ABSTRACT

BAKER, LEE B. Design of a 3DIC system to aid in the acceleration of edge systems that employ multiple instances of disparate artificial neural networks. (Under the direction of Paul Franzon.)

Although Artificial Neural Networks (ANN) have been known about for many decades, it hasnt been until the last few years that they have demonstrated efficacy in applications such as image recognition and voice recognition. These ANNs have demonstrated significant improvements over what was considered to be state-of-the-art algorithms.

In most cases, ANNs must be "trained" to perform their function and these recent breakthoughs can, in a large part be attributed to the availability of "big" training data.

This data takes the form of images, recordings etc. and this along with the available storage capacity means that the required training data is now available to train these ANNs.

Artificial Neurons (AN) take their inspiration from neuron behavior observed in the mammalian brain, although implementations are simplifications of what actually exists in the brain. These simplifications range from attempts to emulate the actual spiking behavior of real neurons to ANs that simply encode the spiking behavior in the form of a number or rate.

Surprisingly, the ANNs that have demonstrated the most efficacy are those that employ the more simpler rate-based ANs. Now this may be in large part because these simpler rate-based ANs are easier to process or because large NNs employing the more complex spiking ANs have yet to demonsrate improved efficacy in the various applications. There is a belief that the more complex spiking neurons have the ability to outperform the simpler rate-based neurons but this work focuses on the proven efficacy of ANNs formed from rate-based ANs.

The ANNs that have demonstrated to be most effective are a family of neural networks that can be described as Deep Neural Networks. These DNNs are created by cascading layers of ANs to form a large, layered ANN. These ANNs are in most cases generate outputs in the form of a classification, such as the probability of an image containing a certain object or the output of some approximated function, which might be the expected cost from making a stock trade.

Now researchers have experimented with various sized NNs for various applcations, but those that have demonstrated the most efficacy employ tens of thousands of ANs.

In many implementations of these useful sized NNs, the performance is impacted by the memory bandwidth of the system. Much of the NN application specific (ASIC/ASIP) research has focused on taking advantage of the performance and ease of use of Static Random Access Memory or SRAM. These implementations can be shown to be effective with specific NN architectures, such as Convolutional NNs but in reality, these implementations do not provide the flexibility, storage capacity and deterministic performance required to implement all useful sized NNs.

In addition, it is this works belief that that real-world applications will employ multiple instances of these useful sized ANNs and current implementations will not meet the demands of these multi NN systems.

One area of integrated circuit technology that hasnt been widely used in ANNs is 3-D integrated circuits (3DIC). 3DIC has the potential to increase connectivity, and thus bandwidth and keep power dissipation to within acceptable levels.

This work combines ANNs with 3DIC technology to demonstrate how a 3DIC dynamic random access memory (DRAM) memory can be combined with customized IC layers to produce a system providing an acceptable level of performance in systems with multiple instances of various types of DNNs.

This work includes utilizing a customized 3D DRAM along with a system which includes a management layer which coordinates and executes ANN operations in the form of unique instructions and a processing layer able to process data from the DRAM, via the manager at a bandwidth that meets the demands of a system employing multiple ANNs.

Design of a 3DIC system to aid in the acceleration of edge systems that employ multiple instances of disparate artificial neural networks

by
Lee B. Baker

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2017

APPROVED BY:

Winser Alexander	Gregory Byrd
Richard Warr	Paul Franzon Chair of Advisory Committee

DEDICATION

To my wife Mandy, my children Adam, Rachel and Paul and my parents Joan and Barry.

BIOGRAPHY

The author was born in a the United Kingdom. After performing poorly in high school he took a job in a local electronic engineering firm under a vocational program. After seeing the white coated "engineers" being called down from upstairs to solve the "big" problems, he decided he wanted to wear one of those white coats (his dress sense was wanting). The journey took him to Brighton Polytechnic, now Brighton University and a First Class Honours Degree in Electrical Engineering. After working in the UK for a couple of years, he moved to the United States. The journey included a family with a daughter and two son's. The education continued with a Masters in Engineering from Villanova University and a Masters in Business Administration from North Carolina State University.

With the family now being somewhat independent, he decided to make a career change which would hopefully include teaching.

That career change included enrolling in the Electrical Engineering PhD program at North Carolina State University. This stage of the education journey has resulted in this dissertation.

And remember:

"do not stand still."

"do not let your past dictate your future."

