. The triplet amplitude parameters are asymmetric in our implementation making depression stronger than potentiation ($A_3^- > A_3^+$). This is motivated by Pfister and Gerstner's description about $A_2^\pm$ and $A_3^\pm$ being independent parameters that they fit to experimental data. Our decision of $A_3^- = 4$, $A_3^+ = 1$ is motivated by observations from our own experiments. We noticed that when triplet terms are configured symmetrically, slow traces create a positive feedback loop making a neuron that fires more frequently build a larger slow postsynaptic trace, which amplifies potentiation for its synapses and causes it to fire even more. This causes the dominant neuron to win for both digits in classification. We counteract this by making triplet depression stronger than triplet potentiation. The result that we hope for is a learning rule where the triplet terms provide the frequency-sensitive modulation that the paper presents. Additionally, we expanded the synaptic weight resolution from 2-bit to 4-bit. When we attempted triplet learning with four weight levels, the graded updates produced by the triplet rule were quantized to the same coarse steps as the pair-based rule. With 16 weight levels, the learning rule has enough range to capture fine-grain differences between pair-based and triplet-based. To accommodate the larger weights, we widened the weighted sum accumulator to 9 bits and all neuron voltage parameters were scaled by 4 times to maintain the same dynamics as in homework 4.

Finally, we incorporated a constant leak term ($V_{LEAK} = 4$) that is subtracted from the membrane potential each cycle, with clamping to ensure the potential doesn't drop below rest. This is meant to model biological neuron behavior more similarly than the setup in homework 4. The lateral inhibition mechanics from the pair-based design in homework 4 are preserved as well.

B. Hardware Optimization Strategies

Having established that triplet STDP provides a clear functional advantage over pair-based STDP, we next investigated the hardware cost of this improvement and whether targeted optimizations could reduce the overhead. We implemented four independent optimization strategies on top of the triplet design, each controlled by a compile-time Verilog parameter so they can be enabled individually or in combination. All optimizations target specific hardware bottlenecks of an unoptimized triplet design: the STDP multipliers, the weighted-sum adder tree, the trace storage and decay logic, and unnecessary switching activity during idle cycles.

1) Lut-Based STDP: The triplet STDP update requires four multiplications per synapse per cycle, replicated across 50 parallel datapaths (25 synapses $\times$ 2 neurons). We replace all runtime multipliers with pre-computed ROM lookup tables for potentiation indexed by $\{r_1[i], o_2\}$ and for depression indexed by $\{o_1, r_2[i]\}$. This reduces the STDP update to a single table read per synapse, eliminating all multiplications, shifts, and saturation clamping from the critical path. The LUT contents are generated offline by a Python script that evaluates the exact STDP equations for every trace combination, ensuring bit-identical behavior to the arithmetic path.

2) Nearest-Neighbor Trace Simplification: In the default all-to-all mode (MODE=0), traces accumulate over the full spike history via saturating addition, so every spike contributes

to all subsequent weight updates. The nearest-neighbor simplification (MODE=1) instead resets each trace to a fixed value on every spike, so only the most recent spike influences the update. In hardware, this replaces the saturating adder, magnitude comparator, and mux with a simple constant load for all four trace paths (r_1, r_2, o_1, o_2). Pfister and Gerstner show that the nearest-neighbor variant still captures frequency-dependent plasticity, though with reduced sensitivity to spike history [1].

3) Approximate Segmented Adder: The weighted sum accumulates 25 synaptic weights into a 9-bit value for membrane potential integration. The carry chain through this adder tree is the critical path for clock frequency. Following the segmented carry-kill approach from Lecture 10, we split each 4-bit weight at bit 2 into a low half $w[1{:}0]$ and a high half $w[3{:}2]$, and accumulate each half with an independent adder tree. The carry out of the low segment is killed, breaking the single 9-bit carry chain into two shorter 7-bit chains.

