Eenergy Efficient Hardware Design of Neural Networks

by

Shreyas Kolala Venkataramanaiah

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved October 2018 by the
Graduate Supervisory Committee:

Jae-Sun Seo, Chair
Yu Cao
Chaitali Chakrabarti

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Hardware implementation of deep neural networks is earning significant importance nowadays. Deep neural networks are mathematical models that use learning algorithms inspired by the brain. Numerous deep learning algorithms such as multi-layer perceptrons (MLP) have demonstrated human-level recognition accuracy in image and speech classification tasks. Multiple layers of processing elements called neurons with several connections between them called synapses are used to build these networks. Hence, it involves operations that exhibit a high level of parallelism making it computationally and memory intensive. Constrained by computing resources and memory, most of the applications require a neural network which utilizes less energy. Energy efficient implementation of these computationally intense algorithms on neuromorphic hardware demands a lot of architectural optimizations. One of these optimizations would be the reduction in the network size using compression and several studies investigated compression by introducing element-wise or row-/column-/block-wise sparsity via pruning and regularization. Additionally, numerous recent works have concentrated on reducing the precision of activations and weights with some reducing to a single bit. However, combining various sparsity structures with binarized or very-low-precision (2-3 bit) neural networks have not been comprehensively explored. Output activations in these deep neural network algorithms are habitually non-binary making it difficult to exploit sparsity. On the other hand, biologically realistic models like spiking neural networks (SNN) closely mimic the operations in biological nervous systems and explore new avenues for brain-like cognitive computing. These networks deal with binary spikes, and they can exploit the input-dependent sparsity or redundancy to dynamically scale the amount of computation in turn leading to energy-efficient hardware implementation. This work discusses configurable spiking neuromorphic architecture that supports multiple hidden layers

exploiting hardware reuse. It also presents design techniques for minimum-area/-energy DNN hardware with minimal degradation in accuracy. Area, performance and energy results of these DNN and SNN hardware is reported for the MNIST dataset. The Neuromorphic hardware designed for SNN algorithm in 28nm CMOS demonstrates high classification accuracy (>98% on MNIST) and low energy (51.4 - 773 nJ per classification)[1]. The optimized DNN hardware designed in 40nm CMOS that combines 8X structured compression and 3-bit weight precision showed 98.4% accuracy at $33nJ$ per classification [2].

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Deep neural networks (DNNs) have seen exceptional success in numerous cognitive applications such as image classification [8] and speech recognition [9]. However, the large number of operations and parameters in state-of-the-art DNN algorithms have posed significant challenges for energy-efficient DNN hardware designs. In particular, devices that are constrained by limited computing resources and memory are compelling hardware implementations to use techniques to reduce the neural network size and lower the energy consumption. To accomplish this various prior works investigated methods to (1) lower the precision of activations and weights and (2) apply pruning and compression techniques for DNNs while maintaining high classification accuracy. Low-precision techniques rely on quantizing the DNN weights and activations with a small number of bits. The extreme case of DNN quantization is binarizing the weights and activations. BinaryConnect[10] pointed that binarizing the weights does not adversely affect accuracy and, in some cases, can improve the accuracy compared to non-binarized DNN. Binarized Neural Network (BNN) [11] extended the approach by binarizing both weights and activations and XNOR-Net [12] used a binarized network for ImageNet classification. Many prior works have also attempted to compress DNNs[13] through pruning of neurons and weights. However, generating a scattered sparsity may not necessarily result in acceleration on hardware [14, 15] and can also increase the storage overhead for encoding sparsity. Coarse Grain Sparsity (CGS) is proposed in [16], where static sparsity is applied to randomly selected blocks of weights throughout training. While these prior works investigated low-precision and structured compression in isolation, there has been

little work that systematically applied and optimized both techniques in a single framework. Deep compression [11] used pruning and quantization on weights; however, the sparsity remained non-structured. Prior CGS work [8] employed block-wise structured sparsity, but only quantized the weights and activations after training was complete, resulting in limited precision reduction (5-6 bit). Quantization of weights and Structured sparsity was applied throughout the training minimizing the index overhead that stores sparsity information, resulting in prominent acceleration with a low area/- energy hardware implementation. This work presents a custom energy efficient digital hardware for various combinations of low-precision and structured compression. The proposed methodology is empirically validated by implementing the inference phase of DNNs for MNIST dataset. The DNN for MNIST with 8-bit activations, 3-bit weights, and 8X CGS compression showed 98.4% accuracy at 33 nJ per classification, which is a >10X energy improvement compared to the baseline DNN. Traditionally, neural networks comprise an enormous amount of multiplication operations which contribute to a significant amount of logic power consumption. Therefore, another way to optimize neural networks for energy is to get relieved of multiplication operations. This can be achieved by using spiking neural networks comprising of spiking neurons. SNN's transfers the information through a sequence of spikes (binary messages) or timing of spikes [17]. These networks are closely related to our biological nervous systems. Since the neurons communicate through spikes, it eliminates the need for multipliers in the neuron model. However, achieving classification accuracy close to that of traditional neural networks using SNN's is a challenging task. Many SNN algorithms proposed for image recognition varies with different hidden layers and neuron models. To support various SNN algorithms, the hardware should be programmable. Along with energy efficient DNN architectures, this work also discusses configurable spiking neuromorphic architecture that supports

multiple hidden layers exploiting hardware reuse.

1.1 Motivation

Though neural networks are incredibly efficient in several applications because of their complexity, hardware implementations suffer from large silicon area, high memory, and power consumption. Multiply-and-accumulate (MAC) operations account for 99% of total operations in some neural networks and therefore, dominate both processing run-time and energy consumption [18]. Operations in neural networks exhibit a high level of parallelism. Conventional computers with Von Neumann architecture are sequential, thus making them inefficient in handling parallelism. Their performance is restricted due to frequency and power issues. This inefficiency of uniprocessors calls for the implementation of neural networks on hardware that supports the high level of parallelism like GPU's. Due to their huge power consumption researchers are motivated towards implementing custom hardware (ASIC) instead of GPU's where parallelism can be exploited and better performance, concerning power and throughput can be achieved. Around 95% of the neural network inference workloads at google datacenters are from Multi-Level perceptrons (MLP), Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN), out of which 60 % of it is from MLPs [19]. Therefore, there is a need to accelerate the neural networks for energy efficiency and better performance.

1.2 Deep Neural Networks

An artificial neuron is the fundamental building block of all artificial neural networks. It performs the weighted sum of all the input activations and weights. The sum of input-weight products is passed through an activation function which squashes

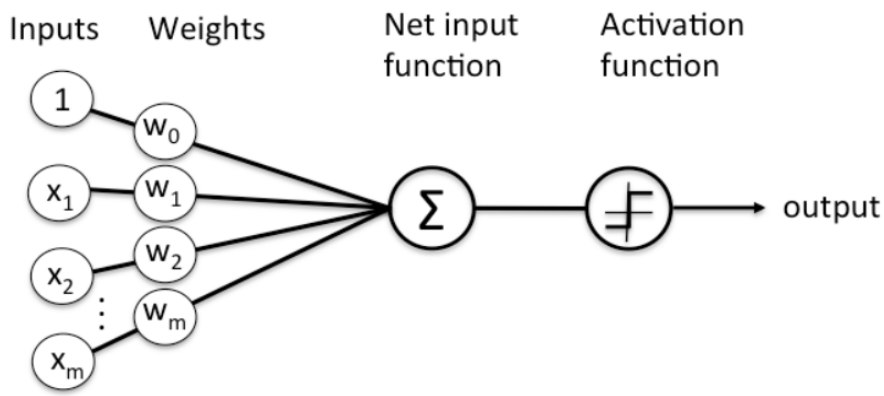

Figure 1.1: Artificial Neuron Model, Adapted from [3]

the weighted sum into bounded values (1.1).

$$Y = a(\sum_{i=1}^{N} x_i w_i) \tag{1.1}$$

Where x_i are inputs, w_i are weights , $a(x)$ is activation function and Y is the output of the neuron. Figure 1.1 shows the block diagram of simple neuron model where x and w are input values and weights. There are multiple variants of the neural network. The basic one is a multilayer perceptron which consists of multiple layers of neuron units as shown in figure 1.2. Each neuron in layer N is connected to all the neurons present in layer N+1. The Weight parameter determines the strength of these connections. The first layer, intermediate layers, and last layer are referred as the input layer, hidden layers, and the output layer respectively. In inference stage, these three layers create a feedforward architecture in which inputs are given to the input layer and their output activations are passed to the succeeding layers. Hence output activations of layer N serve as inputs for layer N+1. Each layer extracts particular features of the images and passes it on to the succeeding layer. For example, for handwritten digit classification, the first hidden layer may find the edges of the image. The next layer takes these edges and finds patterns in it. Ultimately, based

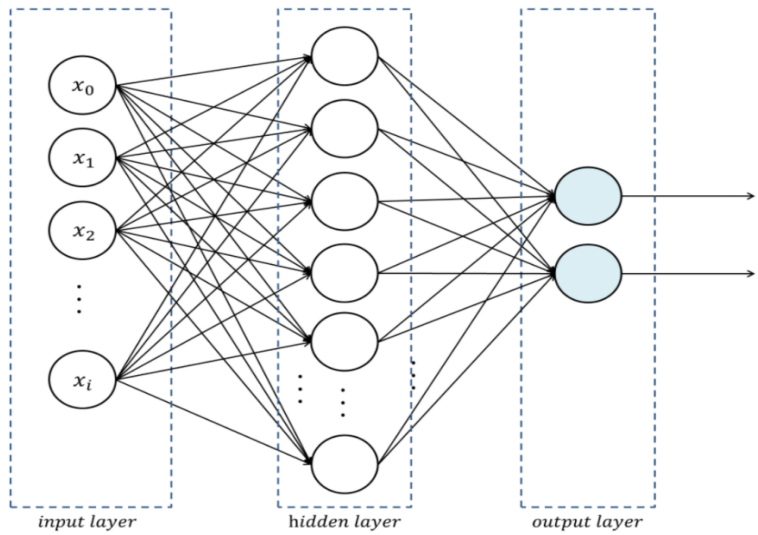

Figure 1.2: Multilayer Perceptrons with One Hidden layer, adapted from [4]

on these patterns the output layer classifies the handwritten digit.

1.3 Spiking Neural Networks

In a biological nervous system, information is processed in the form of electrical pulses called 'spikes.' These spikes travel through the axon which is linked to the dendrites of other neurons via synapses. Synapses present at all dendrites determines the strength of the connection between neurons which is defined during the learning process. An Incoming spike can increase or decrease the membrane potential of the neuron making it excitatory or inhibitory. When membrane potential of these post-synaptic neurons exceeds the threshold, they send spikes to the next set of neurons. Figure 1.3 shows a spiking neuron with many synaptic inputs and its variation of membrane potential. Spiking neural networks (SNN) represent a particular class of artificial neural networks (ANN) where neuron models communicate by sequences of spikes [20]. These neural network models have been employed for pattern recognition tasks that use more biologically plausible mechanisms. One popular approach

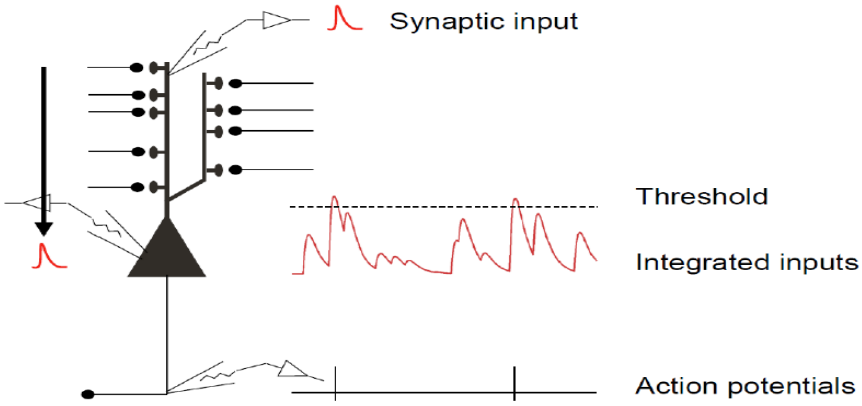

Figure 1.3: A Biological Spiking Neuron Model

to train these models is to rely on backpropagation training and converting an artificial neural network (ANN) into a spiking neural network (SNN), which is termed as rate-based learning [21, 22, 23]. While they show superior performance on tasks like the classical machine learning benchmark MNIST, this rate-based learning is not very biologically plausible and it requires many time steps to achieve high accuracy. SNN's gains power from its additional dimension; the delay variable that exists between the spikes. Along with rate-based information, it can process the temporal information between spikes.

