Direct CMOS Implementation of Neuromorphic Temporal Neural Networks for Sensory Processing

Harideep Nair, John Paul Shen, James E. Smith
Carnegie Mellon University

Abstract—Temporal Neural Networks (TNNs) use time as a resource to represent and process information, mimicking the behavior of the mammalian neocortex. This work focuses on implementing TNNs using off-the-shelf digital CMOS technology. A microarchitecture framework is introduced with a hierarchy of building blocks including: multi-neuron columns, multi-column layers, and multi-layer TNNs. We present the direct CMOS gate-level implementation of the multi-neuron column model as the key building block for TNNs. Post-synthesis results are obtained using Synopsys tools and the 45 nm CMOS standard cell library. The TNN microarchitecture framework is embodied in a set of characteristic equations for assessing the total gate count, die area, compute time, and power consumption for any TNN design. We develop a multi-layer TNN prototype of 32M gates. In 7 nm CMOS process, it consumes only 1.54 mm$^2$ die area and 7.26 mW power and can process 28x28 images at 107M FPS (9.34 ns per image). We evaluate the prototype’s performance and complexity relative to a recent state-of-the-art TNN model.

I. INTRODUCTION

A. Temporal Neural Networks

Temporal Neural Networks (TNNs) [31], [33] are a class of spiking neural networks (SNNs) that encode and process information in temporal form using precise spike timings to represent information [40] unlike other forms of SNNs that use spike rates for information encoding and processing. Furthermore, unlike artificial neural networks (ANNs), TNNs strive for biological plausibility with the goal of achieving brain-like capability and brain-like efficiency.

TNNs are composed of a hierarchy of neurons that perform temporal functions on temporally encoded values. TNN neurons communicate via spikes, encode information using relative spike timings, and employ STDP local learning, as shown in Figure 1. TNNs are trained via Spike Timing Dependent Plasticity (STDP) [10]. With STDP, a synapses weight is updated based only on the relative timings of its incoming spike (from a pre-synaptic neuron) and the outgoing spike, if any, (from its post-synaptic neuron). Hence, STDP training is localized and has the potential for enabling unsupervised, online, and continuous learning, without requiring compute-intensive back propagation commonly employed in ANNs and SNNs. These distinctive attributes of TNNs adhere strongly to biological plausibility and make them truly neuromorphic.

B. Our Approach

TNN research has a long history. Some of the historical contributions are highlighted here. In 1989, Thorpe and Imbert [38], [39] made a persuasive experimentally-supported argument for inter-neuron communication via precisely timed spikes. Hebb [11] observed that repeated temporal coincidence between a synapses input spike and its post-synaptic neurons output spike tends to increase the synapses weight. In 1983, Levy and Steward [16] established the classic STDP update rules. Gerstner et al. [8] proposed a theoretical foundation for STDP as an integral part of a computing paradigm.

This work builds on recent works in [31], [33], [34] which laid the foundation of TNNs as space-time computing networks based on a rigorous space-time algebra. The author in [32], [35] suggested building a silicon neocortex by examining the hierarchical organization of biological neural networks to formulate an analogous hierarchical organization for TNNs. As computer architects, we follow this proposed approach and focus on direct hardware implementation of TNNs.

C. Our Contributions

In this work, we explore the practical feasibility of direct hardware implementation of TNNs using off-the-shelf digital CMOS technology and design tools. In a direct CMOS implementation, the actual hardware clock cycle is used as the basic time unit for temporal processing. We focus on defining a TNN microarchitecture and implementing its hierarchy of building blocks. From the bottom up, these building blocks include: neurons, multi-neuron columns, and multi-column layers, resulting in multi-layer TNNs, as shown in Figure 2.

We present gate-level designs (in Verilog) of neurons and columns, including gate-level implementation of STDP (unsupervised) and R-STDP (supervised) learning rules. We obtain post-synthesis results using Synopsys tools. These designs are evaluated based on the metrics of: gate count, die area, critical path delay/compute time, and power consumption.

Key contributions of this work include:

- A microarchitecture framework for direct CMOS implementation of TNNs, based on building blocks of neurons,
II. BACKGROUND AND PRIOR RESEARCH

A. Deep Artificial Neural Networks

Deep Artificial Neural Networks (ANNs), including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are dominant paradigms for performing human-like sensory processing using machine learning (ML) techniques. These networks effectively support ML applications such as image classification and speech recognition.

Deep ANNs typically employ: 1) tensor processing with high-dimensional matrix multiplications, and 2) supervised global learning with back propagation and stochastic gradient descent. Even though highly effective, neither of these techniques are biologically plausible and require an enormous amount of linear algebraic computation. Since 2012, the computation required for training deep ANNs has been doubling every 3.4 months, or increasing at the rate of 10x/year [26].

TNNs are fundamentally different and are based on a computing paradigm that does not involve high dimensional tensor processing. TNNs perform computation based on spike timing relationships and do not use linear algebraic operations. The learning paradigm for TNNs is localized and predominantly unsupervised, and is capable of online and continuous learning. In TNNs, inference and training can occur simultaneously.

B. Spiking Neural Networks

Spiking Neural Networks (SNNs) use spikes to communicate between neurons and are a broader class of neural networks that includes TNNs as shown in Figure 1. However, many SNNs in the literature employ rate encoding as opposed to TNNs which use temporal encoding. In rate encoding, spike rates are used to represent values whereas in temporal encoding, relative spike times are used.

Many SNNs also use backpropagation for learning, similar to deep ANNs. Multilayer SNNs proposed in [15], [24], [25] employ both rate encoding and backpropagation, whereas SNNs in [3], [6], [29], [36], [37] use rate encoding coupled with STDP. The works in [17], [21] implement feedforward SNNs with temporal encoding but use backpropagation for learning. TNNs use spike times for temporal encoding and STDP for local learning.

C. Neuromorphic Hardware

In recent years, several experimental neuromorphic chips, implemented in standard digital CMOS, have been introduced.