4) Spike-Gated Clock Enable: In the unoptimized design, every synapse evaluates its full STDP logic every cycle, even when no spikes occur. The spike-gating optimization adds a per-synapse enable: synapse i only computes its STDP update when its presynaptic input fires or an output neuron fires. When idle, the weight register holds and the combinational arithmetic is suppressed. Trace decay still runs unconditionally every cycle for correctness.

C. Hardware Optimization Validation

Each optimization is validated along two axes: functional correctness and physical design cost. Functional regression ensures that optimized variants still classify correctly on a curated set of spike patterns, while static ASIC synthesis quantifies the area, power, and timing impact of each optimization. Both analyses cover eight architecture variants spanning three baseline designs (Original Pair, Pair, Triplet) and five optimized configurations derived from the triplet design.

1) Functional Regression Methodology: We define a suite of 17 spike-train patterns drawn from three cases where triplet STDP outperforms pair-based STDP: weight-resolution sensitivity (6 patterns), low-frequency rate dependence (2 patterns), and burst asymmetry (9 patterns). Each pattern specifies 40-bit white and black pixel spike trains for the 5×5 input grid. All eight architecture variants are simulated against every pattern using Icarus Verilog, and the testbench reports per-neuron firing counts after training and testing phases.

Classification is evaluated at three levels. *Level 1 (Accuracy)*: a pattern passes if the two test phases produce different winning neurons with no ties. *Level 2 (Margin)*: the absolute spike-count difference between the winner and loser measures classification confidence. *Level 3 (Weight Fidelity)*: post-training weight vectors are compared against the unoptimized Triplet baseline using mean absolute error and cosine similarity to quantify how much the optimization perturbs learned representations.

2) PPA Measurement Methodology: All eight variants are synthesized to gate-level netlists targeting the SkyWater 130 nm high-density standard cell library (sky130_fd_sc_hd) using LibreLane (OpenLane 2) [2] with a target clock period of 15 ns. Post-synthesis metrics include cell area (μm^2), internal/switching/leakage power, and setup timing slack.

D. Spike Train Experiments

The implementation for triplet STDP learning was first tested on the same 40-bit spike train that gave successful results in the pair STDP-based neural network to calibrate the design and ensure correct functionality. From this point, we attempted to identify weaknesses in the pair STDP algorithm based on Pfister and Gerstner's work on the effects of triplet spike pattern [1]. It was shown that pair-based STDP models were unable to capture the effects of both low frequency spike events and asymmetric data patterns.

To show a successful implementation of a triplet STDP algorithm, multiple cases had to be shown. The first case was set up to show how triplet STDP did not break a setup where pair STDP was successful. The remaining cases were used to show examples of realistic spike trains where triplet STDP successfully captured the information as expected, but pair STDP failed. This would show that for the added hardware for the triplet learning network, it was able to more closely represent the learning algorithm displayed in experimental brain data.

This STDP network was trained as a 25 neuron input layer corresponding to a 5x5 black and white pixel grid containing patterns for a 0 and a 1. 2 Output neurons were used to classify images as either 0 or 1. As stated previously, pixel input spike trains were expanded to 40 bits per pixel to allow more flexibility in the input data. To capture low frequency events, white pixel spike train inputs were kept extremely sparse. A successful learning algorithm would still be able to capture these inputs and decrement the weight values accordingly, where a failed model would not, and leave neurons associated with white pixels in training unmarked. Testing runs were executed after a model was trained using an image of a 0 or a 1 with a single flipped pixel.

Experiment 0.1 Input Spike Trains:

```
White:0100000100000000010
Black:0101010010101000101
```

Experiment 0.2 Input Spike Trains:

```
White:0000000000001000000000000010000000001000
Black:0000100001000010000100001000010000100010
```

This was the starting point for the input spike trains. Experiment 0.1 was carried out in Homework 4, so to allow more flexibility in spike train deviations, this input was extended to 40 bits in Experiment 0.2. This experiment was expected to fail, as the increase in spike generation per pixel would break the model if no other parameters were tuned accordingly.