1.4 Thesis Organization

Chapter 2 presents SNN and DNN based image classification algorithms that are implemented in hardware. This section comprises rate based and temporal based learning algorithms for SNN and structured compression technique adopted in DNN to reduce memory utilization. Hardware implementation details of these algorithms , which includes approaches and optimizations applied to design an energy efficient neural network hardware, are detailed in Chapter 3. It also covers the detailed design

of programmable hardware for MLP SNN's. Chapter 4 presents the key results and provides a discussion of the results. Chapter 5 summarizes this work and sheds light on some possible future work ideas.

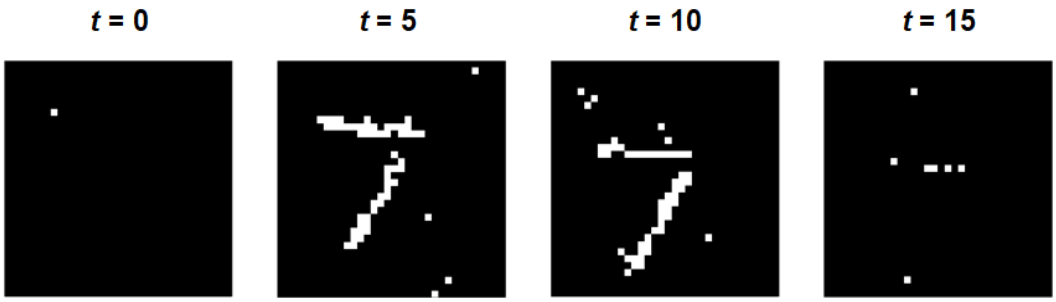

Figure 2.1: An Example of Discrete-Time N-MNIST Input Spikes (digit 7) at Four Timesteps (t = 0, 5, 10, 15), Adapted from [5]

Chapter 2

IMAGE CLASSIFICATION ALGORITHMS FOR MNIST AND NMNIST

MNIST is a dataset of handwritten images which has served as the foundation for benchmarking classification algorithms. It provides 60000 greyscale 28x28 training images and has a test set of 10000 examples [24]. The Neuromorphic-MNIST (N-MNIST) dataset is a spiking version of the original frame-based MNIST dataset. NMNISTdataset images ($34 * 34$ pixels) are generated by moving an asynchronous time-based image sensor (ATIS) in front of the MNIST images [5]. Figure 2.1 is an example of discrete-time N-MNIST input spikes of digit 7 at four time steps (t = 0, 5, 10, 15). Note that the digit seven is moving downward. In this chapter, we present SNN and DNN based image classification algorithms designed for the classification of MNIST and NMNIST datasets.

2.1 Spiking Neuron Models with Binary Activations

A spiking neuron model provides a formulation of how a neuron processes the incoming spikes and triggers firing accordingly. Hodgkin and Huxley developed the

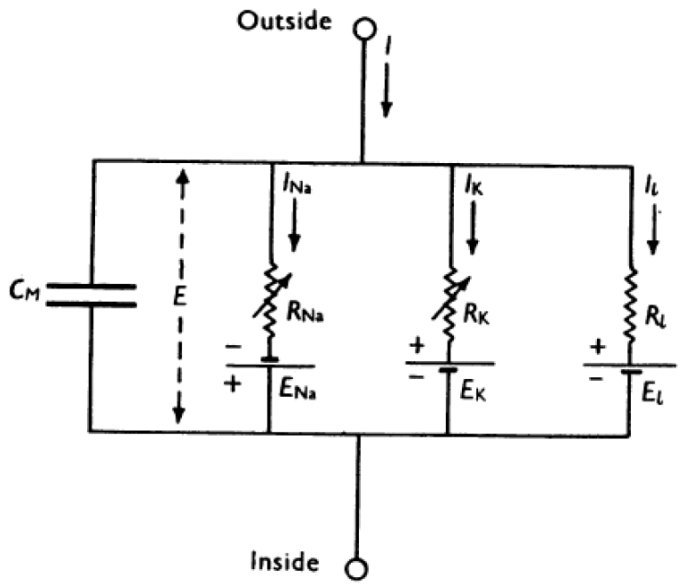

Figure 2.2: Hodgkin and Huxley Spiking Neuron Model, Adapted from [6]

first spiking neuron model [6] in 1952. It modeled the electrophysical of the neural membrane through conductance of ion channels. Hodgkin and Huxley description of the electrical behavior of neuron membrane potential is shown in figure 2.2 and it follows equation (2.1).

$$I = C_M \frac{dV}{dt} + I_i, \tag{2.1}$$

Where I is the total membrane current density, I_i is the ionic current density, V is the displacement of the membrane potential from its resting value, C_M is the membrane capacity per unit area, and t is time. Though this model laid the framework for the development of SNN neuron models, many biological relevant behaviors of the neuron could not be mapped using this model. The leaky integrate-and-fire model introduced the leak term into the models making them more biologically realistic. A simplistic digital implementation of LIF neuron [7] is summarized in equations (2.2)(2.3)(2.4).

$$V_j(t) = V_j(t-1) + \sum_{i=0}^{N-1} x_i(t) s_i \tag{2.2}$$

$$V_j(t) = V_j(t) - \lambda_j \tag{2.3}$$

$$\text{if } v_j(t) > \alpha_j, V_j(t) = R_j, \text{spike} \tag{2.4}$$

For any j^{th} neuron in the tth timestep, the membrane potential $V_j(t)$ is the sum of the membrane potential in the previous timestep $V_j(t-1)$ and the synaptic input. For each of the N synapses, the synaptic input is the sum of the spike input to the synapse $x_i(t)$ at the current timestep multiplied by the signed synaptic weight s_i. λ_j represents the linear leak factor which is subtracted from $V_j(t)$ every timestep. Then $V_j(t)$ is compared with threshold α_j. If the membrane potential is greater than or equal to the threshold voltage, the neuron fires a spike and resets its membrane potential. Figure 2.3 depicts the graphical model illustrating the functionality of a LIF neuron [7]. The Izhikevich model [25] combined the biological plausibility of HodgkinHuxley-type dynamics and the computational efficiency of integrate-and-fire neurons. In this section, we examine two variants of discrete-time leaky integrate-fire (LIF) spiking neuron models that are suitable for BP-based training of deep SNNs.

2.1.1 Spiking Neuron with Discontinuous Integration (SNN-DC)

In ANNs, the activation output of a neuron k in layer L is:

$$a_k^l = y(\sum_i^m (a_i^{L-1} w_{i,k}^{L-1}) + b_k^L) \tag{2.5}$$

where $y(x)$ is the activation function, $w_{i,k}^{L-1}$ the synapse weight connecting neuron i to neuron k, and b_k^L is the bias. One popular activation function is rectified linear unit (ReLU) as illustrated in Figure 2.4. Compared to these artificial neurons, spiking neurons have two distinct properties: (1) the neuron value (membrane potential) is integrated over time and (2) each neuron outputs a binary spike. The SNN algorithm incorporates these by introducing a discrete time step variable t and by choosing a

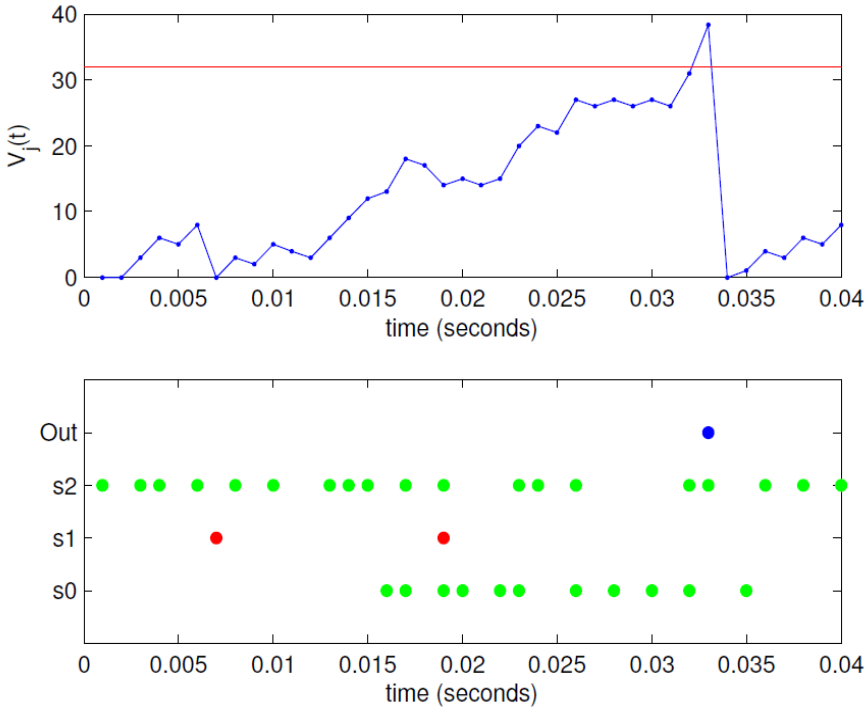

Figure 2.3: Top: Variation of LIF Neuron Membrane Potential $V_j(t)$ with Time t Bottom: Spike Raster Plot with Excitatory Inputs (green), Inhibitory Inputs (red) and Output Spikes (blue), , Adapted from [7]

special binary activation function.

$$v_k^L(t) = \sum_i^m (a_i^{L-1}(t) w_{i,k}^{L-1}) + b_k^L), \tag{2.6}$$

$$a_k^l(t) = y_b(v_k^L(t)) \tag{2.7}$$

where $v_k^L(t)$ is the membrane potential and $y_b(x)$ is a binary activation function depicted in Figure 2.4 $y_b(x) = 1$ if $x > \theta$, otherwise $y_b(x) = 0$, where θ is firing threshold. This spiking neuron model performs synaptic integration in a discontinuous manner and is denoted as SNN-DC. In other words, $v_k^L(t)$ is reset to zero at the beginning of every time step t, before the neuron integrates the presynaptic injections and the bias term. If the membrane potential after integration at time step

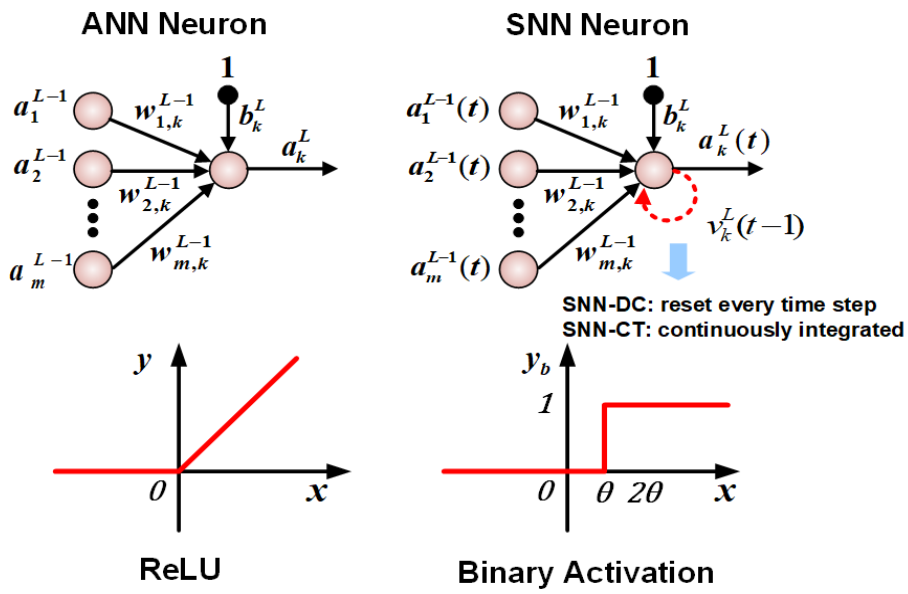

Figure 2.4: Comparison of Neuron Models for ANN with ReLU Activation and Discrete-time SNN with Binary Activation (SNN-DC and SNN-CT)

t exceeds threshold θ, the neuron fires ($a_k^L(t) = 1$); otherwise, the neuron remains silent ($a_k^L(t) = 0$). Since the neuron membrane potential is discontinuous between sequential time steps, time is not incorporated into SNN training. Thus, the SNN-DC model can be reduced to a form similar to equation (2.5) except that the activation is a unique binary activation function $y_b(x)$. SNNs trained with the SNN-DC model is suitable for rate coding spike inputs such as Poisson spikes, which assume no temporal correlation between adjacent time steps. Compared to the SNNs converted from ANNs in [21, 22], which are also designed for rate-coding spike inputs, SNNs in this algorithm. By using the SNN-DC model, good accuracy can be achieved with much fewer time steps (even with one-time step) because the proposed SNNs are directly trained over many single-time-step training samples. Figure 2.5 illustrates the variation of classification accuracies with time steps for various algorithms. We can observe that SNN-DC can achieve high accuracy even with just one-time step,