TrueNorth is a digital neuromorphic chip introduced by IBM in 2014 [20] fabricated using 28 nm CMOS technology. It consists of one million neurons distributed across 4096 cores, with each core containing 256 leaky-integrate-and-fire (LIF) neurons, each with 256 synapses. Synaptic weights can be set to either 0 or 1 (1 bit). The chip runs at 1 KHz with 4-bit resolution for time steps (0 to 15). Time delays are stored to either 0 or 1 (1 bit). Spikes are communicated as packets using address event representation (AER). The authors use offline training for their performance demonstration.

Intel unveiled Loihi in 2018 as their flagship neuromorphic chip built in 14 nm process [5]. Each chip consists of 130,000 LIF neurons distributed across 128 cores; each core consists of 1024 neurons, each with 1024 synapses. It can support 1-9 bit synaptic weights. Spikes are represented as packetized messages. Intel has provided Loihi boards for cloud access and experimentation by researchers.
Figure 3: A TNN Neuron Model

Figure 4: A TNN Column Model

Figure 5: TNN Multi-Column Layers: Two Types

III. TNN ORGANIZATION AND OPERATION

We leverage insights from both conventional computer systems as well as biological neocortex systems. Both are hierarchical systems with multiple levels of abstraction. The building blocks in our hierarchical TNN microarchitecture mimic those of biological neural networks (Figure 18). TNN building blocks include neurons, multi-neuron columns, and multi-column layers. This paper focuses on columns as the primary building blocks for multi-layer TNNs (Figure 2).

A. Key TNN Building Blocks

As shown in Figure 3a, each neuron has \( p \) synaptic inputs and one output. Each synaptic input carries a synaptic weight, which is updated locally based on the relative timing of the incoming spike to that synapse and the outgoing spike from the neuron body. The rules for updating synaptic weights constitute the STDP learning method.

As shown in Figure 4a, a column is a stacking of \( q \) parallel neurons. Every neuron in a column shares the same set of \( p \) inputs. There is a \( pxq \) synaptic crossbar containing \( pxq \) synaptic weights, each of which is independently updated by STDP. On the output side of the \( q \) neurons, lateral inhibition is performed by selecting the earliest spiking \( k \) neurons from among the \( q \) neurons as the winners (\( k \)-WTA). Output spiking is disabled for non-winning neurons. Typically \( k=1 \) is used.

This paper presents the CMOS implementation of the neuron (Section IV) and the column (Section VI). In Section STDP rules for updating synaptic weights are given. The baseline STDP method is unsupervised. We also introduce a variation, called reinforcement STDP, which is similar to the reward modulated STDP in [23]. R-STDP is usually deployed in the last layer of a multi-layer TNN. Figure 5 illustrates the two types of TNN multi-column layers: Unsupervised Layer with STDP and Supervised Layer with R-STDP.

B. Temporal Encoding and Processing

A distinctive attribute of TNNs involves the use of temporal encoding. With temporal encoding, information is represented by relative timings of spikes. In a TNN, computation occurs in volleys or waves of spikes. A volley consists of at most one spike per synaptic input (some will have no spike). The value represented by each input spike is based on its spike time relative to the first spike in the volley. The first spike in the volley represents a value of 0 and subsequent spikes are assigned increasing values based on increasing delays relative to the first spike.
Temporal encoding is illustrated in Figure 6. If no spike occurs on an input, it is assigned the symbol “∞”. It has been proposed, with experimental support, that computational volleys are synchronized by inhibitory oscillations at gamma frequencies (50-100 Hz) [9]. These gamma oscillations divide neocortical processing into alternating inhibitory and excitatory phases. The inhibitory part of the cycle essentially performs a reset function and the excitatory part performs computation. This leads to a temporal coding interval of 5-10 msec during which spikes are transmitted in a coordinated volley. Neuron spiking behavior has been shown to be repeatable to within 1 msec [4], [18]. A 5-10 msec window combined with the 1 msec encoding precision implies only 3-4 bits of precision is needed within the encoding window. Consequently, the computing model used in this paper is based on low resolution integers.

In this work, temporal encoding and processing are employed with the actual hardware clock cycle directly serving as the basic time unit. This work uses 3 bits of precision for encoding and synaptic weights. Spikes in a volley are represented using pulses which are a form of unary encoding and volleys are separated using gamma clock cycles. With unary encoding, it takes up to 7 time units to encode a 3-bit value. To allow additional time for a column to process a spike volley, the gamma cycle is extended to 15 time units. This is explained in further detail in Section IV-B.

In summary, the proposed design uses two clocks. The unit time clock is the finest temporal resolution in the computation model and is also the synchronizing clock used in the digital hardware. The gamma clock frames the computing window and is the time required for a column to communicate and process spike volleys and update synaptic weights.

IV. NEURON IMPLEMENTATION

This work focuses on the SRM0 excitatory neuron model based on the widely-used Spike Response Model [12] shown in Figure 3b. This section presents the components of this model and their detailed gate level designs along with corresponding gate count equations (using equivalent gate counts in terms of 4-input AND gates). For gate level designs and analysis, we set the maximum weight value \( w_{\text{max}} = 7 \).
counter has three modes, two controlled by STDP (described in Section V): increment (up to \(w_{\text{max}}\)) and decrement (down to 0). The third readout mode is controlled by the input pulse. Readout mechanism is meticulously integrated into the same FSM used for storing synaptic weight and is described below.

As will become apparent, the synapses are the dominant hardware cost, so the synapse design must focus on minimizing hardware. A clever idea involves using a pulse width equal to \(w_{\text{max}} + 1\). The input pulse directly controls the counter readout. When the leading edge of an input pulse occurs (0 to 1 transition), the weight is read out and decremented until it reaches 0. An output of 1 is emitted each unit clock cycle until the weight in the counter is restored to its original value. Thus, once an input spike arrives, readout takes an additional 7 cycles. (Although we assume \(w_{\text{max}} = 7\) in this paper, this technique can be generalized to any \(w_{\text{max}}\).) During training, STDP (explained in Section V) takes another cycle. These coupled with 7 cycles for encoding give rise to a gamma period of 15 clock cycles.