Experiment 1.1/1.2 Input Spike Trains:

```
White:0000000000001000000000000010000000001000
Black:0000100001000010000100001000010000100010
```

Experiment 1.1 was used to calibrate the pair-STDP model to the new spike train length by extending the weight values to 4 bits and observing the effects. Experiment 1.2 was used to observe how the triplet implementation compares to a pair implementation with the same unmodified spike train.

Experiment 2.1/2.2 Input Spike Trains:

```
White:00100000000010000000001000000000010000000
Black:00100100001001000010010000100100001000100
```

The next set of tests was designed to focus on the asymmetric response to low frequency triplet neuron firings. The black pixel spike train follows a spiking pattern with a low frequency occurrence of alternating firing patterns. This allows the model to use different patterns than the previous triplet test with the expectation that pair models would not be able to capture the same weight change.

Experiment 3.1/3.2 Input Spike Trains:

```
White:00001000000010000000100000001000000000000
Black:00011100000001110000000111000000001110000
```

The last experiment was designed to investigate the response of a set of bursting neurons. This type of input spike train has multiple firings in quick succession, followed by a longer period at rest, mimicking the bursting neuron found in research [3]. This was done to put higher emphasis on observing the robustness of the learning strategy, as an implementation that can handle a wider range of firing patterns is easier to use on large scale models with different neuron classifications.

III. Results

A. Functional Results - Pair vs. Triplet STDP

The results of the experiments shown in Methodology are as follows (see end of document for weight maps and output spikes). Figures 4 and 5 show the results of the baseline experiments 0.1/0.2 performed on a pair STDP network for both 20 bit input / 2 bit weight and 40 bit input / 2 bit weight. Experiment 0.1 showed a successful baseline implementation, while Experiment 0.2 showed how increasing the size of input spike trains caused an overall increase in excitation, causing unwanted potentiation in neurons.

Next, Experiment 1 was run on both pair-STDP and triplet-STDP shown in Figure 6 and Figure 7. The resulting weight maps showed that pair-STDP was able to be fully calibrated to 40 bit input spike trains by increasing the weight bit size to 4. Output spike counts showed proper classification for testing image sets. Experiment 1.2 gave identical results, showing that for a baseline implementation, triplet learning does not ruin previously accepted results. This was important to support the hypothesis that triplet learning could be used to improve the robustness of what kinds of input spike trains would generate a successfully trained model.

The next set of tests was shown under Experiment 2 (Figures 8 and 9). The sparsity of the input spikes increased and the output spike tables in Experiment 2.1 showed no ability to classify between a 0 or a 1. Interestingly, the triplet weight map was nearly identical to the pair weight map, with a slight increase in weight depression in the 0 input neuron. This is concurrent with the expected results of low-frequency learning, as long gaps in input neurons mean the model will have trouble recognizing pre-post relationships, but the triplet model is able to more heavily modify weights based on these long term relations. Even though the results were only slightly different in the weight maps, this change made a huge difference in output spikes due to the sparsity of input spikes, and the triplet implementation was able to strongly classify the input images based on their output spikes.

The last set of tests run under Experiment 3 (Figures 10 and 11) showed the results of a burst-heavy spike train. Under Experiment 3.1, the pair-STDP model heavily favored potentiation, as the neuron representing 0 (the first trained image) was filled with extremely high weight values and a larger amount of output spikes, constantly inhibiting the other neuron and preventing it from being trained. Experiment 3.2 showed that the triplet setup was able to properly potentiate relative to the bursting input spikes and output spikes strongly classified each testing image.

Finally, note that we have more results in our codebase, under the `results/` directory. These include 2 experiments for case 2 (frequency changing spike patterns) and 9 experiments for case 3 (high frequency bursts). For each of the experiments, we have the weight maps of each neuron at each step of the training progress. These are labeled by case and saved as .png files. Next, we have a called experiments.log that shows how we ran every experiment, and this file also includes the number of firings for each neuron for the training data and the testing data. Finally, for the 4 experiments that we included in the report above, there are .png files that end in "_wave.png" and each of these has the waveforms for training and testing on each digit.

