TNN7: A Custom Macro Suite for Implementing Highly Optimized Designs of Neuromorphic TNNs

Harideep Nair, Prabhu Vellaisamy, Santha Bhasuthkar, and John Paul Shen
ECE Department, Carnegie Mellon University, {hpnair, pvellais, sbhasuth, jpshen}@andrew.cmu.edu

***Abstract*—Temporal Neural Networks (TNNs), inspired from the mammalian neocortex, exhibit energy-efficient online sensory processing capabilities. Recent works have proposed a microarchitecture framework for implementing TNNs and demonstrated competitive performance on vision and time-series applications. Building on these previous works, this work proposes *TNN7*, a suite of nine highly optimized custom macros developed using a predictive 7nm Process Design Kit (PDK), to enhance the efficiency, modularity and flexibility of the TNN design framework. TNN prototypes for two applications are used for evaluation of TNN7. An unsupervised time-series clustering TNN delivering competitive performance can be implemented within 40 uW power and 0.05 mm^2 area, while a 4-layer TNN that achieves an MNIST error rate of 1% consumes only 18 mW and 24.63 mm^2. On average, the proposed macros reduce power, delay, area, and energy-delay product by 14%, 16%, 28%, and 45%, respectively. Furthermore, employing TNN7 significantly reduces the synthesis runtime of TNN designs (by more than 3x), allowing for highly-scaled TNN implementations to be realized.**

***Index Terms*—temporal neural networks, custom macros for temporal functions, neuromorphic sensory processing units**

I. Introduction

Deep Neural Networks (DNNs) have achieved state-of-the-art performance on diverse applications involving sensory processing tasks such as computer vision and speech recognition [4]. However, the computing demand for DNNs has been increasing exponentially and is on a highly unsustainable path in terms of computational, economic and environmental costs [10]. In contrast, Temporal Neural Networks (TNNs) [8], [9], a special class of Spiking Neural Networks (SNNs), strive to mimic biological neural networks with the goal of achieving both brain-like capability and brain-like energy efficiency.

Inspired by brain's *temporal* computational paradigm, TNNs are based on a rigorous *space-time* algebra [8] and use precise spike timings to represent and process information [12]. Unlike DNNs that utilize compute-intensive tensor processing, TNNs do not involve complex linear algebraic computation and employ simple feed-forward processing based on spikes and their timing relationships. Furthermore, TNNs are capable of online continuous learning using biologically-plausible local learning algorithms called Spike Timing Dependent Plasticity (STDP), unlike backpropagation-fueled DNNs that have a strict bifurcation between training and inference phases.

These features make TNNs truly neuromorphic and therefore suitable for building extremely energy-efficient edge-native sensory processors for applications such as time-series clustering [1]. A microarchitecture framework for efficient CMOS implementation of TNNs has recently been proposed in [6]. The proposed implementation methodology utilizes two notions of temporal resolution and thereby hardware clocks: 1) *unit* clock serving as the finest temporal resolution to calibrate the spike timings within a single instance of input, and 2) a coarser resolution *gamma* clock to separate different input instances. The authors in [6] use standard off-the-shelf 45nm CMOS to implement the key TNN microarchitectural elements, namely, neurons, columns and STDP learning rules.

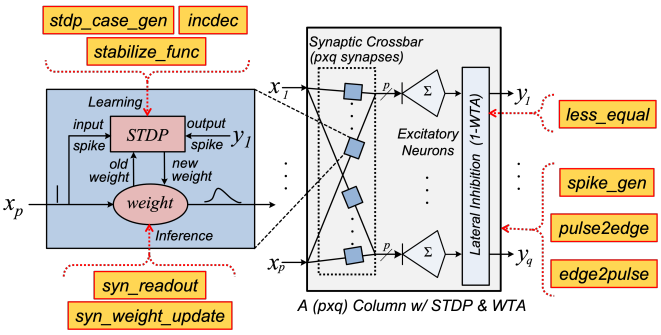


Fig. 1: Functional components of a *pxq* TNN Column and associated custom macros (highlighted in yellow)