In summary, a synapse and its weight are implemented with a counter FSM that can 1) increment, saturating at \(w_{\text{max}}\), 2) decrement, saturating at 0, and 3) wrap-around decrementing, emitting an output of 1 prior to wrapping around and a 0 thereafter. Based on our logic design, not including STDP, each synapse requires a total of 61 equivalent gates (counting latches as 2 gates, FF as 5 gates) or \(61p\) gates for \(p\) synapses, most of which arise from the state transition logic.

C. Neuron Body

The neuron body is implemented as a parallel counter that adds the thermometer coded weights coming from the synapses, cycle by cycle, thereby accumulating the membrane potential as a sum of RNL response functions. When (and if) the parallel counter output reaches the threshold \(\theta\), an output spike is emitted during that cycle.

Using the work of Parhami [27], the membrane potential accumulator can be implemented using ripple carry adders as fundamental units by integrating a \((p-1)\)-input parallel combinational counter and a \((\log_2 p + 1)\)-bit adder into one design. Figure 9 shows the logic diagram for a 16-input accumulator, with integrated output spike generation. For a \(p\)-input accumulator, \(p\) inputs are accumulated into a \((\log_2 p)\)-bit output, which is then added to the previous stored \((\log_2 p + 1)\)-bit value from the register with the one remaining input bit acting as carry-in. Note that the hierarchical configuration in Figure 9 allows all adder inputs to be efficiently utilized and is particularly optimal when \(p\) is a power of 2.

Furthermore, the accumulating register is initialized with (signed 2s complement) -\(\theta\) at every gamma cycle, which eliminates the need for any comparator for output spike generation. The \((\log_2 p + 1)^{\text{th}}\) bit of the output can be used to determine if the accumulated body potential has crossed the threshold and trigger a 3-bit counter that generates an 8-unit time pulse (output spike). For a neuron body with \(p\) synaptic inputs, the gate count comes out to be \(5p + 8\log_2 p + 31\).

Conventional digital logic is used for implementing temporal encoding and processing with very efficient hardware. Small counters (FSMs) store and update synaptic weights, rather than much longer shift registers with unary representation of weights. A small adder tree sums multiple single-bit inputs to realize the summing of higher precision values over time. STDP and R-STDP designs are presented next.

V. STDP & R-STDP IMPLEMENTATION

STDP is a distinctive feature of TNNs. STDP learning is unsupervised and local to each synapse. It can perform inference and continuous learning at the same time. In this work, we propose an STDP design that is both effective in learning and implementable using standard CMOS technology.

A. Proposed STDP Update Rules

Our learning method is a customized version of the classic Spike Timing Dependent Plasticity (STDP). STDP is implemented locally at each synapse as shown in Figure 10.

The proposed STDP learning rules are summarized in Table I. Here, \(x(t)\) and \(z(t)\) represent input and output spiketimes respectively. \(\Delta w\) denotes change in weight and \(B(\mu)\) represents a Bernoulli random variable with probability \(\mu\).

STDP update rules are divided into four major cases, corresponding to the four combinations of input and output
spikes (represented by \(x(t)\) and \(z(t)\) respectively) being present \((\neq \infty)\) or absent \((= \infty)\). When both are present, two sub-cases are formed based on the relative timing of the input and output spikes in the classical STDP manner. In effect, a synaptic weight is incremented (strengthened) if there is an input spike and it either contributed (Case 1) or can potentially contribute (Case 3) to the output spike; else it is decremented.

The STDP update function either increments the weight by \(\Delta w\) (up to a maximum of \(w_{\text{max}} = 7\)), decrements the weight by \(\Delta w\) (down to a minimum of 0), or leaves the weight unchanged. The \(\Delta\) values are defined using Bernoulli random variables (BRVs) with parameterized learning probabilities denoted as \(B(\mu)\) with a descriptive subscript. \(F(w)\) is a stabilization function \((=B((w/w_{\text{max}})(1 - w/w_{\text{max}})))\) which makes the weights “sticky” at both ends (0 and 7).

### B. Proposed STDP Implementation

The proposed STDP logic implementation is shown in Figure 11. It generates 2 control signals (increment/decrement) at the output that feed into the synaptic weight counters described in Figure 8. Here, it is assumed that all the required BRVs have been generated from an LFSR network shared by many neurons or columns. Although detailed LFSR network designs are not considered here, introducing LFSRs into the implementation should be straightforward once their distribution across the multi-layer TNN is determined. Note that STDP updates (and the associated resets) are performed at the end of a computational cycle (or onset of next gamma clock); inputs for the new computational cycle begin a unit clock cycle later.

The proposed STDP logic implementation can be partitioned into three components.

1) **Case Generation Logic:** The per-synapse case generation logic compares the synapse’s input spiketime \((x_i)\) with its post-synaptic neuron’s output spiketime \((z)\) and generates 4 control signals corresponding to the 4 cases in Table I. Case 5 is implicitly invoked when none of the other 4 cases is a 1.

The logic equations implemented for the 4 STDP cases are:

- **Case 1:** \((x_i \leq z) \cdot (x_i \cdot z)\)
- **Case 2:** \((x_i \leq z) \cdot (x_i \cdot z)\)
- **Case 3:** \((x_i \leq z) \cdot (x_i \cdot z)\)
- **Case 4:** \((x_i \leq z) \cdot (x_i \cdot z)\)

Note that \(((x_i \leq z))\) is implemented here using a much simpler temporal comparator as opposed to a binary comparator. If \(z\) arrives prior to \(x\), the output is 0; else \(x\) is allowed to pass. The gate count for this logic is \(17p + 5\).

2) **Stabilization Function Logic:** This logic selects 1 BRV from a set of finite BRVs generated by \(F(w)\), based on the synaptic weight. For \(w_{\text{max}} = 7\), there are 6 non-zero BRVs to choose from. Its logic implementation results in 12\(p\) gates.

3) **Inc/Dec Logic:** The inc/dec logic assumes 4 BRV inputs from the LFSR network corresponding to the four STDP cases. The \(\text{max}\) operation in Table I is simply implemented by ORing ‘F’ with \(min\) BRV input. The output of the stabilization logic is used along with the cases from case generation logic to generate inc and dec outputs. Its gate count comes out to be just \(7p\).