ACKNOWLEDGEMENTS

At a personal level, I would like to thank my wife Mandy and my children Adam, Rachel and Paul for their encouragement.

I would like to thank my advisor, Paul Franzon for his help in making this possible.

I would also like to thank my fellow students, especially Jong Beom, Josh, Sumon and Weifu for their healthy discussions and, being an older student, referring to me as Lee and not Sir or Mr. Baker.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Recently, there has been much interest in the use of artificial neural networks in systems that employ tasks such as image recognition[Kri12], text recognition[Qiu13] and game playing[Mad14]. In particular, in the field of image recognition these artificial neural network models have demonstrated superior performance over other state-of-the-art technology[Kri12]. These artificial neural networks will continue to be applied to numerous other areas such as voice recognition, text recognition, face recognition and autonomous control.

Artificial neural networks (ANN) take their inspiration from neuron behavior observed in the mammalian brain, although implementations are simplifications of what actually exists in the brain.

The mammalian neuron is a cell that receives input and generates output in the form of electrical and chemical processes. The neuron has a cell body (or soma), a group of dendrites which provide

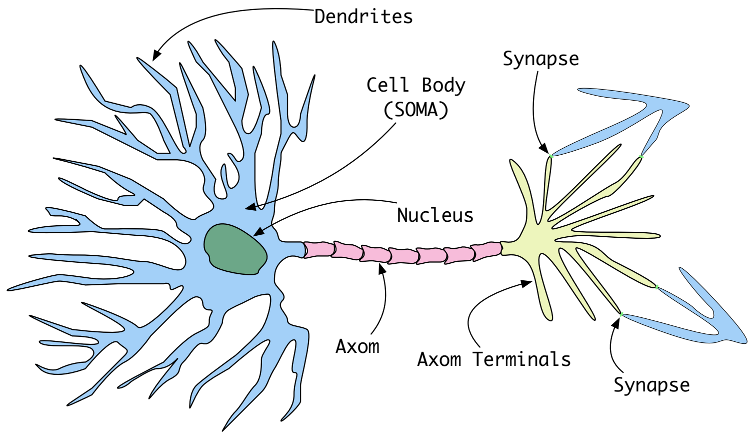


Figure 1.1 Artists impression of a Mammalian Neuron

the inputs from other cells, a cell body, an axom which generates the output signals, and the axom terminals which are the outputs of the cell. The connection from a cells output, or axom terminal to another cells input, or dendrite is known as a synapse. The connection in the synapse is a chemical process stimulated by electrical impulses. The neuron can be seen in figure 1.1.

The connection from one cell to another has both an associated delay and a strength. The strength of the connection can be influenced by the size of the pre-synaptic neuron spike or by the pre-synaptic neuron generating a series of spikes rather than a single spike.

So it is known that mammalian neurons generate "spikes" in response to inputs which for humans include sight, touch, sound etc.. This spiking behavior is often referred to as the neuron being activated. When these neurons are activated, their spikes propagate to other neurons. Under certain conditions, the combination of the various inputs to a neuron cause it to activate. A particular neuron may have many hundreds, perhaps thousands of other neurons connected to its "input". These input neurons are referred to as pre-synaptic neurons. These pre-synaptic neurons may provide input to many neurons which are referred to as post-synaptic neurons. A particular neuron can get activated by a particular arrival pattern of pre-synaptic neuron spikes or simply by the intensity of the pre-synaptic spikes.

The spiking behavior of a neuron also varies and many spiking profiles have been observed, including single spikes, groups of spikes and repetitive spiking. It is believed that information is carried in the delay and strength of the connections and how pre-synaptic neurons combine to cause a neuron to activate. In simple terms, if a neuron is activated by its pre-synaptic neurons, then the activation of the neuron means a pattern has been detected which will influence a reaction. In mammalian terms, that might be the detection of a threat from both smell and sight neurons and the reaction is to control muscles resulting in flight.

The various chemical and electrical processes that result in the generation and propagation of these neuron spikes is beyond the scope of this dissertation, but how neurons and networks of neurons are artificially emulated is what we will duscuss next.

1.1 Artificial Neural Networks

When modeling these neurons in artificial neural networks, the neuron models either generate actual spikes similar to actual neurons or produce a value which is proportional to the rate at which spikes occur. These artificial neural networks can be categorized as rate-based coded or spike time coded neurons.