This paper builds on and goes beyond the work in [6], exploring the potential for creating a customized cell library that utilizes inherent TNN principles to improve the power, performance and area (PPA) of TNN designs. Furthermore, this work serves as the first step towards creating a scalable design framework and toolsuite for building TNN-based neuromorphic processors. We make four key contributions: 1) the TNN design process, including gate-level implementations, is replicated in 7nm predictive CMOS using the ASAP7 Process Design Kit (PDK) [2], and post-synthesis Power-Performance-Area (PPA) results are reported; 2) a set of nine new highly-optimized custom macro extensions to ASAP7, called *TNN7*, that can be used for implementing highly energy-efficient parameterizable TNNs is proposed; 3) significantly improved scaling of PPA as well as synthesis runtime for larger design sizes, achieved by TNN7, is demonstrated; and 4) the hardware complexities of TNN prototypes in [9] for image classification, and [1] for unsupervised time-series clustering, are evaluated and shown to achieve significant improvements using the custom macro extensions, demonstrating the potential of TNNs for energy-efficient sensory processing with online learning.

The proposed nine macros have been designed to target and optimize the primary TNN building blocks, or TNN columns.

TABLE I: Proposed Custom Macros

TNN Units	Proposed Macros	Function Description	Figure Label
Synaptic Response	*syn_readout*	Perform RNL readout	Fig. 2
	syn_weight_update	Perform weight update	Fig. 3
WTA	*less_equal*	Perform temporal inhibit	Fig. 4
STDP	*stdp_case_gen*	Control STDP cases	Fig. 5
	incdec	Control update direction	Fig. 6
	stabilize_func	Stabilize weights bimodally	Fig. 7
Utility	*spike_gen*	Perform spike encoding	Fig. 8
	pulse2edge	Convert from pulse to edge	Fig. 9
	edge2pulse	Convert from edge to pulse	Fig. 10

Fig. 1 illustrates the custom macros in a typical *p*x*q* column (key TNN building block) with *p* synapses per neuron and *q* such neurons, followed by winner-take-all (1-WTA) lateral inhibition. As majority of the TNN computation occurs in the synaptic crossbar, five macros are dedicated for synapses (two for synaptic inference or response function generation and three for STDP local learning). These synapses then feed into corresponding neuron bodies which perform response function summation through adder trees. Another macro enables the key comparison operation in WTA and the remaining three macros serve more generic utility purposes (e.g. spike encoding).

These macros (elaborated in Section III) are summarized in Table I along with their functional descriptions and schematic figure labels. It should be noted, these custom macros can be generalized beyond use in the microarchitecture model in [6], and can serve as the foundation for building generic temporal functions based on *space-time* algebra [8]. To the best of our knowledge, this is the first work that proposes custom macros for highly efficient scalable CMOS implementation of TNNs.

II. Design Framework & Methodology

The proposed TNN7 custom cells are developed as hard macros using an open-source 7nm predictive Process Design Kit (PDK), called ASAP7 [2]. This section describes the ASAP7 library and the CAD design flow used in this work.

A. Framework

ASAP7 [2] is an academically certified, foundry agnostic, predictive PDK based on 7nm finFET technology. This involves a standard cell library and a collection of rule-sets for physical verification - design rule checks, layout vs. schematic, and parasitic extraction. The electrical activity of the transistor models is scaled from the BSIM-CMG SPICE models [3], which captures advanced trends in the finFET industry. ASAP7 offers transistor device models at four threshold voltages (SLVT, LVT, RVT and SRAM), and three process corners, typical-typical (TT), slow-slow (SS) and fast-fast (FF).

In this work, following selections are used for the design of custom macros: 1) RVT device models with nominal operating conditions at TT corner (0.7V supply voltage and 25°C operating temperature), 2) composite current source (CCS) modeling for timing files, and 3) Cadence/Mentor Graphics toolchain for logic synthesis, schematic, layout and characterization.

B. Methodology

In developing the custom macros, Cadence tool suite is used as follows: 1) *Genus* for register-transfer level (RTL) logic synthesis, 2) *Virtuoso* for schematics and layouts, 3) *Liberate* for characterization of the macros and generating Liberty (.lib) timing files, and 4) *Abstract* for generating Liberty Exchange Format (.lef) files of the macros. Layout verification, including Layout Versus Schematic (LVS) and Design Rule Check (DRC), is performed using Mentor Calibre and the resulting LVS & DRC-clean Graphic Data Stream (GDS) files are imported to *Abstract*. Moreover, Calibre Parasitic Extraction (PEX) tool reads the layout and generates the extracted netlist which is then used for Spectre simulations in *Liberate*.