Adding the above gate counts, the STDP logic uses \(36p + 5\) gates in total. Now, the total gate count for a neuron with \(p\) synapses, including STDP, can be calculated as below.

\[
\text{Gates}_{\text{neuron-STDP}} = 102p + 8\log_2 p + 36
\]  

### C. Proposed R-STDP Implementation

This subsection introduces a variation of our proposed STDP capable of reinforcement learning (R-STDP) that uses an external reward signal to drive its learning process towards a desired direction. It involves three forms of reinforcement:

- When the column’s (non-null) output matches the desired action, \(\text{reward} = ‘1’\). It operates as per Table I except case 3 results in no synaptic weight update.
- When the column’s (non-null) output does not match the desired action, \(\text{reward} = ‘-1’\). Only Case 1 and Case 3 are performed; for Case 1, instead of incrementing the weight, it is decremented.
- When the column produces no output, i.e., no neuron spikes, \(\text{reward} = ‘0’\) and only Case 3 operates.

In effect, desired behavior is reinforced and undesirable behavior is repressed. Note that R-STDP is still applied locally to each neuron. However, the context for R-STDP is at the column or layer level due to the global reward signal.

The logic modifications required for R-STDP are rather minimal and straightforward as highlighted in Figure 11. Reward is a 2-bit signal (which encodes ‘-1’, ‘0’ and ‘1’ as ‘11’, ‘00’ and ‘01’ respectively). Unsupervised STDP is
invoked when reward is ‘10’. The total gate count for a neuron that implements R-STDP is given below:

\[
\text{Gates}_{\text{neuron}}_{\text{R-STDP}} = 106p + 8\log_2 p + 36 \quad (2)
\]

To the best of our knowledge such gate-level and hardware-efficient implementations of STDP and R-STDP have not been presented or published before.

VI. COLUMN DESIGN AND IMPLEMENTATION

In this section, the column microarchitecture is defined and a scalable implementation is presented. As shown in Figure 12, a column is a stack of excitatory neurons operating in parallel, followed by a winner-take-all (WTA) inhibition across the neurons. STDP is performed after WTA. Columns are used as functional building blocks for TNNs as shown in Figure 2.

A. Column Operation

In this subsection, we briefly explain the functionality of a typical column before moving on to its implementation in the following subsections. Figure 12 illustrates the operation of an example column with 8 RNL neurons and a total of 64 synaptic weights (each neuron has 8 input synapses and a threshold value of 8). Maximum synaptic weight value is 7. Every cross point in the synaptic crossbar is a synapse with its weight value shown inside the circle. Absence of values indicates a weight of 0. Neuron 1 (driving output \(z_1\)), has only one of its inputs \((x_5)\) driving a synapse with weight \(w_{max}\), which is lower than the threshold. Hence, it doesn’t produce an output spike (\(t = \infty\)). This value (\(\infty\)) is shown as the input to 1-WTA inhibition in Figure 12. Neuron 4 \((z_4)\) receives input spikes on three inputs with weights of \(w_{max}\); hence its body potential will cross the threshold at \(t = 2\). The output \(z_4\) is the first output spike, so it is selected as the “winner” by WTA inhibition. Hence, \(z_4 = 2\); all the other \(z_i = \infty\).

B. Winner-Take-All (WTA) Inhibition

Winner-take-all (WTA) lateral inhibition selects the first (or first \(k\) as in \(k\)-WTA) spiking neuron and allows its output spike to pass through intact, while nullifying the outputs of the other neighboring neurons. Figure 12 shows the logic diagram for 1-WTA inhibition in a column. The inhibition operation is performed by a latch-based less-than-or-equal temporal comparison unit (same as in the case generation logic). The first spike is found through a big OR gate, or a tree of small OR gates, (performing a temporal ‘min’ function) and is fed back through a latch which holds the signal at 1 until the next gamma cycle. Any input pulse coming to the latch after this signal is blocked, so only the first spikes are passed. Tie breaking is implemented as a priority-based logic that selects the first spiking neuron with the lowest index. Gate count for 1-WTA (upper bound) is \(8q + q^2\).

C. Proposed Column Implementation

As shown in Figure 4a, each column contains \(q\) excitatory neurons and a synaptic crossbar connecting the \(p\) inputs to the \(q\) neurons via \(pxq\) synapses. A column supports unsupervised learning via STDP or supervised learning via R-STDP at each of those synapses. A column also supports WTA lateral inhibition to assist in convergence of synaptic weights.

A single \((pxq)\) column with \(p\) synaptic inputs and \(q\) excitatory neurons, supported by STDP or R-STDP and WTA becomes a fully operational TNN, capable of performing inferencing and online continuous learning. Columns can also be used as building blocks for creating larger TNNs by stacking multiple columns to form a multi-column layer, as well as by cascading multiple layers into a large multi-layer TNNs as illustrated in Figure 2.

Equations (3) and (4) below provide the total gate counts for a \(pxq\) column with STDP and R-STDP, respectively. These two equations effectively characterize the hardware implementation complexity (in total gate count) of an arbitrary column with \(q\) neurons, each with \(p\) synapses, supporting unsupervised and supervised learning paradigms.

\[
\text{Gates}_{\text{col}}_{\text{STDP}} = 102pq + 8q\log_2 p + 44q + q^2 \quad (3)
\]

\[
\text{Gates}_{\text{col}}_{\text{R-STDP}} = 106pq + 8q\log_2 p + 44q + q^2 \quad (4)
\]

VII. COLUMN IMPLEMENTATION EVALUATION

Scalable designs of neurons and columns have been implemented in Verilog; synthesis results are generated based on a 45nm standard cell library using Synopsys tools. This section presents evaluation of these designs.

A. Methodology

Two types of evaluation, namely gate-level and circuit-level, are performed to determine area (A), (neuron) critical path delay (D), (column) computation time (T) and power (P) for the proposed neuron and column designs.