When used in networks of neurons, both model types employ a connection weight between the pre and post-synaptic neuron, however, the spiking neuron network also introduces a time delay associated with the connection.

The spiking neuron model is characterized by:

- Connections between neurons have both a strength and a delay
  - The pre-synaptic neuron output is multiplied by the connection weight and delayed
- The weighted inputs from all pre-synaptic neurons are accumulated
- The accumulated inputs drives an activation function

  - the activation function $f(x)$ is a spiking model is based on differential equations
  - many models have been proposed with varying levels of complexity

    examples are:

    · Leaky integrate and fire

    · Izhikevich [Izh04] (see Fig. 1.4a)

The Rate-based neuron model is characterized by:

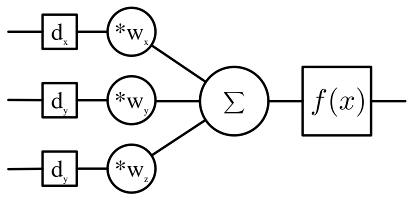
- Connections between neurons have only a strength
  - The pre-synaptic neuron output is multiplied by the connection weight
- The weighted inputs from all pre-synaptic neurons are accumulated
- The accumulated inputs drives an activation function
  - the activation function $f(x)$ is a non-linear function
  - early models used binary functions although in practice the function needs to be differentiable

    examples are (see Fig. 1.3):

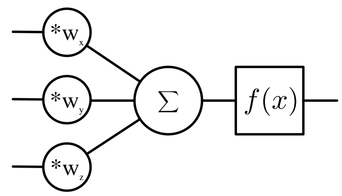
    · sigmoid

    · rectified linear unit

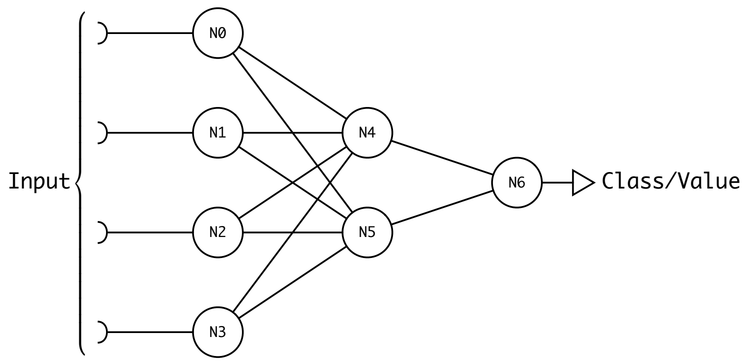
To emulate complex behavior, the artifical neurons are connected in networks, typically with layers of sub-networks which are in affect separated by the non-linear activation function. Examples of both rate-based and spiking artificial neural networks can be seen in Fig. 1.5a and Fig. 1.5b respectively. Typically neural networks process in a feed-forward fashion. Considering Fig. 1.2, this means the input arrives on the left, the inputs propagate to neurons N0 through N3. WHen N0 through N3 are processed, their values propagate forward to neurons N4 and N5 etc.. SOmetime ANNs also include



(a) Spiking Model

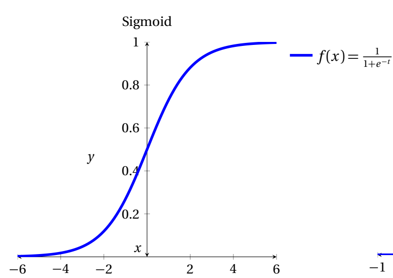


(b) Rate-Based Model

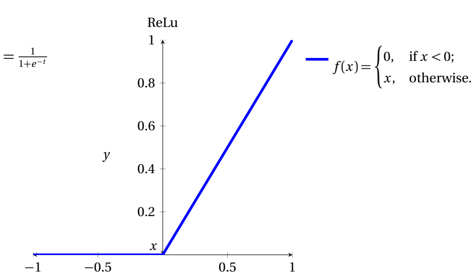


(c) Network of Artificial Neurons

Figure 1.2 Artificial Neurons and Network



(a) Sigmoid

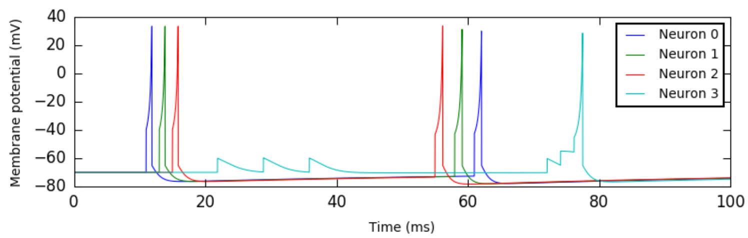


(b) Relu

Figure 1.3 Example Rated-Based Model Activation functions

$$
\begin{aligned}
&v' = 0.04v^2 + 5v + 140 - u - I \\
&u' = a(bv - u) \\
&\text{if } v \geq 30\,\text{mV}, \text{ then } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases}
\end{aligned}
$$