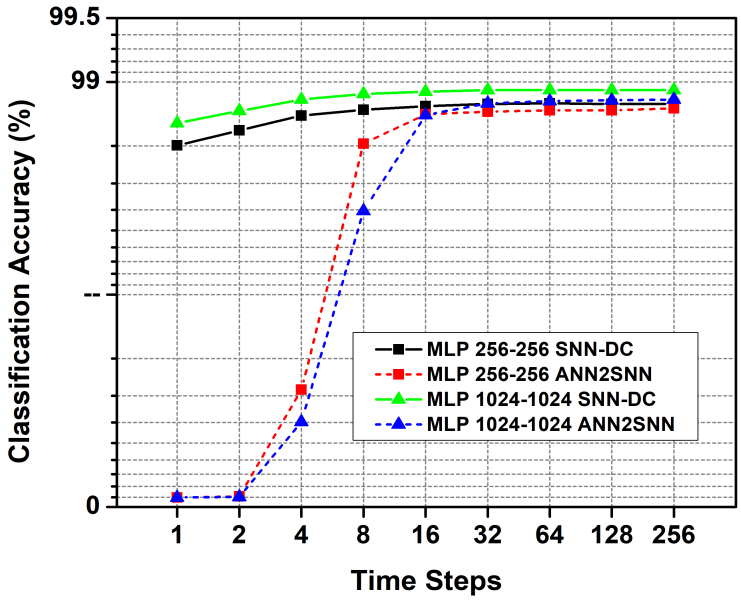

Figure 2.5: MNIST Accuracies of Various SNN Designs for Different Timesteps

whereas the SNNs converted from ANN (ANN2SNN) requires 4-16 more time steps to achieve the same accuracy.

2.1.2 Spiking Neuron with Continuous Integration (SNN-CT)

For spike input encodings other than rate-coding, the SNN-DC model may not be sufficient to capture the temporal correlation between time steps. By including membrane potential integration across multiple time steps, the SNN-DC model can be extended to a spiking neuron model with continuous integration, denoted as SNN-CT:

$$v_k^L(t^-) = \sum_i^m (a_i^{L-1}(t) w_{i,k}^{L-1}) + b_k^L + v_k^L(t-1), \tag{2.8}$$

$$a_k^l(t) = y_b(v_k^L(t^-)), \tag{2.9}$$

$$v_k^L(t) = v_k^L(t^-) - \theta . a_k^l(t), \tag{2.10}$$

where $v_k^L(t^-)$ and $v_k^L(t)$ are the membrane potentials of neuron k at time step t before and after the neuron firing check. The initial membrane potential is set to zero ($v_k^L(-1) = 0$). When $v_k^L(t^-)$ exceeds the threshold θ, the neuron fires and the membrane potential is decremented by θ as shown in equation(2.10) [22]. Compared to conventional frame-based ANNs, SNN-CT generates outputs with one more dimension: time. The N-MNIST dataset was used for training and testing a discrete-time MLP SNN with the SNN-CT model. The N-MNIST dataset was augmented by adding upward and downward movement for the static images. An MLP SNN, consisting of two 256-neuron hidden layers of 256 neurons and a 12-neuron output layer, was trained for this augmented N-MNIST benchmark. The MLP SNN was trained for dual tasks with ten output neurons for digit classification and two output neurons for upward and downward motion recognition. Figure 2.6 shows N-MNIST accuracies of digit classification and motion recognition of the proposed MLP SNN-CT as a function of time steps. Digit classification and motion recognition accuracies are 96.33% and 99.83%, respectively. For both tasks, accuracies increase as the number of time steps increase. Fewer input spikes in the last few time steps cause the motion recognition accuracy to slightly decrease.

2.2 Deep Neural Networks with Coarse Grain Sparsity

2.2.1 Overview

As the majority of computations in DNN are multiply-and accumulate (MAC) operations, constraining the weights and activations to low precision during training can result in a significant reduction in energy with appropriate hardware design for classification. BinaryConnect [10] employs the quantized value of the real-valued weights for forward and backward phases of backpropagation. With the weights

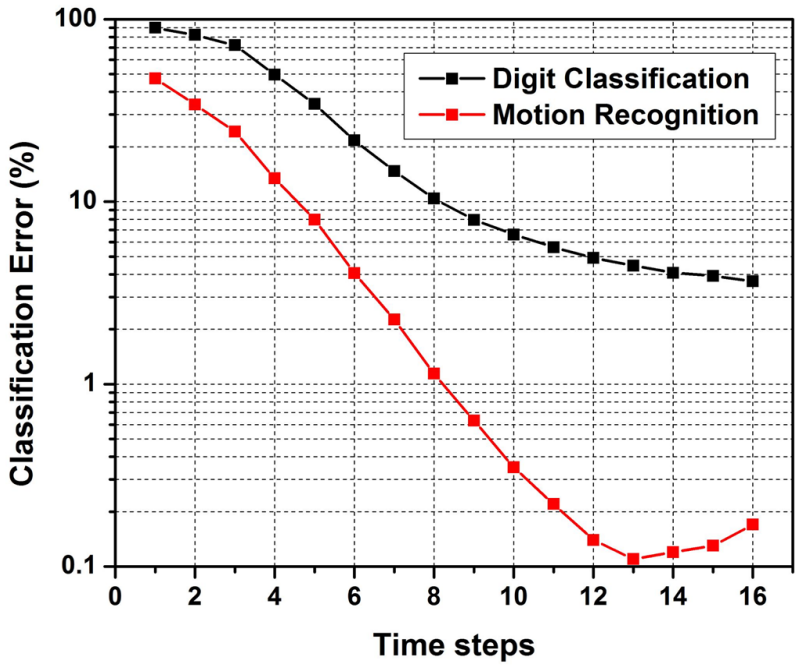

Figure 2.6: N-MNIST Accuracies of Digit Classification and Motion Recognition

constrained to just 1-bit, the MAC operations can be replaced with simple additions and subtractions. BNN [11] quantized both weights and activations to +1 or -1, where MAC operations become bit-wise XNOR and accumulate operations. Authors in [10, 11] argue that the quantization noise acts as a regularizer and hence can give good test accuracy even with 1-bit quantization. Compared to BNN, XNOR-Net [12] showed significant improvement in ImageNet classification accuracy with binary weights and activations.

Structured Sparsity learning (SSL) [14] applies group Lasso regularization [26] to the weights belonging to a DNN structure. This prunes the weights corresponding to the unnecessary structures in the DNN model. SSL generates compact DNN structures which can be efficiently implemented in hardware with less memory utilization, thereby saving silicon area and power. Scalpel [15] applies DNN sparsity depending on the level of data-parallelism of the target hardware. Matrix multiplications on a

sparse matrix need extra computations to decode the sparse format of the matrix. For low-parallelism hardware, SIMD-aware weight pruning maintains weights in aligned fixed-size groups to fully utilize the SIMD units. For high-parallelism hardware, node pruning is applied so that the dense matrix is retained, but redundant nodes are removed. For moderate-parallelism hardware, a combination of SIMD-aware weight and node pruning is performed.

2.2.2 Coarse-Grain Sparsity

Coarse-Grain sparsity (CGS) [16] is a technique to generate structured sparsity by randomly dropping blocks of weights within the DNN weights matrix throughout training. The overall sparsity depends on the CGS block size and the CGS compression ratio (CGS ratio). Since sparsity is formed on a block-by-block basis, the index overhead is minimized allowing the final trained weights to be efficiently mapped onto SRAM arrays.

2.2.3 Combining Low Precision and Structured Sparisty

Training algorithm was based on BNN with additional structured sparsity constraints of Coarse-Grain Sparsity (CGS). Before training, blocks of weights were randomly dropped off according to the CGS block size and CGS compression ratio. These blocks remain zero during training and inference. The training algorithm for non-sparse blocks of weights are similar to that of BNN training using backpropagation. For fully-connected layers, the weight matrix is divided into square blocks. For $x * x$ block size, each block contains x^2 weights. Once the weights are segregated into blocks, a large number of blocks are randomly dropped off with probability equal to the CGS ratio. These blocks remain zero during training and inference and hence do not contribute to the physical memory. Figure 2.7 shows an example weight matrix

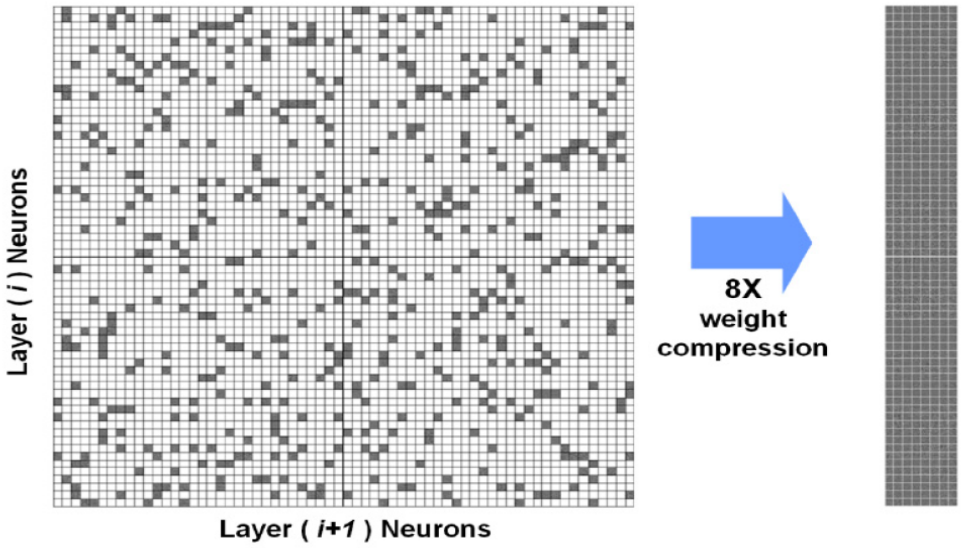

Figure 2.7: Illustration of CGS with Weight Matrix of 1024x1024 having 87.5% of Weights Dropped with Block-Wise Sparsity

for a fully-connected layer of size $1024 * 1024$, where each square represents a block of weights of size $16 * 16$. Grey squares represent blocks where eligible connections are present and white squares represent blocks with the absence of connections. Figure 2.7 (right) illustrates the blocks with active connections, compressed along the row, after applying 8X CGS ratio [16, 27]. The sparse weight matrix/tensor, generated after applying CGS is trained using backpropagation by quantizing weights and activations.