In gate level evaluation, we derive equations for A, D, T and P based on gate count and number of signal transitions, parameterized in terms of number of neurons (\(q\)) and number of synapses per neuron (\(p\)). The procedure is as follows: 1) Gate count is used as a surrogate for A. 2) Total number of gates in the critical path is used for D. T is calculated using the formula \(T = (t_{max} + w_{max} + 1) \times D\). For a b-bit time window, the first and last spikes can be separated by up to
TABLE II: Area, Delay and Power (ADP) Values: (in 45 nm CMOS) for a neuron with STDP relative to synapse counts.

| Synapses | Gate Count | Area [mm²] | Critical Path Delay [ns] | Power [mW] |
|----------|------------|------------|--------------------------|------------|
| 64       | 6,471      | 0.0065     | 1.93                     | 0.031      |
| 128      | 12,859     | 0.0129     | 2.16                     | 0.062      |
| 256      | 25,673     | 0.0258     | 2.41                     | 0.124      |
| 512      | 51,258     | 0.0515     | 2.64                     | 0.249      |
| 1024     | 102,432    | 0.1030     | 2.82                     | 0.497      |

TABLE III: Gate Level Equations: for A, D, T and P for a neuron with p synapses and a pxq column implementing STDP. D is for neuron and T is for column.

\[
\begin{align*}
A &= 102p + 8log_2(p) + 36 \\
D / T &= 8log_2(p) + 4 \\
P_{\text{static}} &= 102pq + 8qlog_2(p) + 44q + q^2 \\
P_{\text{dynamic}} &= 204p + 185log_2(p) + 241 \\
\end{align*}
\]

C. Neuron Circuit Level Evaluation

1) Gate Complexity Breakdown: Figure 13a shows the breakdown for various components of the neuron in terms of gate count, scaled across synapse count from 64 to 1024. We observe a near-linear scaling of total gate counts relative to the number of synapses. The gate count for the neuron body is quite small whereas the synapses incur the highest gate counts. Synapses constitute almost 50% of the entire neuron complexity and STDP logic 40% while the neuron body accounts for the remaining 10%. A neuron with 1024 synapses has a total of 102,432 gates.

2) ADP Complexity (45 nm CMOS): Table II shows the A, D and P values for a neuron with STDP-programmable synapses, for 45 nm digital CMOS technology. It also includes gate count generated from the post-synthesis output of Design Compiler. A temporal neuron with 1024 synapses consumes about 0.1 mm² area and 0.5 mW power. Its critical path delay of 2.82 ns maps to a maximum clock frequency of 354.6 MHz. This is the maximum rate at which the actual hardware clock aclk can be clocked. The near-linear scaling of A and P, and logarithmic scaling of D, relative to synapse counts, can be verified based on the values in the table. The gate count from post-synthesis results in Table II can be easily corroborated by inserting appropriate values for p in Equation (1).

D. Column Circuit Level Evaluation

A, T and P gate level equations for a pxq column having q neurons, each with p synapses, implementing unsupervised STDP are given in Table III. These equations (area/power) can be easily adapted for R-STDP using Equation (4).

Time for a single computational cycle (T) or the gamma cycle can be derived as mentioned in Section VII-A using \( w_{\text{max}} = t_{\text{max}} = 7 \). This is because a column’s critical path is the same as that for a neuron. For estimating dynamic power consumption, each gate in WTA inhibition logic is assumed to have 2 transitions (including 1 after reset).

E. Column Circuit Level Evaluation

Figure 13b shows the gate count breakdown for the three components of a TNN column with 16 neurons, varying synapse count for each neuron from 64 to 1024. Excitatory
TABLE IV: Area, Computation Time, and Power Consumption Values: (in 45 nm CMOS) for three column sizes of: 64x8, 128x10, 1024x16, implementing STDP and R-STDP learning rules. R-STDP incurs minimal overhead over STDP.

column constitutes just above 60% of the entire column complexity while STDP logic incurs close to 39%. Interestingly, WTA lateral inhibition incurs negligible gate count. As can be seen from Figure [13], total column gate count again exhibits a near-linear scaling relative to the total synapse count. In Table [IV], we present 45 nm CMOS results for three column configurations with both STDP and R-STDP learning rules: 1) a small 64x8 column, 2) a medium 128x10 column, and 3) a large 1024x16 column. The gamma cycles for the smallest column and the largest column are 28.95 ns (34.54 MHz) and 42.3 ns (23.64 MHz) respectively. Smallest column with STDP consumes about 0.05 mm² area and 0.25 mW power, whereas the largest column has an area and power footprint close to 1.7 mm² and 8 mW respectively. Note that R-STDP increases area and power by only 5% relative to STDP.

VIII. A MULTI-LAYER TNN PROTOTYPE

This section presents a TNN prototype design based on our TNN microarchitecture in Figure [2] using the scalable multi-neuron columns and multi-column layers as building blocks. This 2-layer TNN prototype has one unsupervised layer followed by a supervised layer as shown in Figure [15] with dimensions: TNN\{[625x(32x12)]+[625x(12x10)]\}.

We compare this prototype network against the 3-layer baseline network shown in Figure [14] from Mozafari et al. [23]. This baseline network represents the current state-of-the-art TNN model. As was used in [23] for the baseline network, we also use the MNIST data set [14] for our benchmarking. MNIST data set of 28x28 images is widely used for image classification. We compare the two networks based on MNIST training time, accuracy, and total synapse count. Note that the goal here is not to propose a state-of-the-art TNN accelerator for MNIST, but merely to demonstrate the potential of TNNs for future TNN-based sensory processing units.

We also present hardware metrics for our TNN prototype, including total gate count, die area, compute time, and power consumption, with technology scaling from 45 nm to 7 nm.

A. Evaluation of Our TNN Prototype

Baseline Network Model: See Mozafari et al. [23]. The baseline network as shown in Figure [14] consists of two unsupervised layers followed by a supervised layer for classification. For direct comparison, we convert this network from the original feature-map organization used by the authors to an equivalent TNN column organization.