In order to report the optimization gains presented in Section IV, following steps are adopted: 1) *Genus* is used to synthesize the original functional modules from [6] with the ASAP7 standard cell library and establish the baseline values; 2) TNN7 macro equivalent of the original modules are designed by either (i) structurally optimizing at the microarchitectural level, or (ii) creating mixed-signal circuits from scratch in *Virtuoso*; 3) *Genus* is used to resynthesize the modules by replacing the ASAP7 standard cells with the TNN7 .lib and .lef files (obtained from *Liberate* and *Abstract*), to obtain post-synthesis area, power and delay. These values are then compared against the ASAP7-based post-synthesis values to compute the corresponding improvements.

III. TNN7 Custom Macro Cells

This section describes the circuit-level design of the proposed nine macros and their functionalities in detail. The macros are segregated into *TNN functionality cells*, that perform exclusive TNN functions, and *utility cells*, that perform generic functions like spike encoding.

A. TNN Functionality Cells

This subsection describes the six macros implementing synaptic response, WTA and STDP. The following notations are used for the two hardware clocks introduced in Section I - *aclk* for the unit clock and *gclk* for the gamma clock.

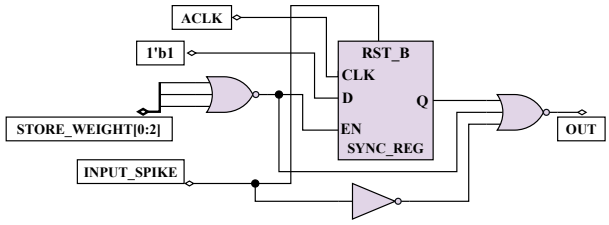


Fig. 2: *syn_readout* macro

1) syn_readout and syn_weight_update: As noted in [6], synapses constitute majority of the hardware complexity in TNNs. Hence, in this work, main synaptic functions are identified, optimized and modularized into custom macros. The two key synaptic functions of response function generation and weight update are implemented as *syn_readout* (Fig. 2)

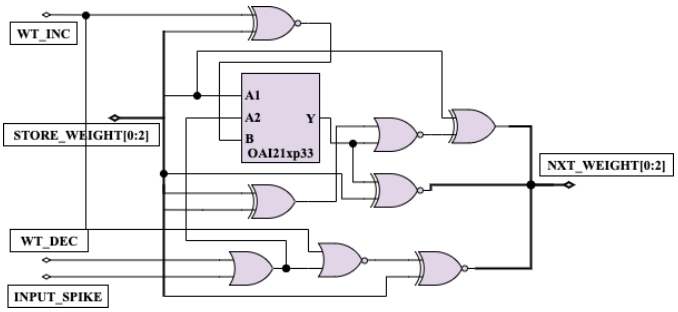


Fig. 3: *syn_weight_update* macro

and *syn_weight_update* (Fig. 3) macros respectively. When an input spike pulse arrives, the synaptic weight undergoes a unit decrement every cycle, until it wraps around to the original value. During this process, the *syn_readout* macro takes in the weight value every cycle and asserts the output until the weight reaches zero, and then deasserts it. This parallels the unary-coded ramp-no-leak (RNL) response function in [6]. The *syn_weight_update* macro controls the weight decrementing process during readout, and updates the synaptic weight during "learning", via the STDP-based control signals (*WT_INC* and *WT_DEC*). Only one of the control signals is active at a time and performs either unit increment or decrement. Note that the *syn_weight_update* macro merely updates the weight based on external control signals; the control signals are generated by input spike (inference) and the three STDP macros (learning).

This modular approach to designing synapses provides flexibility to implementing TNN frameworks. For example, the response function can be changed by modifying *syn_readout* while keeping the other macros intact. This flexibility adds to the diversification of TNN models for diverse applications.