A Multi-Layer Perceptron (MLP) architecture with two hidden layers was used for training MNIST. Model selection for best architecture was performed based on accuracy study on different architectures. Test accuracy for different architecture settings have been compared in Figure 2.8 for different models. Examining of test accuracies of architectures with 1, 2 and 3 hidden layers with a different number of neurons per hidden layer (128, 256, 512, 1024 neurons), we can observe that the

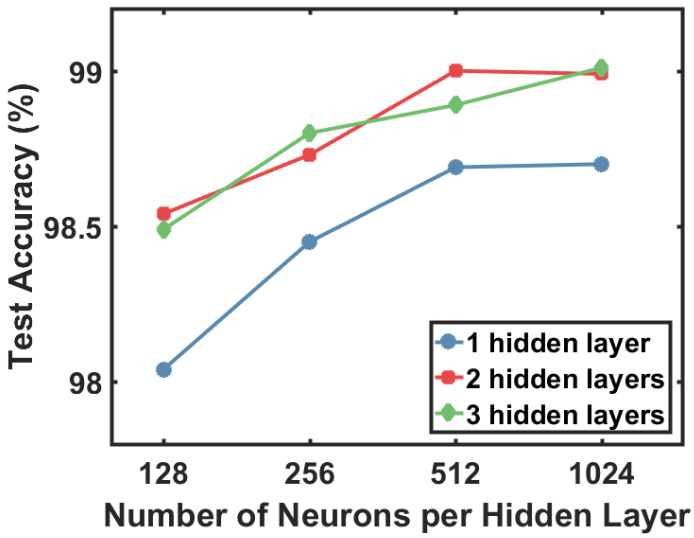

Figure 2.8: Design Point Selection of MLP for MNIST Dataset

2-hidden layer architecture with 512 neurons provides uncompromised test accuracy with fewer neurons/layers. Hence, 2-hidden layer architecture with 512 neurons was used for the MLP investigation of MNIST dataset.

Chapter 3

IMAGE CLASSIFICATION HARDWARE DESIGN

In this chapter, we discuss the hardware implementations of SNN and DNN algorithms for classification of MNIST/N-MNIST dataset. The SNN algorithm had two different modes namely SNN-DC and SNN-CT which differed in synaptic integration. Both SNNs have two 256-neuron hidden layers. The SNN-DC (SNN-CT) for MNIST (N-MNIST) has 784 (1,156) input neurons and 10 (12) output neurons. All the weights and biases were stored in on-chip SRAM arrays, and membrane potentials of the neurons were saved in registers. A configurable hardware architecture was developed to support both modes of operation, thereby supporting the classification tasks of both MNIST and NMNIST dataset. Additionally, hardware reuse was also introduced to the architecture to make it more programmable. CGS compression technique was employed in DNN algorithm for MNIST digit classification. Compressed weights and corresponding index information were saved in on-chip SRAM arrays. A CGS decompressor unit was designed to decompress the weights which were then passed to MAC units. To explore various sparsity structures with low precision neural networks, different memory modules were used and demanded changes/optimizations were done in RTL to obtain different custom hardware implementations for all chosen design points. All of this DNN hardware supported two hidden layers with 512 neurons and 10 output layer neurons.

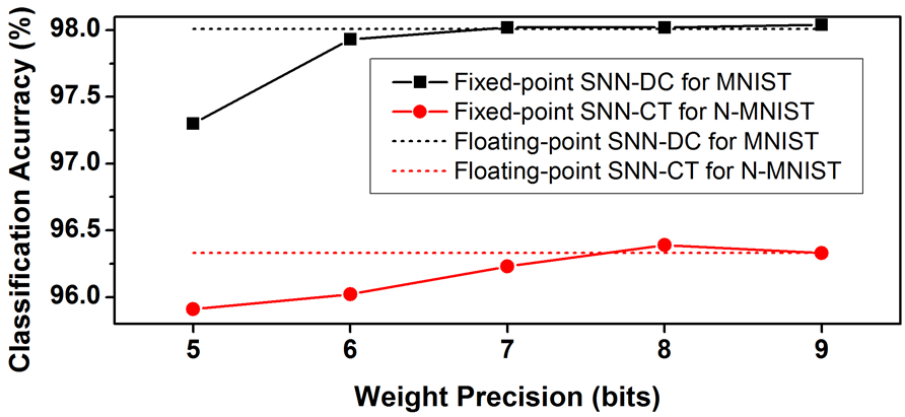

Figure 3.1: Classification Accuracies for Different Precision of Weights.

3.1 Neuromorphic Hardware for Discrete-time MLP SNN's

3.1.1 Weight Precision Study

Due to power/area limitations, high precision synaptic weights cannot be employed in the hardware implementation of neural networks. Hence, we need to convert high precision weights into the fixed-point format. Due to this pre-processing of weights, we endure little loss in accuracy. Figure 3.1 shows the variation of classification accuracy with weight precision. We selected 7-b weight precision for both SNN-DC and SNN-CT resulting in negligible accuracy loss compared to floating-point precision.

3.1.2 Hardware Architecture Supporting SNN-DC and SNN-CT

Figure 3.2 shows the overall hardware architecture where synchronous clocking is used with extensive clock gating. This feed-forward pipelined design consists of a spike scheduler to recognize the active spike position, two hidden layers of 256 accumulation units and one output layer of 12 accumulation units acting as artificial neurons and control logic to ensure the precise functionality of pipelining. We employ parallel

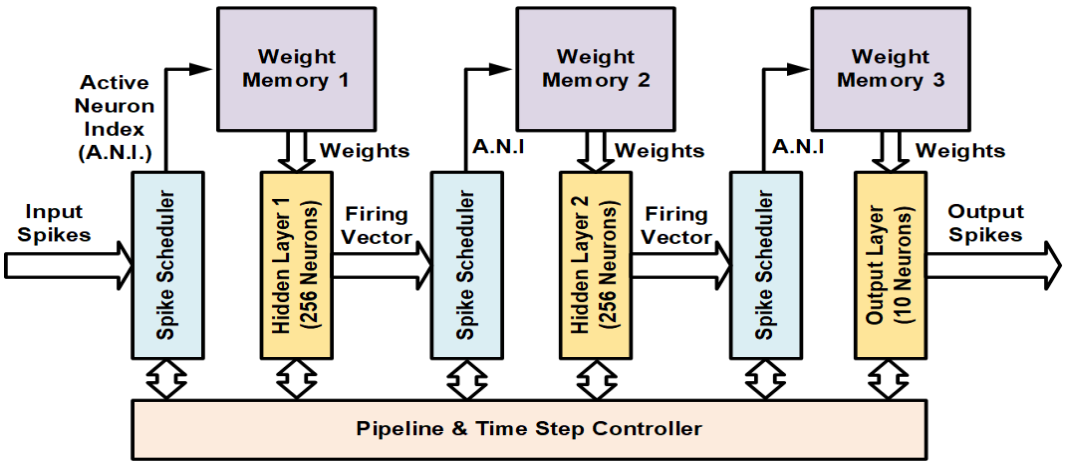

Figure 3.2: Hardware Architecture for the Proposed MLP SNNs

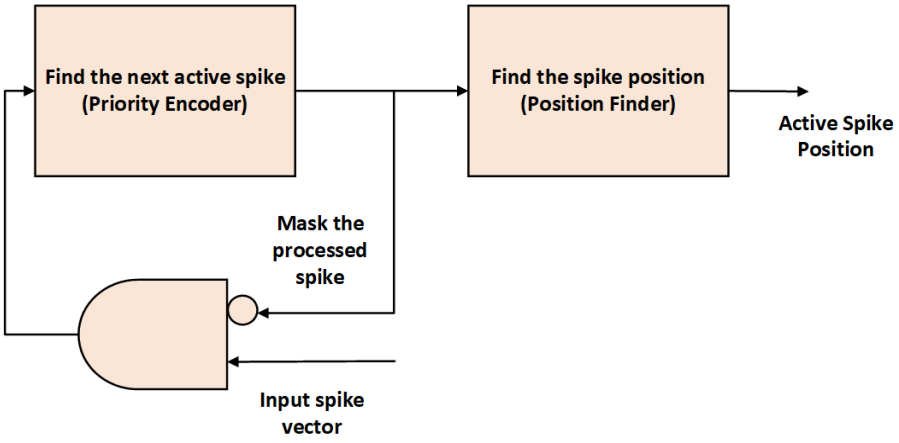

Figure 3.3: Block Diagram of Spike Scheduler Unit

output neurons for the hidden/output layers while the input spikes of the neurons are processed serially in each clock cycle. SNN-CT (SNN-DC) having network structure of $1156 - 256 - 256 - 12$ ($784 - 256 - 256 - 10$) requires $346kB$ ($289kB$) of memory for 7 bit weights. Design of each block is discussed in next subsections.

Spike scheduler for event-driven operation: Spike scheduler was used to utilize the spike sparsity. Figure 3.3 shows the block diagram of spike scheduler unit consisting of a priority encoder and a position finder unit. Output spikes generated

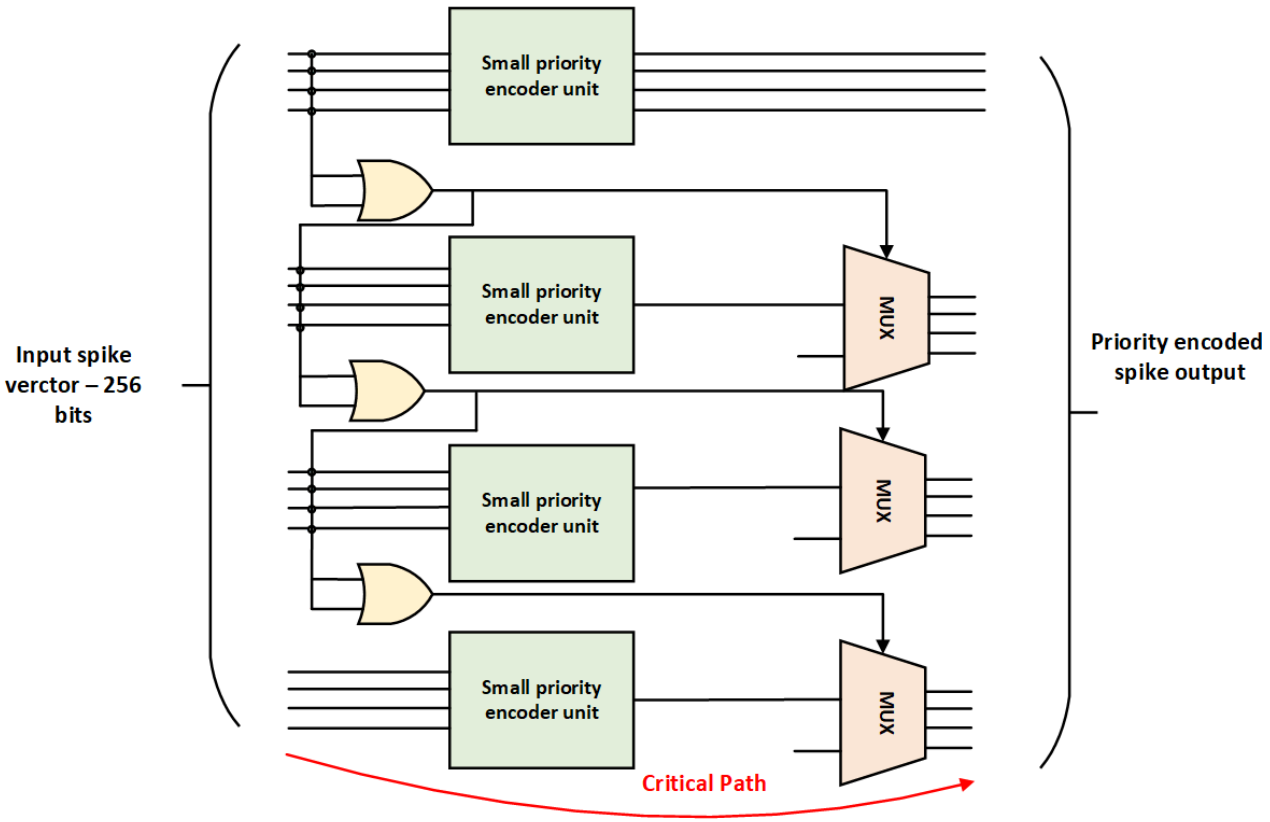

Figure 3.4: Block Diagram of Prioriry Encoder

at every timestep are given to the priority encoder to obtain a unique active spike. In the next cycle, the previous unique spike is masked using an 'AND' gate thereby generating a new spike input to the priority encoder. The position finder unit obtains the index values of the sequentially generated unique active spikes. Hence, the spike scheduler unit featuring 256-/1156-input priority encoders sequentially generates active presynaptic neuron indices from binary input spike vectors. An 1156 bit priority encoder hinders the high-frequency operation because of its vast critical path. To break this critical path without increasing the latency, architecture shown in figure 3.4 was employed for priority encoders. The maximum delay was either the delay of small priority encoder or the or gate chain. The generated neuron index is transferred to the weight memory to fetch the weights for all parallel postsynaptic