Given a convolution layer in the baseline model, the number of equivalent TNN columns can be determined using the input feature map size, kernel window size and the stride. The number of neurons per column (#outputs) is equal to the number of output feature maps, since each neuron in a column learns a distinctive feature. The number of synapses per neuron in each column (#inputs) is given by the number of input feature maps and the kernel window size. Essentially, the column organization is a structural transpose of the feature map organization with the same total synapse count.

Parameters for all the layers of the baseline network are shown in Figure [14] after conversion to the equivalent column organization. Each layer has a pooling sub-layer attached. This network uses 5x5 receptive fields (RF) and DoG filtering with 6 channels. The complexity of each layer is based on the synapses count (#synapses), which can be calculated using dimension (#inputs x #outputs) of each column, and the number of columns in each layer (#columns); see Table [V]

The entire 3-layer baseline network employs over 36 million synapses. Although not explicitly constructed as an online system, it can be adapted to operate online in a straightforward way. The first layer is trained with 100K input patterns, the second layer with 200K, and the supervised layer requires 40M. The best MNIST accuracy achieved after 40M training inputs is 97%. We have validated these results in the Spyke-Torch [22] based functional simulator.

Prototype TNN Network: This network, shown in Figure [15], consists of two layers, one unsupervised and one supervised. It uses 4x4 receptive fields (RF) with On/Off encoding and stride of 1 across the 28x28 image resulting in 25x25 RFs. Hence, Layer 1 has 625 columns (one for each RF) of size...
TABLE V: Complexity Comparison: 3-layer baseline model vs. our 2-layer TNN prototype in terms of total synapse counts.

| Model       | # Inputs | # Outputs | # Columns | # Synapses |
|-------------|----------|-----------|-----------|------------|
| Baseline    | Layer 1  | 150       | 30        | 784        | 3,528      |
|             | Layer 2  | 270       | 250       | 196        | 13,290     |
|             | Layer 3  | 6250      | 200       | 16         | 20,000     |
|             | Total    |           |           |            | 36,758     |
| TNN Prototype| U1:Layer 1 | 32       | 12        | 625        | 240        |
|             | S1:Layer 2 | 12       | 10        | 625        | 73         |
|             | Total    |           |           |            | 315        |

TABLE VI: Technology Scaling (from 45nm to 7nm) of the TNN Prototype (complexity: 32M gates or 128M transistors).

| Technology | Transistor Density | Area [mm²] | Comp. Time [ns] | Power [mW] |
|------------|--------------------|------------|-----------------|------------|
| 45 nm      | 4 MT/mm²           | 32.61      | 43.05           | 154.36     |
| 28 nm      | 10 MT/mm²          | 13.04      | 27.23           | 61.74      |
| 16 nm      | 22 MT/mm²          | 5.93       | 18.36           | 28.06      |
| 10 nm      | 46 MT/mm²          | 2.84       | 12.70           | 13.42      |
| 7 nm       | 85 MT/mm²          | 1.54       | 9.34            | 7.26       |

(32x12). Layer 2 has the same number of columns but each of size (12x10), and is essentially a supervised voting layer that generates 625 votes (1 or 0) for each label. The Tally (T) sub-layer of Layer 2, consists of 10 adder trees, each with 625 inputs and determines the label with the maximum votes.

We implement a fully functional model of our TNN prototype network using PyTorch [28] and therefore can make direct comparison with the baseline network at the functional level. Since [23] does not provide a direct hardware implementation, we use synapse count (dominant factor in hardware complexity) for comparison. From Table V, the total synapse count for the prototype network is only 315,000. This is two orders of magnitude less than the baseline network (36M synapses), demonstrating the efficiency of our proposed TNN microarchitecture.

TABLE VI: Technology Scaling (from 45nm to 7nm) of the TNN Prototype (complexity: 32M gates or 128M transistors).

B. Online Image Classification

In contrast to the typical epoch-based, back propagation training methods, STDP is an online training method that consumes and processes the inputs in a streaming manner and amenable to online real-time applications. Our goal is not to compete with state-of-the-art performance on MNIST, but just using it as a benchmark to validate the functionality and demonstrate the efficiency of our proposed TNN microarchitecture. The MNIST accuracy achieved by the simple two-layer network is 93%. (A deeper network, beyond the scope of this paper, can achieve 98%.) Our 93% accuracy is achieved with less than 30K training inputs (vs. 40M with the baseline), exhibiting very quick weight convergence.

B. Online Image Classification

In contrast to the typical epoch-based, back propagation training methods, STDP is an online training method that consumes and processes the inputs in a streaming manner and amenable to online real-time applications. Our goal is not to compete with state-of-the-art performance on MNIST, but just using it as a benchmark to validate the functionality and demonstrate the efficiency of our proposed TNN microarchitecture. The MNIST accuracy achieved by the simple two-layer network is 93%. (A deeper network, beyond the scope of this paper, can achieve 98%.) Our 93% accuracy is achieved with less than 30K training inputs (vs. 40M with the baseline), exhibiting very quick weight convergence.

1) **Online Classification via Centroid Formation:** Figure 16a shows the synaptic weights converged to the 10 class centroids, which resemble the corresponding digits.

2) **Fast Training Convergence:** The synaptic weights in Figure 16a and Figure 16b converged after approximately 10,000 training samples, which implies that TNNs can learn very quickly and can generalize from small datasets.

3) **Online Incremental Learning:** In this experiment, R-STDP training is first performed with only 9 classes (0 to 8) by hiding the label ‘9’, resulting in the converged weights shown in Figure 16b. Then the label ‘9’ is introduced in the input sequence to illustrate the ability to dynamically learn a previously unseen class. As shown in Figure 17, the rightmost synaptic weight matrix converges to label ‘9’ after only about 500 testing samples.

TNNS have the potential for both incremental and continuous learning. Online incremental learning enables a TNN to adapt to new input data not seen before during the original (offline) training. Continuous learning allows a TNN to keep learning and improving its accuracy concurrently with inference.