TABLE I
FUNCTIONAL REGRESSION RESULTS ACROSS ARCHITECTURES

Design	Acc	Marg	Cos Sim	Δ Acc	Δ Marg
Original Pair	0/17	2.35	0.796	−17	−2.41
Pair	6/17	5.29	0.940	−11	+0.53
Triplet (baseline)	17/17	4.76	1.000	—	—
Triplet Optimized	17/17	4.76	1.000	+0	+0.00
Spike-Gated	17/17	4.76	1.000	+0	+0.00
Nearest-Neighbor	11/17	5.88	0.960	−6	+1.12
Seg. Adder	11/17	4.59	0.907	−6	−0.18
LUT-STDP	17/17	4.76	1.000	+0	+0.00

B. Optimization Results

Using the regression suite and synthesis flow described in Methodology, we present functional and physical results for all eight architecture variants. These span the three baselines of Original Pair (2-bit weights, IF neurons), Pair (4-bit weights, LIF neurons, pair-only STDP), and Triplet (4-bit weights, LIF neurons, triplet STDP). An additional five optimized configurations derived from the Triplet design include Triplet Optimized (minor RTL syntax optimizations over the Triplet baseline, serving as a control), Spike-Gated, Nearest-Neighbor,

Segmented Adder, and LUT-STDP. Table I summarizes classification accuracy and weight fidelity across all variants, while Figures 1, 2, and 3 show area, power, and timing from static synthesis.

1) Functional Regression Results: Table I shows the regression results across all eight architecture variants. The Triplet Optimized (all optimizations disabled as a control), Spike-Gated, and LUT-STDP pass all 17/17 patterns with identical weight vectors (cosine similarity 1.0000) to the unoptimized Triplet baseline. This is expected as the spike gate suppresses STDP computation only when both pre- and post-synaptic spikes are absent (making all update terms zero), and the LUT tables store values computed from the exact same arithmetic expressions. However, we note that this equivalence is verified empirically over our test suite rather than formally proven across all possible inputs.

The Nearest-Neighbor optimization (MODE=1) passes 11/17 patterns with a cosine similarity of 0.9598. The six failures occur on Case 1 weight-resolution patterns and Case 3 burst patterns where the all-to-all trace accumulation provides critical history that the nearest-neighbor reset discards. Interestingly, the patterns it does pass show a higher mean spike-count margin (5.88) than the baseline (4.76), suggesting that nearest-neighbor traces produce sharper, more decisive classifications when they work.

The Segmented Adder also passes 11/17 with a lower cosine similarity of 0.9074, reflecting the bounded approximation error from carry-kill truncation. Its failures cluster in Case 1 patterns where small weight differences are critical. The weighted-sum error is enough to tip marginal threshold crossings, and the mean margin drops slightly to 4.59.

2) PPA Results - Static Synthesis: As shown in Figure 1, triplet STDP roughly triples the area of the pair-based design due to the added trace registers and multipliers. Among optimized variants, Nearest-Neighbor provides the largest area reduction by simplifying trace logic, while the other optimizations have minimal area impact. Figure 2 shows that Segmented Adder and Nearest-Neighbor achieve the largest power savings, followed by LUT-STDP which eliminates multiplier switching. Spike-Gated shows only a modest reduction in static synthesis, though the true benefit is data-dependent and would require dynamic power analysis with realistic activity factors to fully capture. For timing (Figure 3), all designs meet the target clock period comfortably. The Segmented Adder slightly improves slack by shortening the carry chain, while Nearest-Neighbor has the tightest slack due to added mux logic in the trace-reset path.