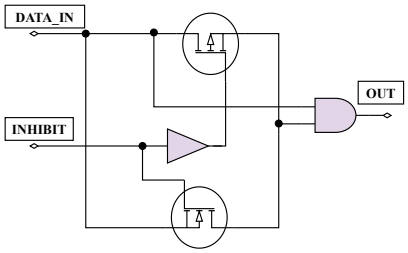


Fig. 4: *less_equal* macro

2) less_equal: The *less_equal* macro (Fig. 4) models the temporal inhibitor and functions as the basic unit for WTA inhibition. More generally, it implements the temporal operation of "less_equal" from space-time algebra [8] and hence is widely used in the TNN design framework. The input data (DATA_IN) value is propagated to the output if and only if it arrives earlier or at the same time as INHIBIT; else, it is suppressed. This module's functionality can be achieved by using a single transistor [11]. However, to mitigate the high leakage current observed during the cell's characterization, a pair of NMOS and PMOS transistors is employed.

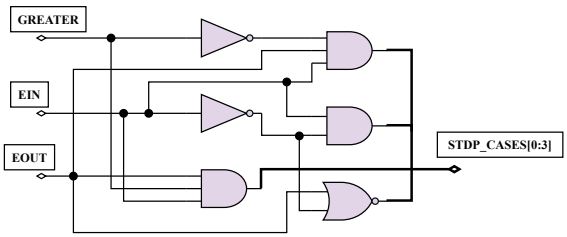


Fig. 5: *stdp_case_gen* macro

3) stdp_case_gen: The *stdp_case_gen* macro generates the essential control signal outputs, corresponding to the four STDP cases from Table I in [6]. As shown in Fig. 5, it takes in the negated output of *less_equal* (GREATER) and input/output spikes represented as edge transitions (EIN/EOUT), and generates a one-hot encoded output for the STDP cases. When both input and output spikes are absent, the output is zero, resulting in no weight update during STDP.

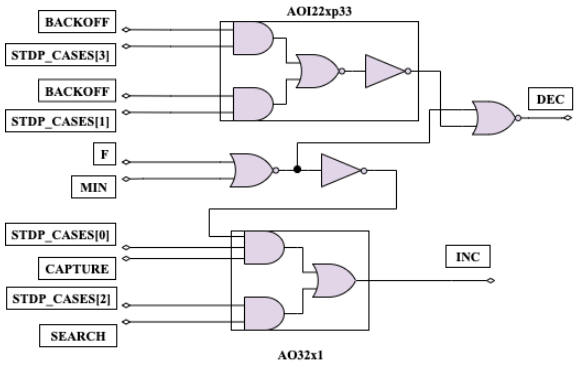


Fig. 6: *incdec* macro

4) incdec: The *incdec* (Fig. 6) macro takes in the STDP cases and Bernoulli random variables (BRVs) as inputs (as in [6]), and generates control signals for driving the local synaptic weight update process. It consists of AND-OR-INVERT (AOI) cells that activates INC for STDP cases 0 and 2, and activates DEC for cases 1 and 3, if the BRV is one. It is important to note that the modularity in STDP logic (due to *stdp_case_gen* and *incdec*) allows for easy modification of STDP rules.

5) stabilize_func: This macro (Fig. 7) is responsible for selecting the appropriate BRVs as per the stabilization function in [6], and plays a key role in establishing weight convergence. It is architected as an 8:1 multiplexer module with a hierarchy of Gate Diffusion Input (GDI) cells [5], each acting as a 2:1 multiplexer. The 2:1 GDI multiplexers utilize just two transistors, however suffer from degraded output levels. This is corrected by applying level restorers at the output, making the final design both robust and highly efficient.

B. Utility Cells

The remaining three macros are utility cells generalized to perform broader functions within the TNN framework such as spike encoding, synchronization, etc. They are detailed below.