(a) Izhikevich Model[Izh04]



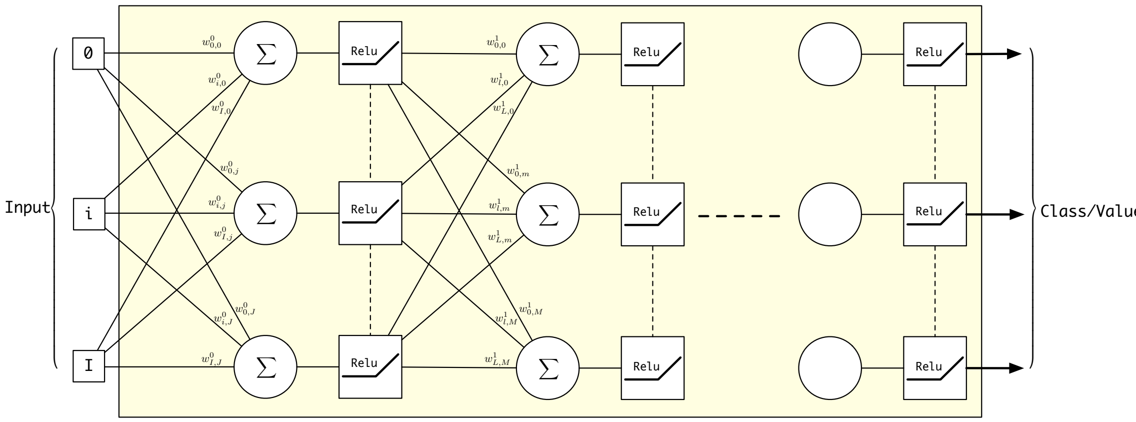
(b) Izhikevich [Izh04] Model Simulation

Figure 1.4 Example Spiking Activation Function Model

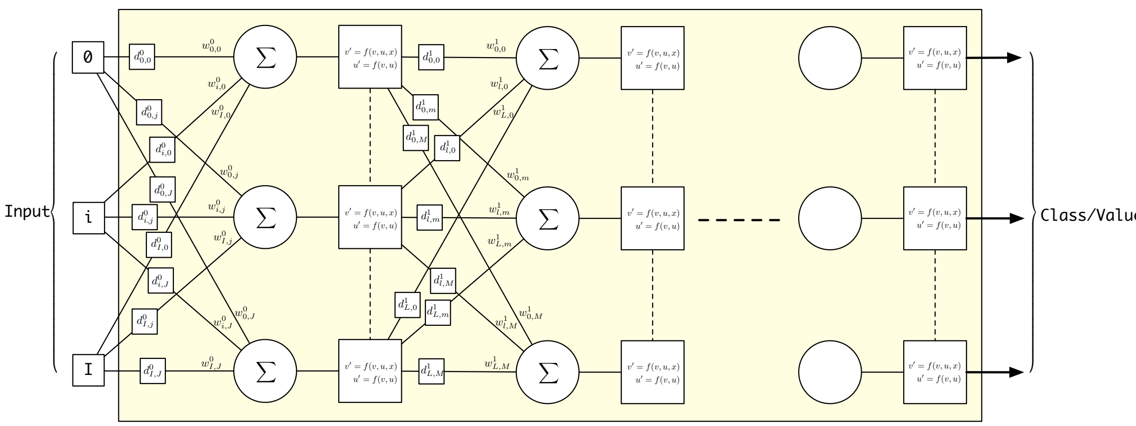
recursion where for example neurons N0 through N4 are not only influenced by the input, but also by themselves. Many ANNs operate only in feed-forward fashion but some popular ANNs, such as Long short-term memory (LSTM), employ recursion.