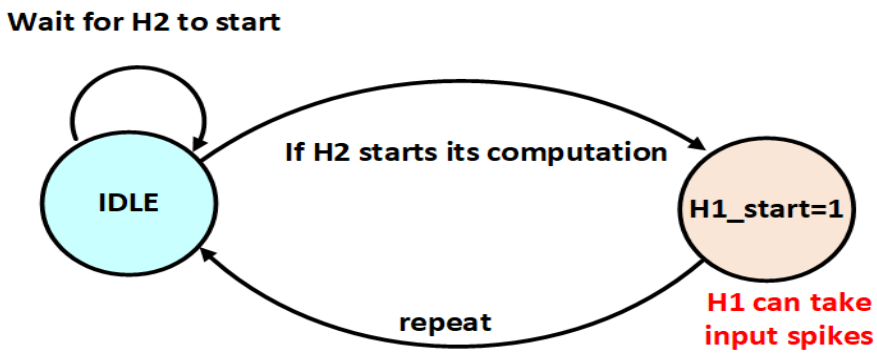

Figure 3.5: State Diagram used to Generate Handshaking Signals for First Hidden Layer

neurons. These weights are integrated into the neurons membrane potentials. After all input spikes are processed, the postsynaptic neurons fire if their membrane potentials exceed the threshold. Since only a small fraction (only 4.8% in the SNN-CT for N-MNIST dataset) of the presynaptic neurons are active at each time step, the spike scheduler enables event-driven computation for only active neuron spikes, substantially reducing the latency and energy.

Pipeline architecture and handshaking : All SNN layers are pipelined to enhance throughput. Since the number of active spikes varies with layer and time, handshake signals are exchanged between adjacent layers to reduce the overall latency. Figure 3.5 shows the state diagram used to generate handshaking signals for the first hidden layer. The first hidden layer can start its computation only when the next stage takes the computed output else the output will be overwritten resulting in the improper operation. Two conditions have to be verified to start computation of second hidden layer:

(1) Outputs of the first hidden layer should be ready so that second hidden layer can begin processing it.

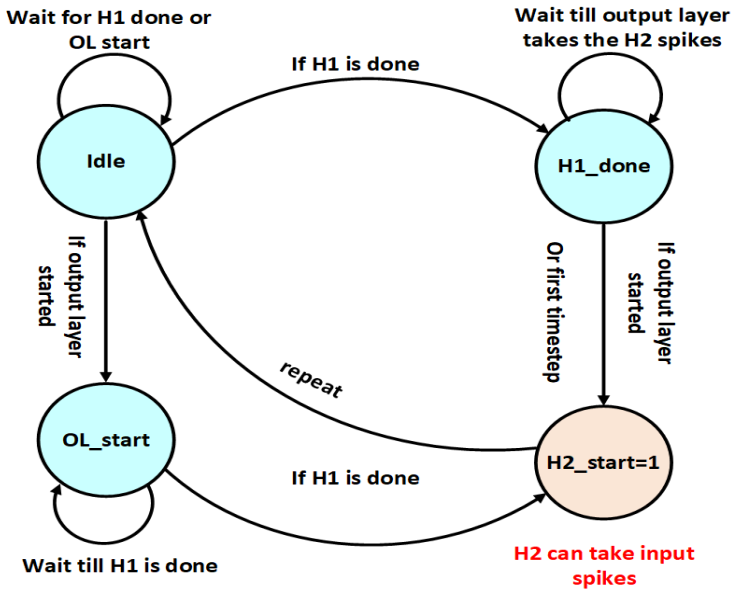

Figure 3.6: State Diagram used to Generate Handshaking Signals for Second Hidden Layer

(2) The output computed by the second hidden layer should be taken by output layer ensuring that it will not be overwritten when new input is processed.

Since the number of active spikes in each layer varies with every timestep, any of the above two conditions can happen first. Figure 3.6 shows the state diagram which assures proper stalling of the second hidden layer. Only when both conditions are satisfied the 'H2_start' signal is generated. Figure 3.7 shows the state diagram used to produce handshaking signals for the output layer. The state diagram of output is similar to second hidden layer, but the conditions are varied. The output layer can start its computation only when both second hidden layer and output layer has completed their previous calculations.

Configurable neuron units : Binary spikes in spiking neural networks benefit in getting rid of multipliers inside the neuron architecture. The figure 3.8 shows the con-

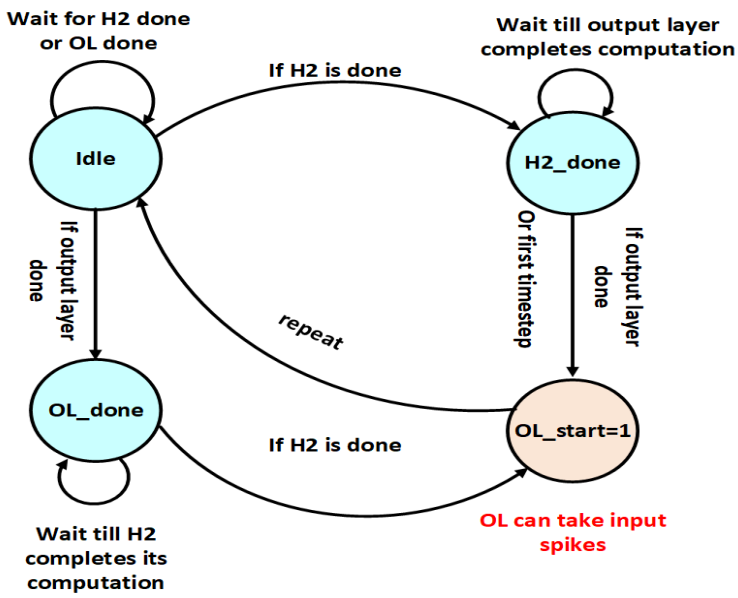

Figure 3.7: State Diagram used to Generate Handshaking Signals for Output Layer

figurable neuron architecture designed to support SNN-DC and SNN-CT algorithms. After receiving the active neuron index from the spike scheduler, SRAM modules provides synaptic weights to all the postsynaptic neurons. These 7-bit weights are accumulated and passed to flooring/ceiling unit to guarantee that the accumulated values will not suffer from overflow/underflow errors. The '*done*' signal is created by spike scheduler unit when there are no active spikes left. After the completion of computation, the accumulated values are compared with the threshold (θ), and a spike is generated. The '$refresh$' signal is used to configure the neuron to SNN-DC/SNN-CT mode. If refresh signal is asserted then neuron supports SNN-DC by resetting the membrane potential after every timestep else membrane potential of the previous time step is retained. In SNN-CT mode, after the spike generation, the membrane potential of the neuron is subtracted from the threshold and passed to the accumulator. Two logical 'AND' gates are used to generated right control signals for the MUX's ensuring correct operation.

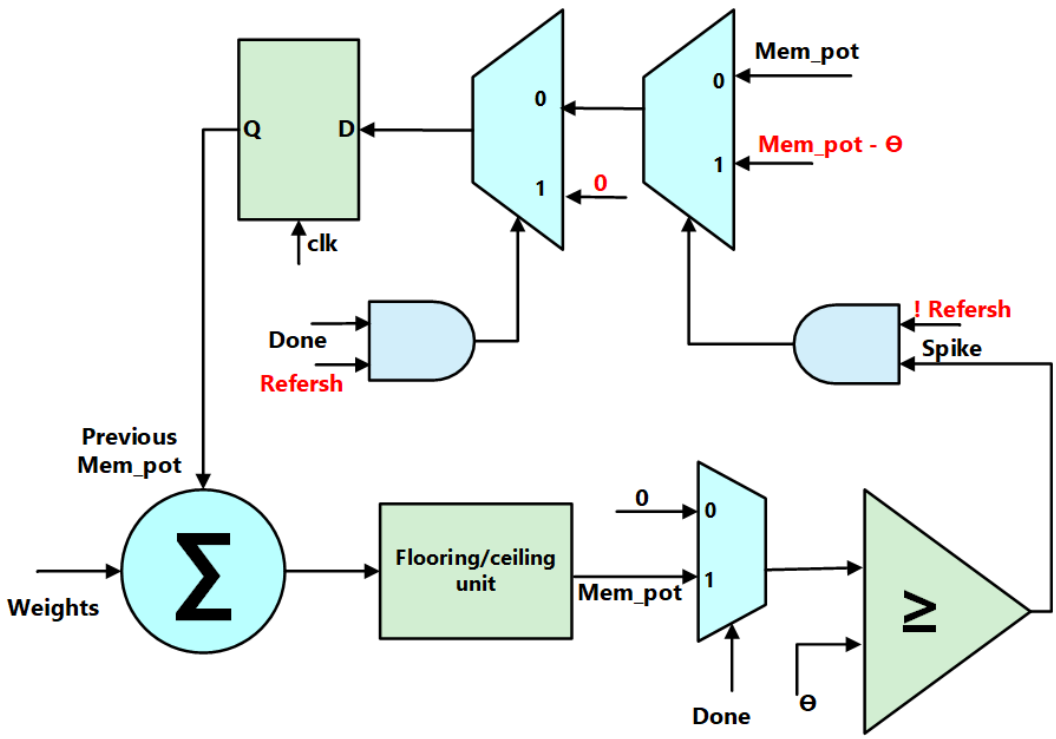

Figure 3.8: A Configurable Neuron Supporting SNN-DC and SNN-CT

3.1.3 Programmable Hardware Design for MLP SNN's

The SNN-DC/SNN-CT hardware architecture was augmented practicing hardware reuse to support algorithms with a different number of hidden layers. Figure 3.9 shows the flowchart of the algorithm used where N represents the number of hidden layers, i represents current layer and A, B, C represents 256-256-12 accumulator layers present in hardware respectively. After each hardware re-use, new weights were uploaded to every memory module and the intermediate spike outputs were saved SRAM blocks. Based on i and N(even/odd), layers B and C were bypassed using control signals. Figure 3.10 shows an example of hardware reuse for five hidden layers. In this example, we can see that example layer B is bypassed in the last iteration. During each iteration, pipelining was done between active layers, thereby reducing the latency.