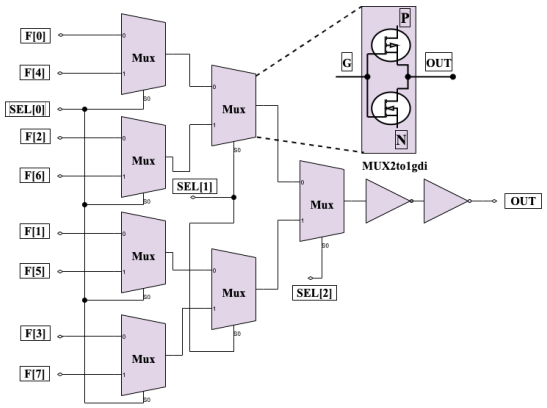


Fig. 7: *stabilize_func* macro

1) spike_gen: The *spike_gen* macro (Fig. 8) plays a key role in spike encoding. It implements the combinational logic associated with a 3-bit counter used to convert input pulses of any width to an 8 cycle-wide output pulse (for 3-bit synaptic weights). As demonstrated in [6], this spike encoding is central to the ramp-no-leak (RNL) functionality of the compact synapse design used in the TNN framework, and can be easily extended to generalize for arbitrary pulse widths.

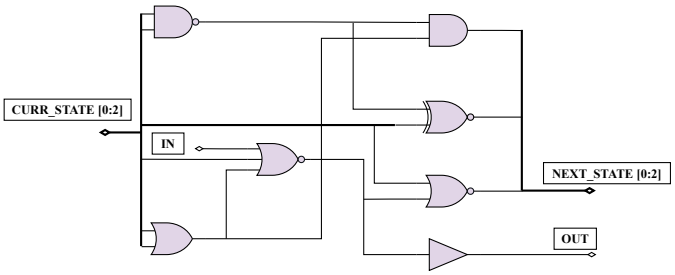


Fig. 8: *spike_gen* macro

2) pulse2edge and edge2pulse: TNN implementations utilize edge-encoded signals (i.e., encoded as edge transitions from 0→1) for performing various temporal operations. The *pulse2edge* macro (Fig. 9) transforms an incoming pulse signal into an edge signal (lasting until the end of current *gclk* cycle), and is used extensively across the TNN framework. On the contrary, *edge2pulse* macro (Fig. 10) outputs a pulse lasting one *aclk* cycle as soon as an edge signal arrives at its input. It is typically used to produce internal reset pulses from *gclk* to synchronize the sequential blocks in the datapath.

All nine macros have been carefully designed to use minimal number of gates and transistors to achieve their corresponding functionalities. In order to further reduce cell area, we perform diffusion layer overlapping during manual layout. Table II reports their respective PPA metrics. In order to demonstrate their benefits, these macros are used to build various TNN prototypes as discussed in the next section.

IV. Benchmarking and Results

This section presents 7nm post-synthesis power, performance, area (PPA) results for application-specific TNN prototypes. Performance is measured in terms of computation time (time taken to process one input), and is derived from the critical path delay and the gamma period as in [6]. Area is the total cell and net area, while power includes dynamic (calculated using Cadence *Joules*) as well as leakage power.

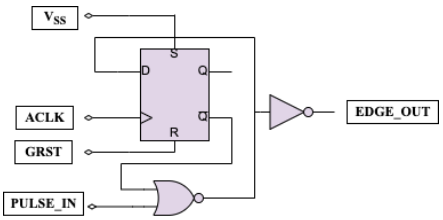


Fig. 9: *pulse2edge* macro

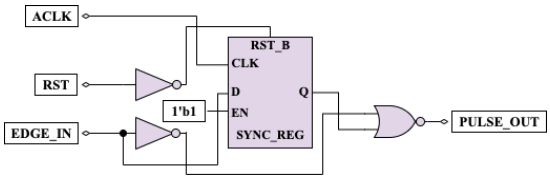


Fig. 10: *edge2pulse* macro

TABLE II: 7nm PPA for proposed custom macros

Custom Macro Name	Leakage Power (nW)	Delay (ps)	Cell Area (μm^2)
syn_readout	0.43	32	0.50
syn_weight_update	1.22	190	1.24
less_equal	0.17	30	0.17
stdp_case_gen	0.34	66	0.60
incdec	0.26	56	0.34
stabilize_func	0.12	158	0.36
spike_gen	1.46	28	1.55
pulse2edge	0.44	22	0.44
edge2pulse	0.49	58	0.61

In order to demonstrate the efficacy of the TNN7 macros, we perform benchmarking for two groups of TNN prototype designs targeting two application domains: 1) 36 single-column TNN designs for unsupervised time-series clustering on 36 UCR datasets from [1], with total synapse counts ranging from 130 to 6750; and 2) three much larger multi-layer TNN designs for MNIST digit recognition, namely, *2-layer*, *3-layer* and *4-layer* TNNs (from [9]) with total synapse counts of 389K, 1,310K and 3,096K, respectively. Following [6], an operating frequency of 100 kHz is chosen for *aclk* based on real-time operation requirement. We observe linear scaling of dynamic power with frequency and omit those results here for brevity.