3) Optimization Tradeoff Summary: The four optimizations fall into two categories based on their accuracy-efficiency tradeoff. Spike-Gated and LUT-STDP both pass all 17 patterns with identical weights to the baseline, while reducing power by 8.6% and 14.3% respectively. Spike-Gated adds virtually no area and its savings are conservative in static synthesis. Measuring the full benefit would require dynamic power analysis with realistic spike activity factors. LUT-STDP trades a small area reduction (-3.6%) from removing multiplier trees for ROM storage; however, the timing improvement was

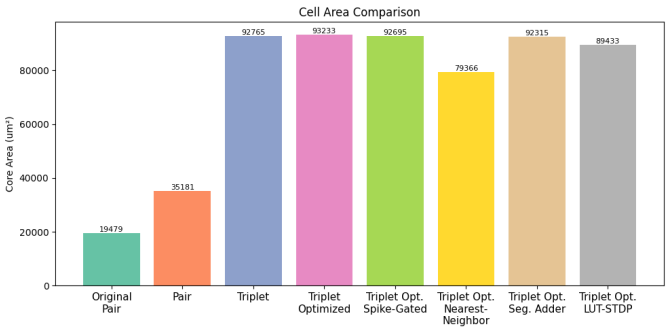

Fig. 1. Cell area comparison across all eight variants.

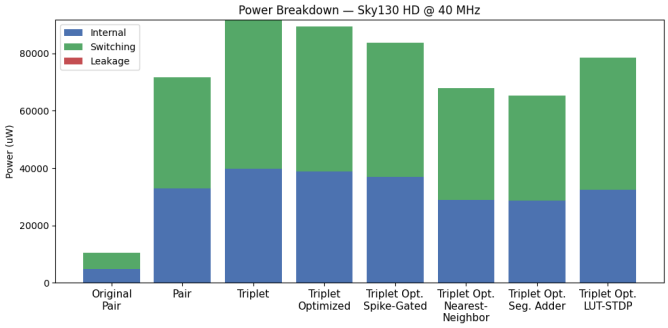

Fig. 2. Power breakdown at 40 MHz on SkyWater 130 nm.

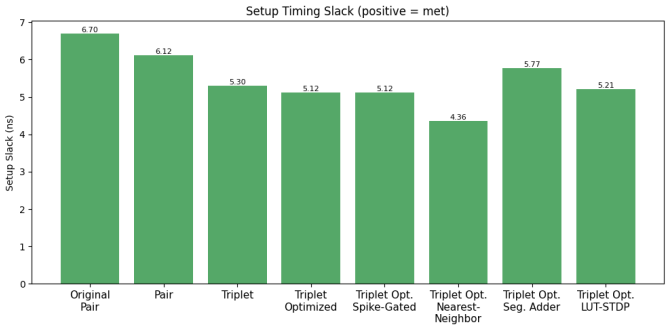

Fig. 3. Setup timing slack with a 15 ns target clock period.

smaller than expected, as the ROM read path replaces but does not substantially shorten the critical path.

Nearest-Neighbor and Segmented Adder each drop to 11/17 accuracy in exchange for the largest power savings (26% and 29%). Nearest-Neighbor's failures stem from discarding spike history that the all-to-all trace accumulation preserves. Patterns requiring fine weight resolution or sustained burst context are most affected, though the patterns it does pass show higher classification margins than baseline. The Segmented Adder's failures come from carry-kill error tipping marginal threshold crossings in patterns where small weight differences matter. Notably, Nearest-Neighbor also provides the largest area savings (-14.4%) but introduces the tightest timing slack, while the Segmented Adder has negligible area impact but actually improves timing by shortening the carry chain.

IV. Discussion

A. Further Work

We were only able to answer a fraction of the questions that we had when we started this project. If we had more time to develop our architecture, we could have explored many avenues. With more time, we could scale up the image size to get closer to the 28x28 images in the MNIST handwriting dataset. We made a Python script that would allow us to draw numbers using the cursor and use those for training and testing, which did not make it into our final project. This would have greatly increased the amount of data we could have used for training and testing. However, it would have presented additional complications like finding appropriate noisy filters to add to the training samples to ensure greater generalization of the model upon testing, and we didn't want to turn this project into a traditional machine learning challenge. We could use grayscale images or color images with RGB channels. We could make the lengths of the spike trains even longer, and we could even make a script that could generate spike trains based on the integer color value for each pixel. Additionally for optimizations, a natural next step would be validating against larger and randomly generated spike-train datasets, combining multiple optimizations together, and exploring the design space across different synthesis configurations and hardware platforms.