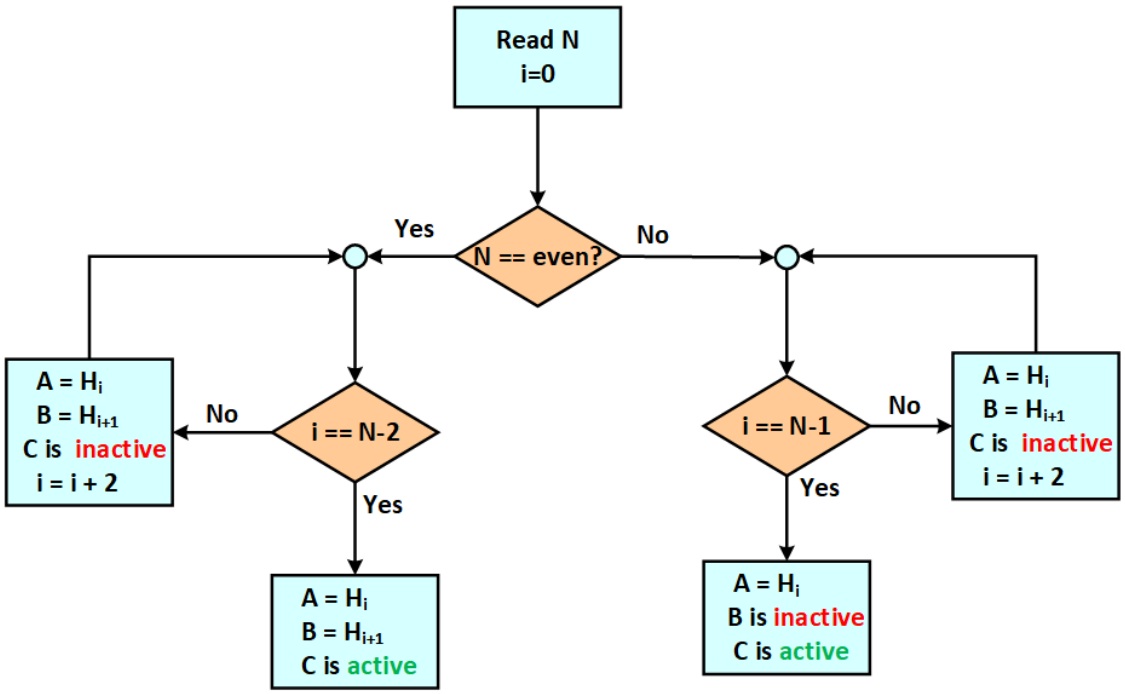

Figure 3.9: Flowchart Depicting the Algorithm used for ProgrammableHardware

3.2 Hardware Design of Deep Neural Networks with Coarse Grain Sparsity

Figure 3.11 shows the overall DNN acceleration system. The hardware supports two hidden layers with 512 neurons and 10 output layer neurons. On-chip SRAM arrays stored the weight and bias values. Each layer constitutes a set of MAC units followed by batch-norm and activation layer. In each hidden layer, input neurons are processed serially whereas the accumulation of weighted sum is done in parallel. Since the significant amount of output activations were zero's, zero-skipping block was deployed to improve latency similar to SNN hardware. By finding the active input neuron index, the zero skipping block skips the computation cycles for zero input activations. Active input neuron index is sent to on-chip compressed weight memory to fetch weights for all parallel MAC units. By exploiting the sparsity of input activations, latency can be reduced by 4.2X, on average, for the DNN using

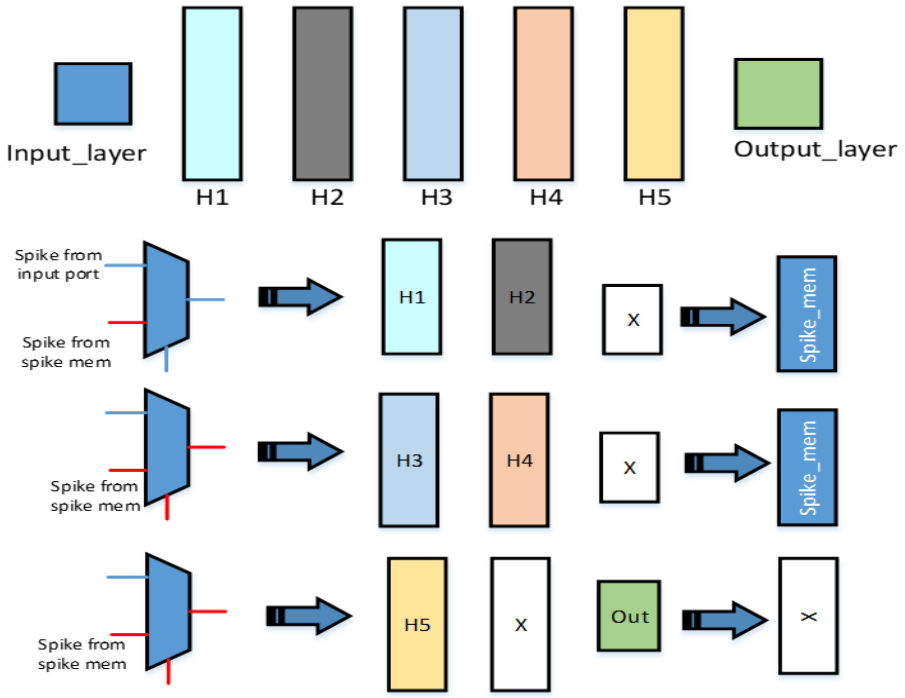

Figure 3.10: An Example of Hardware Reuse when N=5

input images from the MNIST dataset. All the layers in feedforward architecture were pipelined to increase the throughput. The number of cycles consumed by each layer depends on the number of non-zero input activation's. Since the number of non-zero input values varies with input data, handshake signals are exchanged between adjacent layers to ensure proper execution of pipeline.

3.2.1 Coarse Grain Sparsity in Hardware

Weights dominate fully connected DNN memory. To reduce the memory utilization CGS-based compression is used to store the weights. To achieve structured compression, the neural network is trained by dividing DNN weights into blocks and randomly dropping them with a probability. Only the non-dropped weights with their corresponding index values are stored in on-chip SRAM arrays. Weight vectors

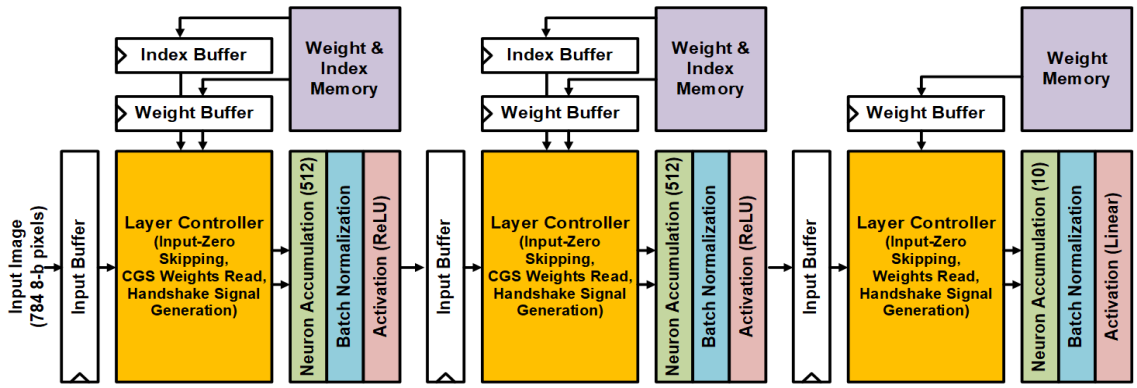

Figure 3.11: Hardware Architecture used for Fully-Connected DNNs.

are decompressed using decoders by providing the index bits as select signals. Block diagram of a CGS decompressor unit for a block size of 128, weight precision of 8 bit and 50% CGS is shown in fig 3.12. Considering 512 neurons in hidden layers and weight precision of 8-bits, each row in the weight memory will accommodate 512 eight bit weights. These weights are divided into 4 blocks, each of block size 128 (i.e. 128 x 8 bits). Given a CGS of 50%, 2 out of these 4 blocks are randomly dropped with a probability. The index values of two chosen blocks are provided to the index decoder unit. Decoder helps in activating the path only for the selected weight block. The bitwise AND gate aids in passing the selected weight block to the activated path. As there is no possibility of both decoders selecting same weight block, a bitwise OR gate is used so that each decompressed weight block can get is weights from any of the selected two weight blocks. As every index values require a decoder unit, the number of decoders increases as we decrease the block size. All the non selected weight blocks will be zeros in the decompressed output. The final decompressed weight row of (512 x 8 bits) is given to the output neurons. The results of DNN hardware shown in next sections are obtained for a block size of 16.

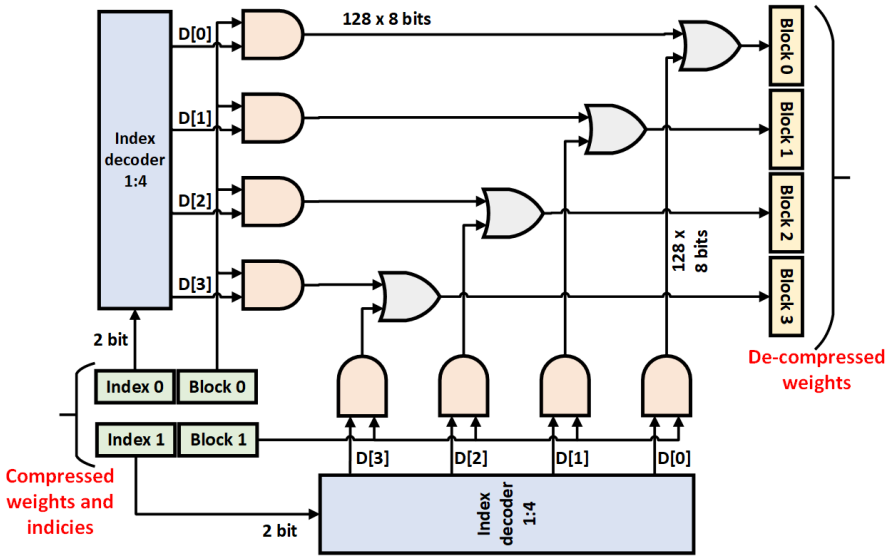

Figure 3.12: Block Diagram of CGS Decompressor (Block size:128, Weight Precision:8 bits, 50% CGS)

3.2.2 Implementation of Batch Norm and Low Precision Neuron Models

Conventional batch normalization follows equation (3.1),(3.2),(3.3) where we need to perform three additions, one multiplication, and one division operation.

$$x^{'} = \sum_{i}^{n} w_i.a_i \tag{3.1}$$

$$x = x^{'} + b \tag{3.2}$$

$$y = \frac{x - \mu}{\sigma}.\gamma + \beta \tag{3.3}$$

Where a_i is input activation, w_i weight, $b$is bias, $x^{'}$s the weighted sum, y is the output value before activation, and μ andσ are the mean and standard deviation of the weighted sums in a batch, respectively. γ and β are batchnormalization scaling and shifting parameters. These operations are optimized by using new constants, namely, addition parameter $\beta^{'}$ and multiplication parameter $\gamma^{'}$ then equation (3.1),(3.2),(3.3) are reduced to equation (3.4), where only one multiplication and one addition oper-

ation is needed. This results in significant reduction in power and area.

$$y = x^{'}.\gamma^{'} + \beta^{'} \tag{3.4}$$

where $\beta^{'} = (\beta + \frac{b-\mu}{\sigma}).\gamma$ and $\gamma^{'} = \frac{\gamma}{\sigma}$.

For low precision weights (¡3 bits), MAC multipliers are replaced by shifters following the scheme in LightNN [28]. Each possible weight value is encoded with a signed shift value. For example, 2-bit weights are encoded as 00: -1/4, 01: -1/2, 10:1/2, 11:1/4 and multiplication of these weights with input activations was performed by shifters. High precision weights followed conventional MAC architecture.

Chapter 4

RESULTS

4.1 SNN Hardware for SNN-CT and SNN-DC

The hardware architecture supporting SNN-DC and SNN-CT was implemented in TSMC 28nm LP CMOS. Synapse weights are stored in SRAM generated from a commercial memory compiler, and digital logic is synthesized using standard cells. The SNN hardware consisted of three memory blocks of size 1156x256x7, 256x256x7, and 256x12x7 bits respectively. The 1156x256x7 bit memory block was implemented employing fourteen 1024x128 and 256x128 blocks. Similarly, 256x256 bit block was implemented using fourteen 256x128 blocks. Figure 4.1 shows the placement of these