Another popular ANN known as Deep Neural Networks (DNN) have proved very popular over the last few years. They get good press in applications such as image recognition and speech recognition. Deep Neural Networks are often formed from tens of layers of ANs with each layer containing many ANs. DNNs are also processed in a feed-forward manner with one layer being the inputs to the next layer. As mentioned [Kri12], these useful DNNs often require hundreds of thousands of ANs and within the network, each AN can have hundreds, even thousands of feeder or pre-synaptic ANs. There have been implementations that use different number formats from double precision floating point to eight bit integers, but in all cases these useful ANNs require a significant amount of memory to store the connection weights (parameters).

Although the spiking neural network more closely models the behavior of real neurons, over the



(a) Rate-based Model Artificial Neural Network (with ReLu activation function))



(b) Spiking-based Model Artificial Neural Network

Figure 1.5 Example Artificial Neural Networks

last 20 years there have been breakthroughs in the configuring of rate-based models especially with the introduction of the back-propagation algorithm and stochastic gradient descent. Along with the abundance of data now available in the form of voice, images etc. to "teach" these networks using back-propagation, most of the effective applications of artificial neural networks have employed these rate-based models.

1.2 The Problem

To approach the capabilities observed in human behavior, such as object recognition these ANNs become very large. They often utilize hundreds of thousands of neurons to implement what a human would consider a relatively straightforward task. For example, a "useful" ANN similar to that described in [Kri12] that is used to recognize up to 1000 different object classes has a network size of approximately 650,000 neurons and 630 million synaptic connections [Kri].

The increased performance of ANNs over classical methods in image recognition and voice recognition might suggest that ANNs might out-perform other existing systems in other applications. There is reason to believe that these ANNs will replace existing control and monitoring functions in existing systems.

If ANNs fulfill their potential, systems employing ANNs will utilize them for various functions, such as engine monitoring, anomaly detection, navigation etc. all within the same system. Considering the various functions a complex customer facing or edge application system performs, it is likely that many real-world applications will employ multiple disparate instances of these useful sized ANNs. Assuming these complex functions will require ANNs similar in size to [Kri12], these implementations will be processing multiple large ANNs at or near real-time.

Considering the storage required for the input, the AN states and most significantly the weights for each of the ANs, the storage requirements results in gigabytes of memory. When these ANNs are required to be solved in fractions of a second, the processing and memory bandwidth becomes prohibitive.

As a metric, this work asumes that any useful DNN will employ 100's of thousands of ANs. Although there is a lot of debate regarding number formats for ANNs, this work also assumes single-precision floating point. Assuming an ANN with 250K neurons and an average fanin to each AN of 2000, a system employing 10 ANNs for various disparate functions and an average processing time of 10 ms suggests a average bandwidth of 16 Tbit/s (see equation 1.1).

$$
\begin{aligned}
\text{Average Bandwidth} &= \sum_{\mathbf{n}=0}^{\bar{\mathbf{N_n}}} \left(\frac{\bar{\mathbf{N_a}} \cdot \bar{\mathbf{C_p}} \cdot \bar{\mathbf{b_w}}}{\bar{\mathbf{T_p}}} \right) \\
&= \sum_{\mathbf{n}=0}^{9} \left(\frac{250 \times 10^4 \cdot 2 \times 10^3 \cdot 32}{10 \times 10^{-3}} \right) \\
&= 16\,\text{Tbit/s} \qquad (1.1)
\end{aligned}
$$

where $\mathbf{N}_n$ is the number of ANNs

$\mathbf{N}_a$ is the average number of ANs

$\mathbf{C_p}$ is the average number of connections

and $\mathbf{T_p}$ is the processing time

Therefore, when imlementing ANNs, the memory requirements are significant. The storage is required for the input, the AN state and most significantly the weights for each of the ANs. This storage requirement often results in gigabytes of memory.

In addtion, in edge applications, it is anticipated that these ANNs are required to be solved in fractions of a second, in which case the processing and memory bandwidth becomes prohibitive.

The problem becomes "**to provide deterministic at or near real-time performance within tolerable power and space constraints for edge systems employing inference on multiple disparate useful-sized neural networks.**"

Considering that DRAM is required to store the NN parameters, why use SRAM as an intermediate store? Well, in practice there are benefits if you can operate solely out of SRAM. Certainly good performance and potentially low power. But use of SRAM makes assumptions on the NNs that can be supported. The primary requirement of the NN to allow effective use of SRAM is "reuse". Once parameters are stored in SRAM, can they be reused such that the SRAM isn't simply an intermediate memory but something akin to a cache.