Ultimately, we wanted to deeply investigate the mechanisms of triplet STDP and uncover useful applications for it based on more variable spike train data. We quickly found that the complexity was not limited by small images and training classes.

B. Conclusion

Our results highlight that the implementation of triplet-based STDP presented in the paper, provides learning improvement over pair-based STDP in a simple binary classification scenario. Additionally, we found that pair-based STDP works in situations where the weight and trace depths are expanded to 4 bits, and the spike trains for black and white are expanded to 40 bits. We saw that altering the frequency of the spikes for each color led to degraded performance for the pair-based model, while the triplet model was able to handle these patterns without any problems. Even though the weight maps did not look as pretty as the baseline from Homework 4, the spiking of the networks tells us that neuron 1 and neuron 2 specialized to each recognize a distinct digit. When we tested the pair-based model and the triplet model with spike trains that contained high-frequency bursts, we saw where the triplet model showed higher success over the pair-based model. The weight maps for our best-case experiment made it clear that neuron 1 was recognizing digit 0 and neuron 2 was recognizing digit 1. On the hardware side, triplet STDP roughly triples the area of the pair-based design, but targeted optimizations can recover much of this cost. Spike-Gated and LUT-STDP preserve full classification accuracy while reducing power, and the lossy Nearest-Neighbor and Segmented Adder optimizations offer larger area and power savings at the expense of some pattern coverage.

References

[1] J.-P. Pfister and W. Gerstner, "Triplets of spikes in a model of spike timing-dependent plasticity," *Journal of Neuroscience*, vol. 26, no. 38, pp. 9673–9682, 2006. [Online]. Available: https://www.jneurosci.org/content/26/38/9673

[2] M. Shalan and T. Edwards, "Building openlane: A 130nm openroad-based tapeout- proven flow : Invited paper," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–6.

[3] Y. Çakir, "Modeling of synchronization behavior of bursting neurons at nonlinearly coupled dynamical networks," *Dynamical Systems: An International Journal*, 2016. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/27830974/