A. UCR Time-Series Clustering

As shown in [1], TNN designs outperform or are competitive to state-of-the-art algorithms for unsupervised time-series clustering, averaging across the 36 UCR benchmark datasets. A specific column configuration is used for each of the 36 UCR datasets depending on the corresponding input size and number of clusters. While the hardware complexity analysis in [1] uses standard technology scaling to estimate the 7nm results from 45nm post-synthesis results, we present direct

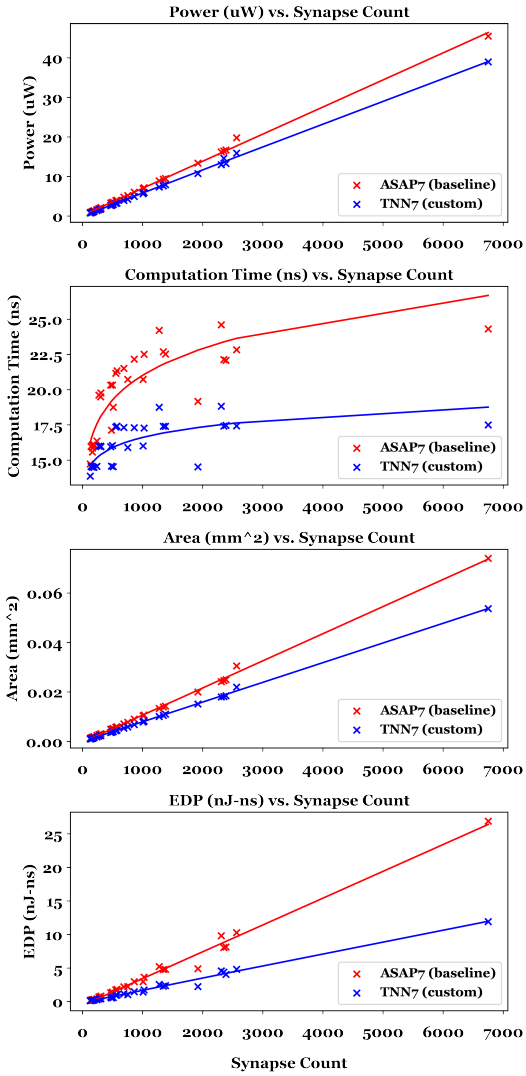


Fig. 11: ASAP7 vs. TNN7 7nm PPA scaling across synapse counts for the 36 single column TNN designs as used in [1]

post-synthesis 7nm PPA results for all 36 TNN designs and further optimize them with our custom macros.

To assess the range of PPA complexities for time-series clustering TNNs, we plot area, power, computation time, and energy-delay product (EDP), for the 36 single-column designs in Fig. 11. EDP is used here to gauge both energy-efficiency and performance. Three key results can be observed here:

1) *PPA synaptic scaling*: Area and power scale linearly with total synapse counts for both ASAP7 baseline and TNN7 custom designs, whereas computation time scales logarithmically with synapses per neuron (p). This corroborates with the characteristic scaling equations in [6]. Note that x-axis is monotonic in $p*q$ (not p), making computation time data points non-monotonic in Fig. 11.
2) *PPA improvements with TNN7*: TNN7 designs consume about 18% less power and 25% less area compared to baseline designs, and are about 18% faster. EDP improves by more than 45%, which clearly shows TNN7 designs are significantly more energy-efficient and are also faster. The gap between the two designs grows with increasing synapse count, which implies, as TNN designs grow larger, they reap even more benefits from custom macros.
3) *Potential for low-power edge-native sensory processors*: With custom macros, even the *largest* TNN column with 6,750 synapses consumes just 0.054 mm^2 area and 39 μW power. Note that this also accounts for on-chip learning via STDP, highlighting the value of proposed macros for highly energy-efficient TNN sensory processing units capable of online continuous learning.