In some ANNs there are reuse opportunities. Specifially, with CNNs, the weights are reused. A convolutional filter is passed across an input to form the next layer. These filter "kernels" can be held in memory and the input is read from DRAM thus reducing the DRAM bandwidth. Even with DNNs where weights may not be reused, when implementing multiple DNNs, there is opportunity to hold the input in memory. If the system is being employed in cloud applications or in training, there is opportunity to reuse inputs whilst performing batch processing.

But SRAM comes at a price, its big. Often when we see physical layouts of NN processors, they are dominated by the silicon area of the SRAM. The area required for SRAM has been understood for quite some time and companies attempt to create custom SRAMs to minimize the area impact.

So the question becomes, can a system employ DRAM with minimal SRAM and still provide a high performance system within acceptable area constraints?

Even in cloud applications, there are limitations on reuse. We paraphrase a quote from a Google paper [Aba15] on their Tensor Processing Unit ASIC (TPU):

“the architecture research community is paying attention to NNs, but of all the papers at ISCA 2016 on hardware accelerators for NNs, alas, all nine papers looked at CNNs, and only two mentioned other NNs. Unfortunately CNNs represent only about 5% of our datacenter NN workload”

The applications targeted by the google TPU [Aba15] assume multiple requests, so reuse in the form of batch processing is still of great benefit, but the bulk of the requests in [Aba15] are fully-connected DNNs and in these cases weight reuse is not as benefitial and the performance of the TPU is degraded when implementing these fully-connected DNNs.

Therefore, implementations that focus on CNNs can suffer from severe degradation in performance when targeting generic types of ANN, such as locally and fully connected DNNs and LSTMs.

This work focuses on edge applications employing disparate ANNs and assumes both weight reuse and batch processing do not apply. Considering systems will want to perform multiple DNNs simultaneously suggests that these edge systems will require usable memory bandwidth of the order of 10’s of Tbit/s.

In these cases, **DRAM bandwidth is the bottleneck**.

Some might suggest the requirements of these applications would be satisified by employing multiple graphics processor units(GPU). In fact, Graphics processing Units (GPU) are used to implement large ANNs and in some ANN architectures, such as Convolutional NNs (CNN), they are quite effective. However, we should not forget they are not optimized purely for ANN processing and are restricted by available SRAM and they are power hungry. These limitations will limit the effectiveness of GPUs regardless of what we might hear from the GPU community. Even in the case of newer GPUs which are employing 2.5DIC technology, the memory bandwidth will still be limited by available DRAM tecnology. For example, a 2.5D solution employing High bandwidth Memory (HBM) would be limited to a maximum raw bandwith of the order of 4 Tbit/s. Also, its has proven very difficult, if not impossible to take advantage of the available memory bandwidth [Far11] [Aba15]. Given these multiple GPU systems have high real-estate and power requirements and given each instance consumes of the order of 100 W to 200 W. Overall GPUs have limited suitability to meet edge application requirements.

Much of the ANN application specific (ASIC/ASIP) research has focused on taking advantage of the performance and ease of use of Static Random Access Memory or SRAM. These implementations can be shown to be effective with specific ANN architectures (CNN), server applications or the "toy examples" but when a system requires multiple disparate ANNs in an edge application, these implementations do not provide the required flexibility, storage capacity and deterministic performance.

1.3 The Solution

Most researchers acknowledge that realistically, DRAM is required to meet the main storage requirements of useful sized ANNs.

We further believe that to support all types of disparate ANNs, we need to be able to operate directly from the DRAM memory.

This is because SRAM-based solutions assume memory locality when processing a neural network. However, when ANNs do not provide sufficient locality these solutions become DRAM bandwidth bound. If we then ensure the DRAM can feed the SRAM at the necessary bandwidth, why use an SRAM and waste the significant silicon area they require.

This works system operates directly from DRAM, but not just DRAM, 3D-DRAM.

In addition, this work has designed a system that can stay within the physical footprint of the 3D-DRAM.

By ensuring the system stays physically within the 3D stack, we take advantage of high density connectivity provided by TSVs. Therefore, this work is able to propose a custom 3D-DRAM that exposes more of the DRAMs internal page and thus generates interface bandwidth that is of the order of 64 times that of the standard 3D-DRAM.