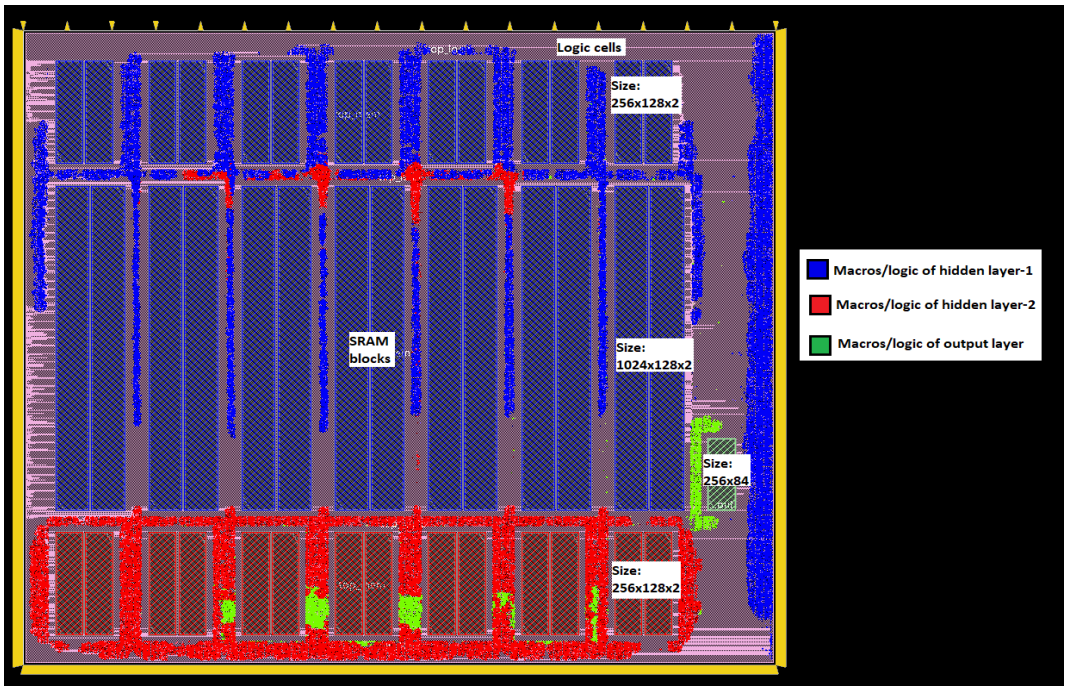

Figure 4.1: Macro Placemnet and Floorplanning of SNN Hardware

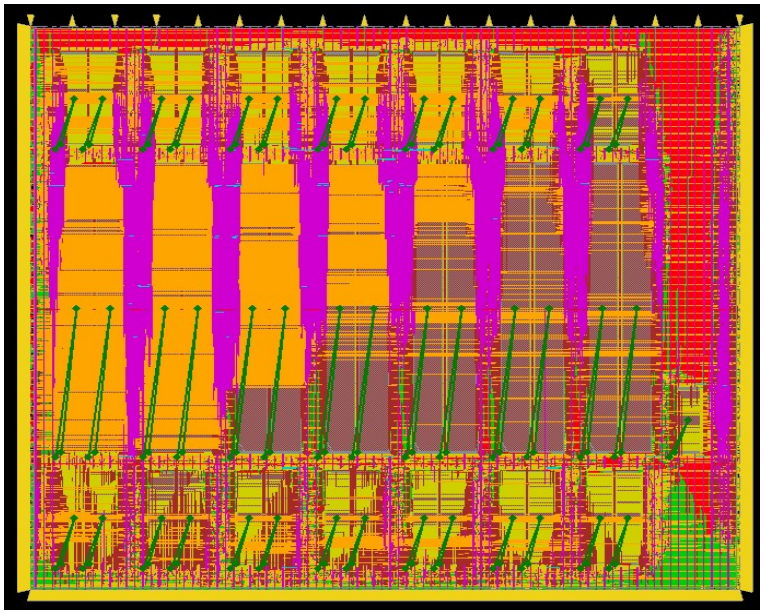

Figure 4.2: Layout View of the Neuromorphic Processor after Adding Filler Cells

43 macros (blue:1156x256x7 , red:256x256x7, green:256x12x7) and floorplan designed for SNN hardware in Cadence Innovus. Power distribution for all macros was done using power rings, and power grids (till M7) were used for logic. Placement of standard cells near the macroblocks was avoided by using hard blockages near all macroblocks. Figure 4.2 shows the layout view of the neuromorphic processor after adding filler cells. Test accuracy and latency are obtained from post-layout simulation for the entire MNIST test dataset of 10k images. The total post-layout neuromorphic processor area is 1.65 mm^2, with 0.79 mm^2 logic and 0.86 mm^2 memory. At the nominal supply voltage of $0.9V$, power consumption results are obtained from Cadence Innovus with data switching activity information from post-layout simulation. The proposed SNN hardware implementation results are summarized in table 4.1 where the capability to train/classify with a small number of time steps greatly reduced the energy down to 51.4 nJ per classification. Figure 4.3 shows a comparison to previous MNIST hardware designs [29, 30, 31] for accuracy and energy. Compared to a recent 28nm ANN design

SNN Design	No. of timesteps (N)	Freq. (MHz)	Latency ($Cycles$)	Power (mW)	Energy per classification (nJ)
SNN-DC for MNIST	1	163	119	70.4	51.4
	16	163	1780	70.8	773
SNN-CT for N-MNIST	16	163	654	73.2	294

Table 4.1: SNN Hardware Implementation Results

[29], the proposed SNN-DC reduces energy by 3X at iso-accuracy between 98% and 99%. Note that our reported energy is based on post-layout simulation while others are based on chip measurement results.

4.2 DNN Hardware Implementation Results

The hardware architecture for DNN was implemented in TSMC 40nm LP CMOS with high Vt devices. All the designs, with different weight and activation precisions, are synthesized at 100MHz with extensive clock gating. DNN weights are stored in SRAM arrays generated from a commercial memory compiler. For each design with different weight precisions, new SRAM arrays were generated such that 512 weights fit in one row. Test accuracy and latency are obtained from postsynthesis simulation for the entire MNIST test dataset of 10k images. Power numbers are obtained from Synopsys Primetime PX using data activity of fully connected DNN layerspostsynthesis netlist. Figure 4.4 shows the accuracy and post-synthesis power breakdown of the competing designs. Compressing the memory using CGS significantly reduces the power consumption by reducing the memory size and number of weight accumulations. The total logic power is dominated by accumulation of weighted sum. This exceeds

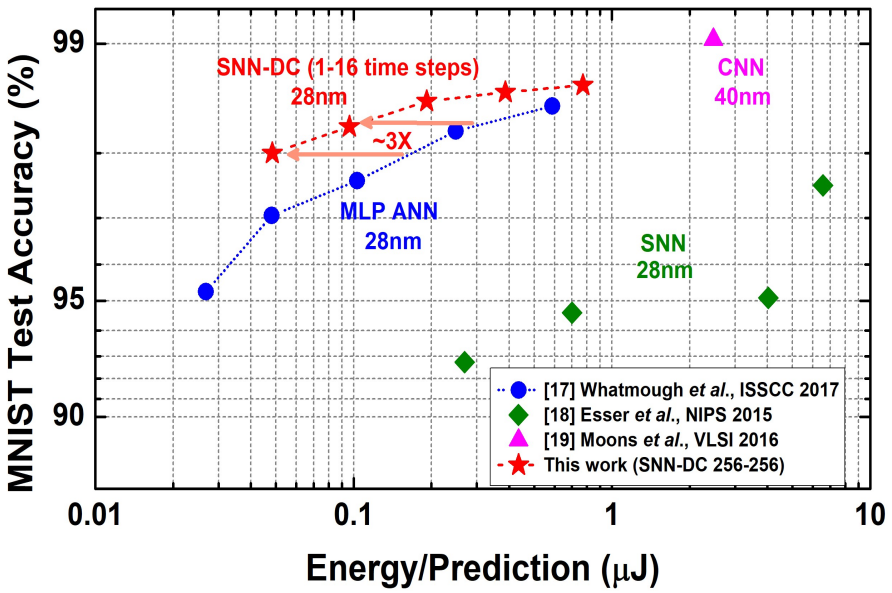

Figure 4.3: MNIST Accuracy and Energy Comparison to Hardware Design Literature

the memory power which has been reduced due to aggressive weight compression. The controller power remains almost constant for all design points. For a constant CGS ratio, reducing the activation precision from 8b to 3b showed a vital decrease in total power. Figure 4.5 shows the accuracy and post-synthesis area of memory and logic for different weight/activation precisions and CGS ratios. When using higher CGS compression ratio and lower precision, the memory area significantly decreases and the logic area starts dominating the total area. Reducing the weight precision is more effective for the overall area reduction compared to lowering the activation precision. The smallest area of $0.47mm^2$ is achieved by the BNN design (1- bit activation and 1-bit weight) with 8X CGS compression. Figure 4.6 shows the energy per image and the classification accuracy tradeoff for different activation/weight precision values and CGS compression ratios. With 8-bit activations, 3-bit weights, and 8X compression, 98.4% MNIST accuracy was achieved with 20nJ energy per classification which is

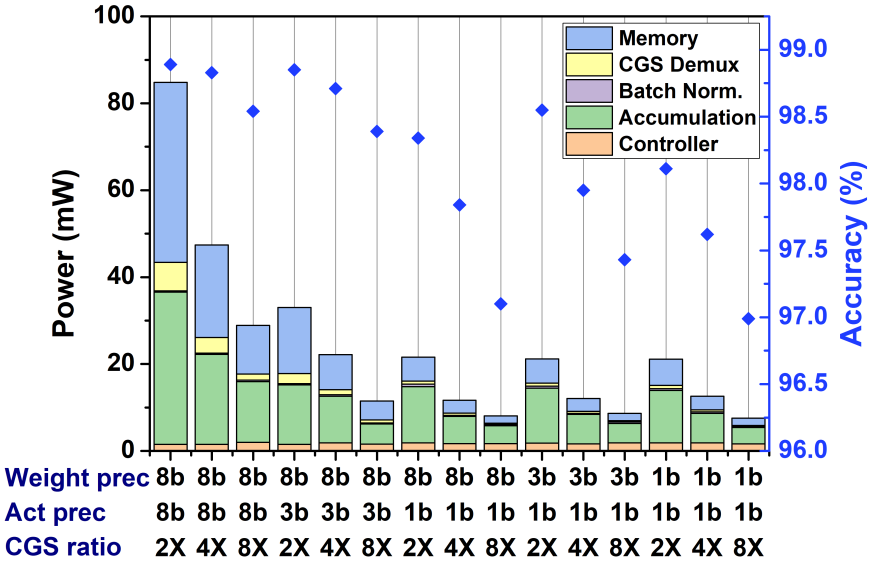

Figure 4.4: Power Breakdown for Different Combination of Weight Precision, Activation Precision and CGS Compression Ratio.

a favorable accuracy-energy trade-off compared to much lower precision DNNs with less compression (BNN achieves 13nJ energy at 97% accuracy). Compared to the uncompressed DNN with 8-bit precision, we achieve greater than 10X energy reduction with only 0.6% accuracy loss, by optimal combining of low precision and CGS compression. Automatic Place and Route was done for design points resting on the forefront of Pareto optimal curve using Cadence Innovus. Floorplan of each design varied because of different sized memory modules. Figure 4.7 shows the floor plan and macro placement of the DNN hardware with 8b weights, 3b activations and 50% CGS. The area highlighted by blue color represents the logic modules (CGS decompressor, MAC units, batch norm module) of hidden layer 1. Similarly red and green color indicates logic modules related to hidden layer 2 and output layer respectively. The SRAM modules were of sizes 1024x128x16, 512x128x16 and 512x80 supporting 8-bit weight precision and 50% CGS. Placement of these blocks was made near their

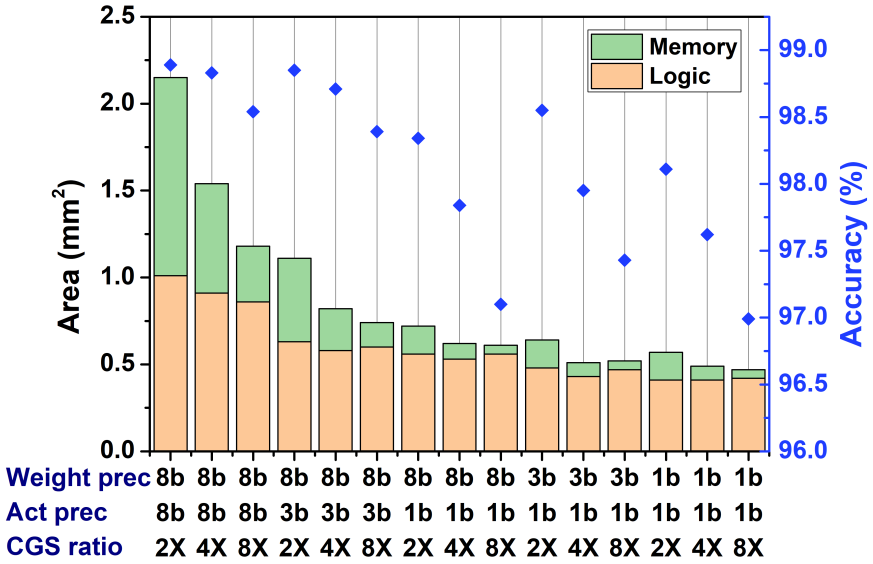

Figure 4.5: Area Breakdown for Different Combination of Weight Precision, Activation Precision and CGS Compression Ratio.

individual write data pins ensuring no routing congestion. Macros were covered with hard blockages to avoid placing violations. Figure 4.8 shows the overall layout view of DNN hardware after adding filler cells. The post-layout area, power, and energy numbers of design points present on the forefront of Pareto optimal curve are summarized in table 4.2. The post-layout energy numbers were almost 2x higher than the post-synthesis results. This increase in energy is mainly because of metal routing resulting in layout parasitics. Figure 4.9 gives the comparison of post-synthesis and post-layout energy numbers. The difference in energy numbers is not constant because of the different routing optimizations done by the tool.