C. Hardware Complexity and Technology Scaling

Based on our results from earlier sections, we can assess the hardware complexity, performance, and power of the prototype network (or any other arbitrary multi-layer TNN network). Using the gate-level implementation of our scalable column design and Equation 3 and Equation 4, we can compute the gate-level complexity for the prototype TNN as: 32M gates or 128M transistors. Further breakdown of the gate counts for the layers are: 24.12M (U1), 7.91M (S1), and 31.25K (T).

Using 45 nm standard cell library and transistor density of 4 MT/mm², we have 32.61 mm² die area, 43.05 ns compute time, and 154.36 mW power consumption for the prototype.
TNN as shown in the first row of Table VI. Using a common approach based on the scaling of transistor density from one process node to the next, we can estimate the die area, compute time, and power consumption for our TNN prototype as we scale the CMOS process from 45 nm down to 7 nm.

Technology scaling for area and power can be performed by multiplying them by the ratio of transistor densities between the source and target nodes. For critical path delay, we use the square root of the above ratio. As can be seen in Table VI across the five technology nodes, ranging from 45nm to 7nm, area and power for each subsequent generation are scaled by a factor of 0.4, 0.45, 0.48 and 0.54, respectively. The critical path delay scales with the square root of these scaling factors, namely, 0.63, 0.67, 0.69 and 0.73, respectively.

With the current 7 nm CMOS process, the entire 2-layer TNN prototype takes up only 1.54 mm² of die area, consumes just 7.26 mW, and incurs only 9.34ns for each compute cycle. Compared to 7 nm SoCs in high-end smartphones, with typical die sizes of over 100 mm², this is less than 2% of the mobile SoC die area, and less than 1% of its 1W power budget. Note that our TNNs do not perform any matrix multiplication. There are no MAC units or SRAM arrays. This is a whole new way of implementing online sensory processing units.

IX. CONCLUDING REMARKS AND FUTURE RESEARCH

We take a bottom up approach by examining the hierarchical organization of biological neocortex systems to formulate an analogous hierarchical organization for TNNs. Based on this approach, we propose the neuromorphic abstraction levels shown in Figure 18. This paper focuses on Levels 3 and 4.

Two key questions: “Can such TNNs perform useful functions?” and “Can such TNNs be implemented using standard CMOS?” The work by Mozafari et al. [23] contributes towards the first question. In this paper, we focus on and address the second question. We implement a scalable column, with STDP and R-STDP, in Verilog, and obtain the post-synthesis results for the 45 nm CMOS process. We can use these two column types to create two layer types for either unsupervised (STDP) or supervised (R-STDP) learning, as shown in Figure 5.

The key contributions of this paper include the TNN microarchitecture and implementation model embodied in Equations 3 and 4 and Tables I, II, III and IV. Equations 3 and 4 characterize the gate level complexity of TNN columns, from which we can compute the complexity of any multi-column layer (with STDP or R-STDP) and any multi-layer TNN. Table III provides process-independent characteristic equations (based on our gate level designs) for die area (A), delay/compute time (D/T), and power consumption (P). Tables I and IV provide actual data on A, D/T, and P for the neuron and column models for the 45 nm process node. Based on these two tables, we can estimate these metrics for any multi-layer TNN in 45 nm CMOS. Using standard process scaling parameters, we can also estimate these metrics for any TNN design at any process node, as illustrated in Table VI.

Some key observations from this work include:

1) About 50% of a neuron’s complexity comes from the synaptic weights, 40% from STDP and the remaining 10% the neuron body.
2) A column’s complexity is mainly determined by its excitatory column’s size and configuration, and scales near-linearly with respect to the product of the neuron count and the synapse count.
3) Area and power for a neuron and a column scale linearly with the total number of synapses $p x q$, whereas critical path delay scales logarithmically with the number of synapses per neuron $p$. Since the neuron body mainly determines the critical path delay, the number of neurons $q$ in a column has minimal affect on the critical path in a column or even in a multi-layer TNN.

This work represents only an initial step in a very promising path for follow up research. We take a bottom up approach, starting with Levels 3 and 4 of the hierarchy of neuromorphic abstraction levels in Figure 18. We plan to move up into Levels 1 and 2 in our future research.

We believe that our TNN microarchitecture and direct CMOS implementations can facilitate the design of highly energy efficient, online, and edge-native, sensory processing units capable of incremental and continuous learning that can be incorporated into mobile and edge computing systems as well as always-on IoT sensor processing devices. The CMOS implementation results in this paper should be viewed as a first opportunistic attempt, using readily-available and existing design methods and tools. There are plenty of potential improvements and new innovations, in both TNN designs and TNN design frameworks, waiting to be harvested.

Current computing demand for AI/ML workloads is increasing at the rate of 10x per year or doubling every 3.4 months [26]. Moore’s law, at best, is only doubling every 2 years. The gap between the increasing computing demand and the best that computing hardware can provide is widening at the rate of 8x per year or 500x every 3 years. This trend is not sustainable. We need a new, much more brain-like, type of computing systems that are several orders of magnitude more complexity and energy efficient (see “Green AI” [30]). Based on this work, we believe practically useful online sensory processing units that consume only few mm² of die area and few mW of power are quite feasible in the near future.
REFERENCES

[1] E. Baek, H. Lee, Y. Kim, and J. Kim, “Flexlearn: Fast and highly efficient brain simulations using flexible on-chip learning,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Computer Architecture, 2019, pp. 304–318.

[2] G.-q. Bi and M.-m. Pozo, “Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type,” Journal of neuroscience, vol. 18, no. 24, pp. 10464–10472, 1998.

[3] J. M. Brader, W. Senn, and S. Fusi, “Learning real-world stimuli in a neural network with spike-driven synaptic dynamics,” Neural computation, vol. 19, no. 11, pp. 2881–2912, 2007.

[4] D. A. Butts, C. Weng, J. Liu, C.-I. Yeh, N. A. Lesica, J.-M. Alonso, and G. B. Stanley, “Temporal precision in the neural code and the timescales of natural vision,” Nature, vol. 449, pp. 7158, pp. 92–95, 2007.

[5] M. Davies, N. Srinivasas, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain et al., “Loihi: A neuromorphic manycore processor with on-chip learning,” IEEE Micro, vol. 38, no. 1, pp. 1–12.