TABLE III: ASAP7 vs. TNN7 7nm PPA comparison for three TNN prototype designs for MNIST from [9]

TNN Design	Synapse Count	Error Rate	Cell Library	Power (mW)	Comp. Time (ns)	Area (mm^2)
2-Layer	389K	7%	ASAP7	2.62	49.00	4.27
			TNN7	2.25	41.38	3.09
3-Layer	1,310K	3%	ASAP7	8.83	78.37	14.37
			TNN7	7.57	66.16	10.42
4-Layer	3,096K	1%	ASAP7	20.86	108.46	33.95
			TNN7	17.89	91.58	24.63

B. MNIST Digit Recognition

Here, we move to much larger TNN designs and evaluate three multi-layer TNN prototypes for MNIST digit recognition, with different design points in the error rate vs. hardware complexity tradeoffs. The three designs are as follows: 1) 2-layer TNN (389K synapses and 7% error) derived from ECVT in [9]; 2) 3-layer TNN (1.31M synapses and 3% error) derived from ECCVT in [9]; and 3) 4-layer TNN (3.096M synapses and 1% error) derived from ECCCVT in [9]. Table III provides 7nm PPA for these designs, derived using synaptic count scaling as in [6]. Note that "C" layers above consist of TNN7 columns, however the "VT" layers [9], that are a simpler form of TNN columns, are currently not supported within TNN7. Hence, the synaptic scaling here treats all network layers as "C", thereby providing an upper limit on the PPA complexity.

From Table III, similar PPA improvements with custom macros can be observed for these complex multi-layer TNNs (14%, 16%, and 28% improvements on power, performance and area, respectively). The 4-layer TNN with 3M synaptic weights and 99% MNIST accuracy consumes only 17.89 mW power and 24.63 mm^2 area. This TNN represents an edge-native real-time sensory processing unit that is capable of both online (MNIST-like) image-based classification and continuous learning, while consuming less than 20 mW power.

Using the survey of MNIST neural networks from [7], it can be observed that for similar accuracies, TNN-based processing units that consume a few tens of mW power are about 1000x more efficient comparing to GPUs, 100x comparing to FPGAs and 10x comparing to many state-of-the-art ASICs that consume a few hundreds of mW power. Furthermore, TNN7 enhances this scalability as it offers a lower-cost trajectory in the accuracy vs. hardware complexity tradeoff.

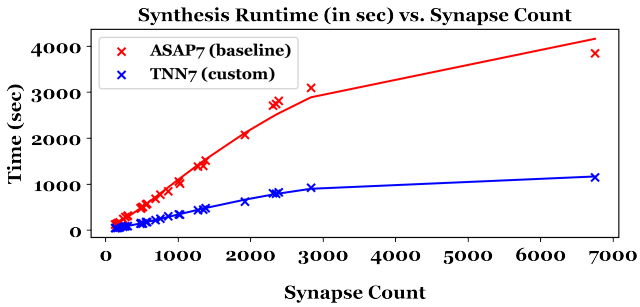
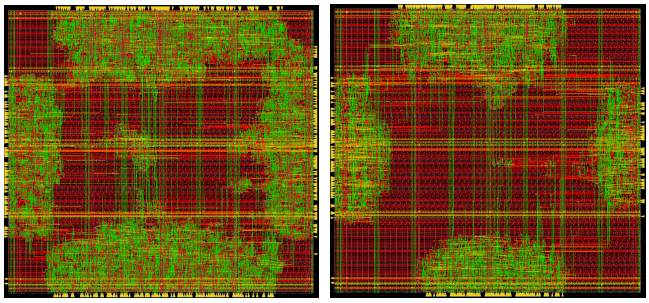


Fig. 12: ASAP7 vs. TNN7 synthesis runtime comparison



(a) ASAP7 (b) TNN7

Fig. 13: ASAP7 vs. TNN7 layouts for *82x2* column

V. Synthesis Runtime Evaluation

A further advantage to using a custom cell library is significantly faster design netlist generation. As the macro design instances are preserved and not manipulated during synthesis, it enables the synthesis tool to realize a design hierarchy by directly mapping the hard macros, thereby mitigating the combinatorial search space complexity for the optimization tool. To evaluate this benefit for TNN7, we use the following setup: Genus v19.1 is run on a server comprising of 48 Intel(R) Xeon(R) E5-2680 CPU cores with the maximum number of CPUs utilized set to 8. Synthesis was performed on the same column configurations from Section IV-A, with the TNN7 custom macros as well as without them (ASAP7 baseline).