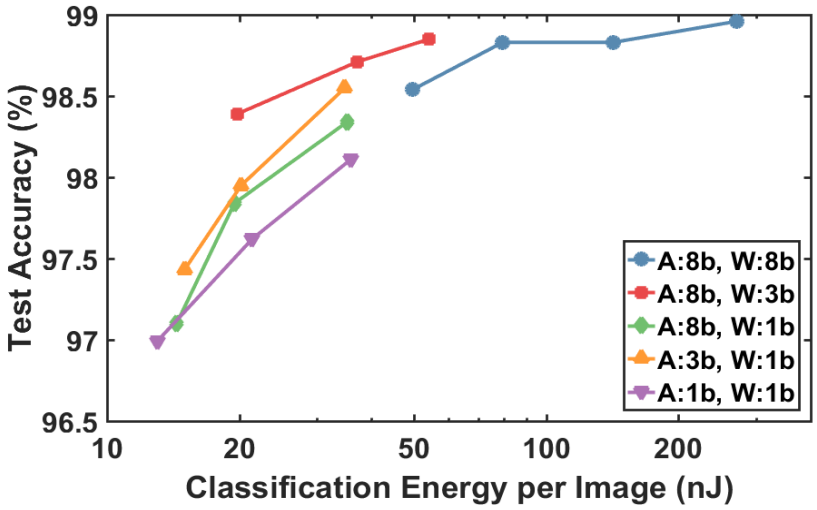

Figure 4.6: Classification Energy and Test Accuracy of MNIST MLP Designs with Different Precision and Sructured Compression.

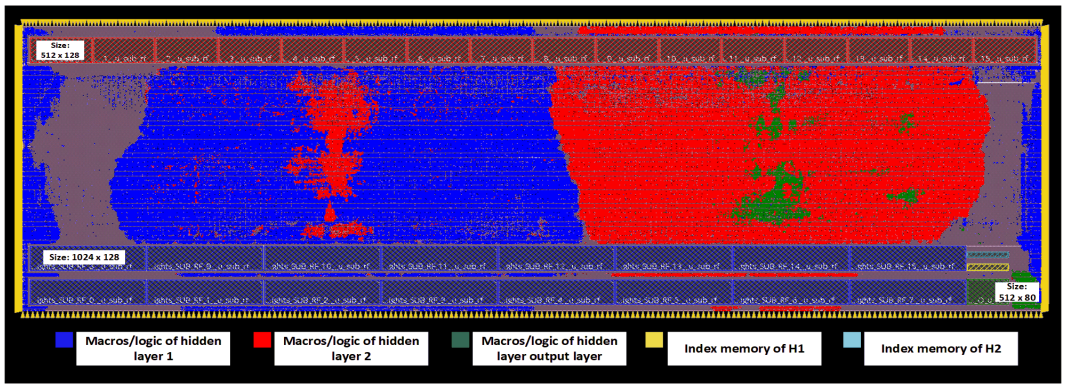

Figure 4.7: Macro Placemnet and Floorplanning of DNN Hardware (8b weights and 50% CGS)

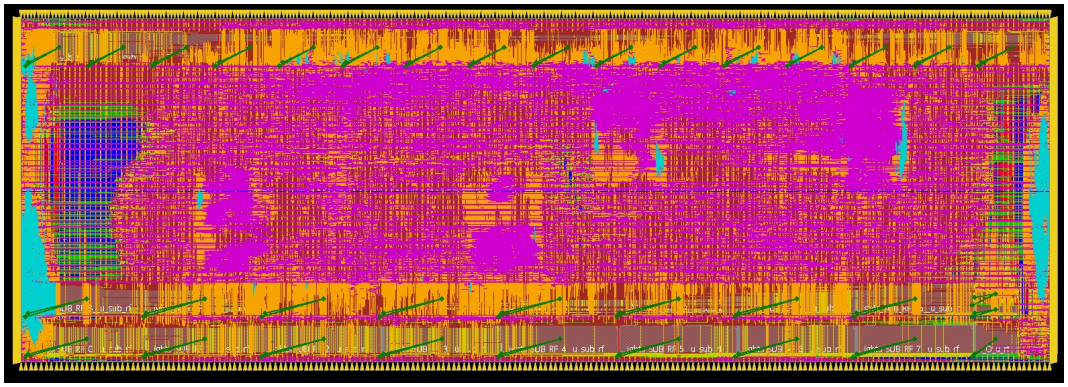

Figure 4.8: Layout View of the DNN Hardware (8b weights and 50% CGS) after Adding Filler Cells

Design Name	Power (mW)			Latency	Energy	Area
	Memory	Logic	Total	$Cycles$	nJ	mm^2
8b_50p	61.2	57.2	118.4	168	198.912	5
3b_50p	19.3	27.1	46.4	164	76.096	2.25
3b_25p	10.4	17.9	28.3	167	47.261	1.81
3b_12.5p	5.5	13.7	19.2	172	33.024	1.56
1b_1b_12.5p	2.19	9.71	11.9	178	21.182	1

Table 4.2: DNN Hardware Post-Layout Results at $100Mhz$

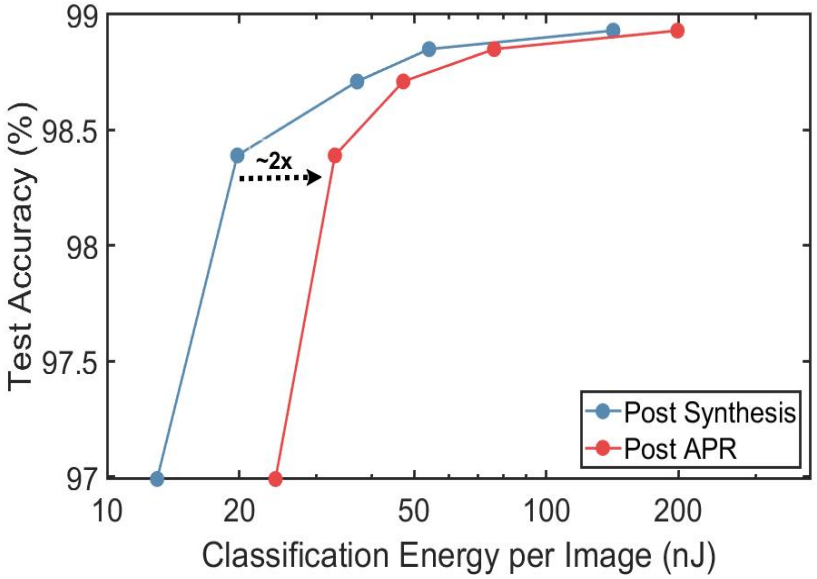

Figure 4.9: Post-synthesis and Post-layout Classification Energy Comparison

Chapter 5

CONCLUSION

In this work, we presented techniques for energy efficient hardware design of neuromorphic algorithms. By operating on SNN algorithms (SNN-DC and SNN-CT) input and output activations were made as binary which aided in getting relieved of multiplier units. We have demonstrated 20.8X improvement in latency (for N-MNIST dataset) by exploiting spike sparsity using a spike scheduler unit. Pipelining was exercised in classification hardware to improve the throughput. We have pointed that pipeline stalls can be decreased by handling handshake signals between pipeline stages. We have introduced a configurable neuron unit and time-multiplexed hardware using which programmability was added to the SNN MLP hardware. We also presented an analysis on an optimized combination of very low precision and structured compression for favorable energy, area, and accuracy tradeoffs, based on many DNN implementations in 40nm LP CMOS. Using structured compression showed 20X weight memory reduction on DNN MLP compared to floating-point DNN counterparts, with minimal accuracy degradation (less than 0.5%). The MLP DNN designed with 8-bit activations, 3-bit weights, and 8X structured compression showed 98.4% accuracy at $33nJ$ energy per classification at 100Mhz, outperforming further lower precision designs with less structured compression. The programmable MLP SNN was implemented in 28nm CMOS, demonstrating high accuracy and low energy. SNN-DC shows 98.0-98.70% accuracy at 51.4773 nJ per classification for MNIST and SNN-CT shows 96.33% accuracy at $294nJ$ per classification for N-MNIST.

REFERENCES

[1] Shihui Yin, Shreyas K Venkataramanaiah, Gregory K Chen, Ram Krishnamurthy, Yu Cao, Chaitali Chakrabarti, and Jae-sun Seo. Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations. In *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2017.

[2] Gregory K. Chen Ram Krishnamurthy Yu Cao Chaitali Chakrabarti Shihui Yin, Shreyas K. Venkataramanaiah and Jae sun Seo. Minimizing area and energy of deep learning hardware design using collective low precision and structured compression. In *Asilomar Conference on Signals, Systems, and Computers*, 2017.

[3] Deeplearning4j. Introduction to deep neural networks, 2017. URL `https://deeplearning4j.org/neuralnet-overview`. [Online; accessed 20-February-2018].

[4] SUNGTAE'S AWESOME HOMEPAGE. Feedforward neural networks, 2017. URL `https://www.cc.gatech.edu/ san37/post/dlhc-fnn/`. [Online; accessed 20-February-2018].

[5] Garrick Orchard, Ajinkya Jayawant, Gregory K Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in neuroscience*, 9:437, 2015.

[6] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[7] Andrew S Cassidy, Paul Merolla, John V Arthur, Steve K Esser, Bryan Jackson, Rodrigo Alvarez-Icaza, Pallab Datta, Jun Sawada, Theodore M Wong, Vitaly Feldman, et al. Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–10. IEEE, 2013.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4960–4964. IEEE, 2016.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[12] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[13] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[14] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[15] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560. ACM, 2017.

[16] Deepak Kadetotad, Sairam Arunachalam, Chaitali Chakrabarti, and Jae-sun Seo. Efficient memory compression in deep neural networks using coarse-grain sparsification for speech applications. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 78. ACM, 2016.

[17] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[18] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. *arXiv preprint*, 2017.

[19] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017.

[20] Filip Ponulak and Andrzej Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011.

[21] Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–8. IEEE, 2015.

[22] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, and Michael Pfeiffer. Theory and tools for the conversion of analog to spiking convolutional neural networks. *arXiv preprint arXiv:1612.04052*, 2016.

[23] Eric Hunsberger and Chris Eliasmith. Training spiking deep networks for neuromorphic hardware. *arXiv preprint arXiv:1611.05141*, 2016.

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[25] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[26] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[27] Shihui Yin, Deepak Kadetotad, Bonan Yan, Chang Song, Yiran Chen, Chaitali Chakrabarti, and Jae-sun Seo. Low-power neuromorphic speech recognition engine with coarse-grain sparsity. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 111–114. IEEE, 2017.

[28] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and RD Blanton. Lightnn: Filling the gap between conventional deep neural networks and binarized networks. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 35–40. ACM, 2017.

[29] Paul N Whatmough, Sae Kyu Lee, Hyunkwang Lee, Saketh Rama, David Brooks, and Gu-Yeon Wei. 14.3 a 28nm soc with a 1.2 ghz 568nj/prediction sparse deep-neural-network engine with¿ 0.1 timing error rate tolerance for iot applications. In *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pages 242–243. IEEE, 2017.

[30] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[31] Bert Moons and Marian Verhelst. A 0.3–2.6 tops/w precision-scalable processor for real-time large-scale convnets. In *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on*, pages 1–2. IEEE, 2016.