[6] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” Frontiers in computational neuroscience, vol. 9, p. 99, 2015.

[7] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, “A 0.086-mm2 12.7-pp/sop 64k-synapse 256-neuron online-learning digital spike neuromorphic processor in 28-nm cmos,” IEEE transactions on biomedical circuits and systems, vol. 13, no. 1, pp. 145–158, 2018.

[8] W. Gerstner, R. Kempter, J. L. Van Hemmen, and H. Wagner, “A neuronal learning rule for sub-millisecond temporal coding,” Nature, vol. 383, no. 6595, pp. 76–78, 1996.

[9] C. M. Gray, P. König, A. K. Engel, and W. Singer, “Oscillatory responses in cat visual cortex exhibit inter-columnar synchronization which reflects global stimulus properties,” Nature, vol. 338, no. 6213, pp. 334–337, 1989.

[10] R. Guyonneau, R. VanRullen, and S. J. Thorpe, “Neurons tune to the earliest spikes through stdp,” Neural Computation, vol. 17, no. 4, pp. 859–879, 2005.

[11] D. Hebb, “Organization of behavior. new york: Wiley,” J. Clin. Psychol., vol. 6, no. 3, pp. 335–307, 1949.

[12] W. M. Kistler, W. Gerstner, and J. L. v. Hemmen, “Reduction of the hodgkin-huxley equations to a single-variable threshold model,” Neural computation, vol. 9, no. 5, pp. 1005–1015, 1997.

[13] J. Knudsen, “Nangate 45nm open cell library; CDNLive, EMEA, 2008.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] J. H. Lee, T. Delbruck, and M. Pfeiffer, “Training deep spiking neural networks using backpropagation,” Frontiers in neuroscience, vol. 10, p. 508, 2016.

[16] W. Levy and O. Steward, “Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus,” Neuro-science, vol. 8, no. 4, pp. 791–797, 1983.

[17] T. Liu, Z. Liu, F. Lin, Y. Jin, G. Quan, and W. Wen, “Mt-spike: A multilayer time-based spiking neuromorphic architecture with temporal error backpropagation,” in 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2017, pp. 450–457.

[18] Z. F. Mainen and T. J. Sejnowski, “Reliability of spike timing in neocortical neurons,” Science, vol. 268, no. 5216, pp. 1503–1506, 1995.

[19] T. Masquelier and S. J. Thorpe, “Unsupervised learning of visual features through spike timing dependent plasticity,” PLoS computational biology, vol. 3, no. 2, 2007.

[20] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura et al., “A million-spiking-neuron integrated circuit with a scalable communication network and interface,” Science, vol. 345, no. 6197, pp. 668–673, 2014.

[21] H. Mostafa, “Supervised learning based on temporal coding in spiking neural networks,” IEEE transactions on neural networks and learning systems, vol. 29, no. 7, pp. 3227–3235, 2017.

[22] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, and T. Masquelier, “Spyketorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron,” Frontiers in neuroscience, vol. 13, 2019.

[23] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, “Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks,” Pattern Recognition, vol. 94, pp. 87–95, 2019.

[24] E. Neftci, “Stochastic synapses as resource for efficient deep learning machines,” in 2017 IEEE International Electron Devices Meeting (IEDM). IEEE, 2017, pp. 11–1.

[25] P. O’Connor and M. Welling, “Deep spiking networks,” arXiv preprint arXiv:1602.08323, 2016.

[26] OpenAI, “AI and Compute,” https://openai.com/blog/ai-and-compute/ (May 2018).

[27] B. Parhami and C.-H. Yeh, “Accumulative parallel counters,” in Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers, vol. 2. IEEE, 1995, pp. 966–970.

[28] P. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., “Pytorch: An imperative style, high-performance deep learning library,” in Advances in neural information processing systems, 2019, pp. 8026–8037.

[29] D. Querlioz, O. Bichler, P. Dollfus, and C. Garnat, “Immunity to device variations in a spiking neural network with memristive nanodevices,” IEEE Transactions on Nanotechnology, vol. 12, no. 3, pp. 288–295, 2013.

[30] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green ai. corb abs/1907.10597 (2019),” arXiv preprint arXiv:1907.10597, 2019.

[31] J. E. Smith, “Space-time computing with temporal neural networks,” Synthesis Lectures on Computer Architecture, vol. 12, no. 2, pp. i–215, 2017.

[32] J. E. Smith, “Reverse-engineering the brain: A computer architecture grand challenge,” ISCA Tutorial, 2018.

[33] J. E. Smith, “Space-time algebra: A model for neocortical computation,” in 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2018, pp. 289–300.

[34] J. E. Smith, “(newtonian) space-time algebra,” arXiv preprint arXiv:2001.04242, 2019.

[35] J. E. Smith, “A roadmap for reverse-architecting the brains neocortex,” FCRC Plenary Keynote, 2019.

[36] A. Tavanaei and A. S. Maida, “Multi-layer unsupervised learning in a spiking convolutional neural network,” in 2017 International Joint Conference on Neural Networks (IJCNN). IEEE, 2017, pp. 2023–2030.

[37] A. Tavanaei and A. S. Maida, “A spiking network that learns to extract spike signatures from speech signals,” Neurocomputing, vol. 240, pp. 191–199, 2017.

[38] S. J. Thorpe, “Spike arrival times: A highly efficient coding scheme for neural networks,” Parallel processing in neural systems, pp. 91–94, 1990.

[39] S. J. Thorpe and M. Imbert, “Biological constraints on connectionist modelling,” Connectionism in perspective, pp. 63–92, 1989.

[40] R. VanRullen, R. Guyonneau, and S. J. Thorpe, “Spike times make sense,” Trends in neurosciences, vol. 28, no. 1, pp. 1–4, 2005.

[41] M. Mozafari, M. Ganjtabesh, A. Nowzari-Dalini, S. J. Thorpe, and T. Masquelier, “Bio-inspired digit recognition using reward-modulated spike-timing-dependent plasticity in deep convolutional networks,” Pattern Recognition, vol. 94, pp. 87–95, 2019.