Fig. 12 depicts the runtimes for both standard ASAP7-based and corresponding TNN7-based designs. On average, TNN7 speeds up the netlist generation (including mapping and optimization) by 3.17x with respect to the baseline ASAP7-based designs. Using TNN7, the largest column with 6750 synapses is synthesized in 926 seconds ($\sim$15 minutes), as opposed to 3849 seconds ($\sim$1 hour) for the baseline design. Fig. 12 illustrates increasing runtime benefits for TNN7 as the designs grow larger. This trend can be extrapolated beyond single columns to multi-layered networks based on synapse counts, demonstrating the scalability of TNN7 to realize deep TNNs, that would have otherwise suffered from long runtimes.

VI. Concluding Remarks

Prior works have shown that TNNs can achieve highly energy efficient brain-like sensory processing. This work develops a customized 7nm cell library, TNN7, consisting of nine new macros to enable extensive TNN design optimization. The TNN7 macros yield 14%, 16%, 28% and 45% improvements in power, performance, area and EDP, respectively. This surpasses typical area-power-delay trade-offs by achieving significant improvements in all three PPA metrics. With TNN7, competitive performance to state-of-the-art can be achieved on time-series clustering with just 40 μW and 0.05 mm^2, and on MNIST with only 17.89 mW and 24.63 mm^2. This shows the feasibility of TNN-based edge-native neuromorphic processors capable of online continuous learning.

This work can serve as a foundation for building a complete design framework and toolsuite, that can translate application-specific TNN designs from the functional level (software models) to hardware implementation and physical design. Towards that goal, our ongoing work involves open-sourcing[1] the custom macros and developing an automated RTL-to-GDSII process flow, to generate signoff layout and PPA metrics for arbitrary TNN designs. Fig. 13 illustrates both baseline and TNN7-based place-and-route layouts for the *82x2* column developed for UCR *TwoLeadECG* application as used in [1]. The layouts corroborate the efficacy of the proposed macros as the routing density in the custom design (Fig. 13b) is visibly less complex as compared to the baseline design (Fig. 13a). Furthermore, the custom library can be generalized to include the space-time primitives in [8] and thereby implement any bounded space-time function directly in CMOS.

References

[1] S. Chaudhari, H. Nair, J. M. Moura, and J. P. Shen, "Unsupervised clustering of time series signals using neuromorphic energy-efficient temporal neural networks," in *Int'l Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021.

[2] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, 2016.

[3] J. P. Duarte, S. Khandelwal, A. Medury, C. Hu, P. Kushwaha, H. Agarwal, A. Dasgupta, and Y. S. Chauhan, "Bsim-cmg: Standard finfet compact model for advanced circuit design," in *41st European Solid-State Circuits Conference (ESSCIRC)*. IEEE, 2015.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, 2015.

[5] A. Morgenshtein, A. Fish, and A. Wagner, "Gate-diffusion input (gdi)-a novel power efficient method for digital circuits: a design methodology," in *Int'l ASIC/SOC Conference*. IEEE, 2001.

[6] H. Nair, J. P. Shen, and J. E. Smith, "A microarchitecture implementation framework for online learning with temporal neural networks," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2021.

[7] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," in *Int'l Symposium on Computer Architecture (ISCA)*, 2016.

[8] J. Smith, "Space-time algebra: A model for neocortical computation," in *Int'l Symposium on Computer Architecture (ISCA)*, 2018.

[9] J. E. Smith, "A temporal neural network architecture for online learning," *arXiv preprint arXiv:2011.13844*, 2020.

[10] N. Thompson, K. Greenewald, K. Lee, and G. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.

[11] G. Tzimpragos, A. Madhavan, D. Vasudevan, D. Strukov, and T. Sherwood, "Boosted race trees for low energy classification," in *Architectural Support for Prog. Languages and Operating Systems (ASPLOS)*, 2019.

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

[1] https://github.com/prabsy96/TNN7