Pair	Train 0	Train 1	Test 0	Test 1
N1	2	0	2	0
N2	0	2	0	2

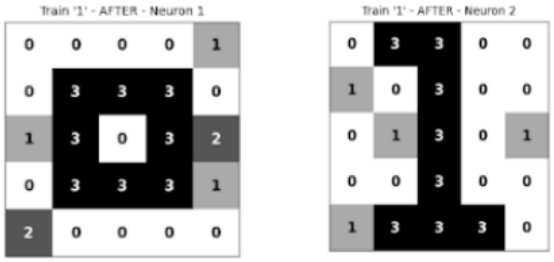

Fig. 4. Experiment 0.1

Pair	Train 0	Train 1	Test 0	Test 1
N1	3	2	3	1
N2	0	2	3	1

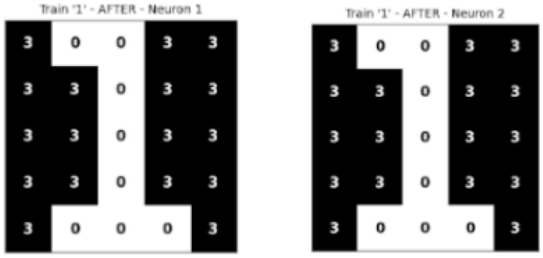

Fig. 5. Experiment 0.2

Pair	Train 0	Train 1	Test 0	Test 1
N1	2	0	2	1
N2	0	2	0	2

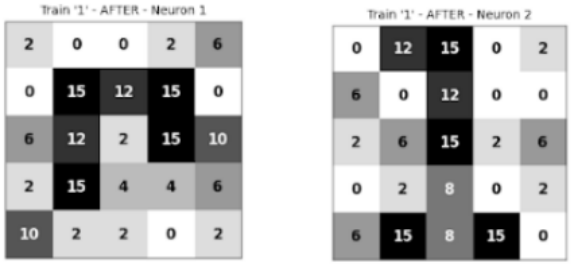

Fig. 6. Experiment 1.1

Triplet	Train 0	Train 1	Test 0	Test 1
N1	2	0	2	1
N2	0	2	0	2

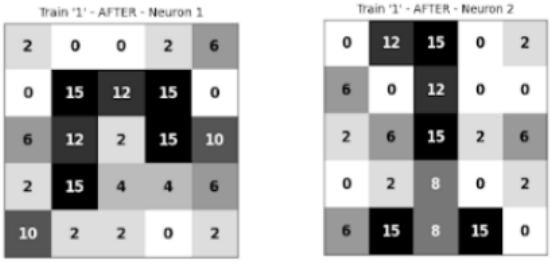

Fig. 7. Experiment 1.2

Pair	Train 0	Train 1	Test 0	Test 1
N1	3	3	4	4
N2	0	1	0	2

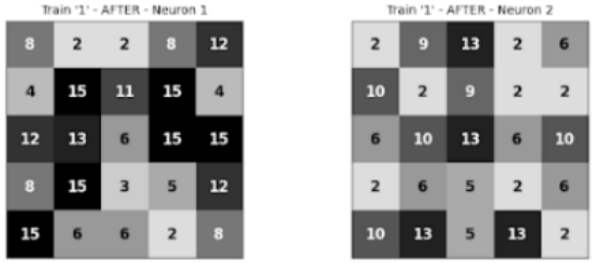

Fig. 8. Experiment 2.1

Triplet	Train 0	Train 1	Test 0	Test 1
N1	3	2	4	1
N2	0	3	0	3

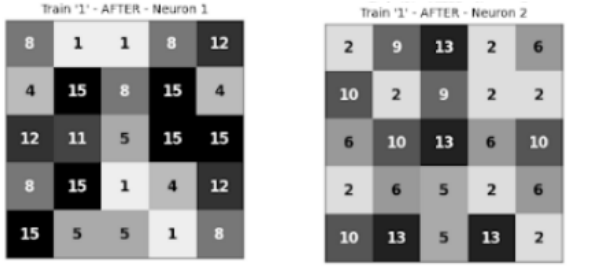

Fig. 9. Experiment 2.2

Pair	Train 0	Train 1	Test 0	Test 1
N1	4	4	5	4
N2	0	0	0	0

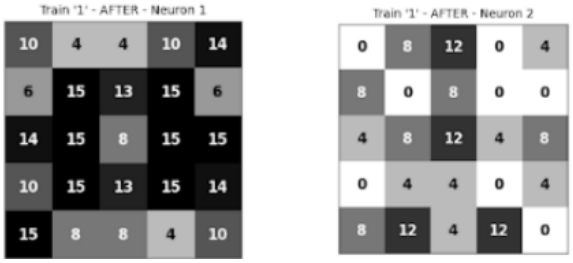

Fig. 10. Experiment 3.1

Triplet	Train 0	Train 1	Test 0	Test 1
N1	4	2	4	0
N2	0	2	0	4

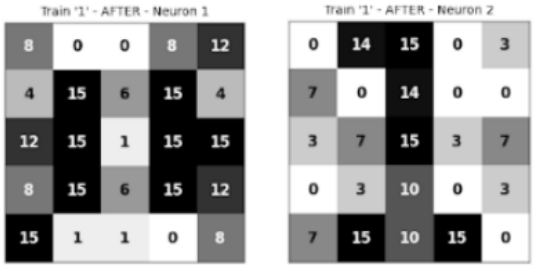

Fig. 11. Experiment 3.2