1.3.1 Novelty

The novelty of this work includes:

- A custom 3D-DRAM providing a 64X bandwidth benefit compared to standard 3D-DRAM
- A system that benefits from the power and performance benefits of 3DIC technology by remaining within a 3DIC stack
- New data structures that allow use to operate directly out of DRAM whilst ensuring effective use of DRAM bandwidth
- A system that can simultaneously process multiple disparate ANNs at or near real-time

This research explores a 3DIC solution using a custom organized 3DIC memory in conjunction with unique data structures and custom processing modules to significantly reduce the area and power footprint of an application that needs to support the processing associated with multiple ANNs.

This works system will provide at or near real-time performance required for systems employing multiple ANNs whilst staying within acceptable area and power limits and will provide greater than an order of magnitude benefit over comparable solutions.

CHAPTER

2

SYSTEM DESCRIPTION

2.1 ANN System

The primary design considerations of this work are :

- Consider systems that are unable to take advantage of memory reuse opportunities and therefore not able to achieve high performance using local SRAM to store ANN parameters or the ANN input
- Acknowedge that DRAM is required for storage of ANN parameters
- Assume edge devices will likely apply many disparate ANNs to perform various system functions
- Assume that many edge applications will have space and power limitations

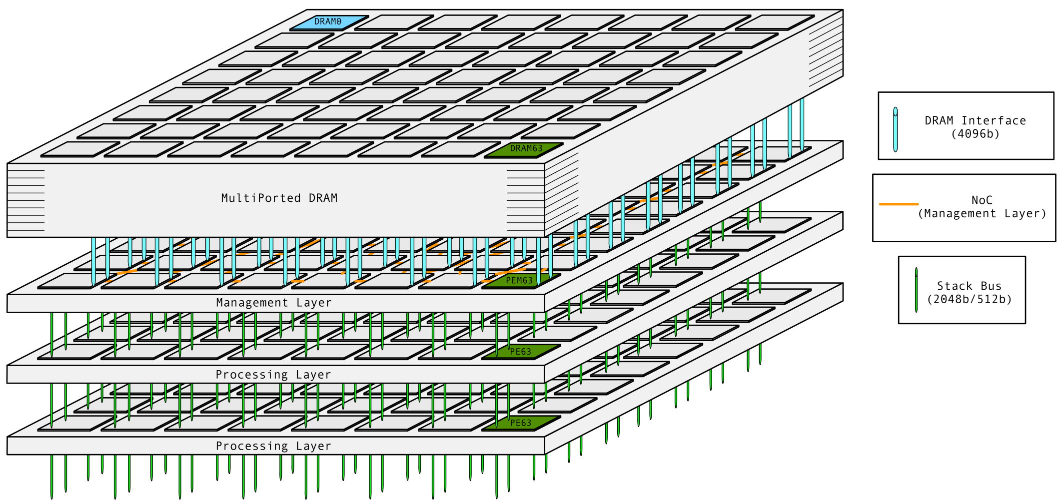


Figure 2.1 3DIC System Stack

These requirements suggest that reuse opportunities will not provide significant performance boosts and therefore this work focused on using the DRAM directly as the operational storage and not rely on a significant amount of local SRAM. This work employs 3DIC technology along with a custom 3D-DRAM. The objective was to demonstrate that a pure 3DIC system can implement multiple disparate ANNs. By staying within the 3DIC footprint and taking advantage of high density through-silicon-vias (TSV) this work is able to maintain a significantly higher bandwidth over 2D or 2.5D ASIC/ASIP solutions.

The 3DIC system die stack (figure 2.1) includes the 3D-DRAM with a system manager below and one or more processing layers below the manager.

3D-DRAM has recently become available in standards such as High Bandwidth Memory (HBM) and Hybrid Memory Cube (HMC) and proprietary devices such as the DiRAM4 available from Tezzaron. These technologies provide high capacity within a small footprint.

In the case of HBM and DiRAM4, the technology can be combined with additional custom layers to provide a system solution.

The question becomes, can a useful system coexist within the same 3D footprint?

This work targeted a baselne system with:

- target single precision floating point for computations
- use the Tezzaron DiRAM4 DRAM for area estimates and memory controller design

The work includes customizing the interface to a 3D-DRAM, researching data structures to describe storage of ANN parameters, designing a memory manager with micro-coded instructions and a processing engine (PE) layer. The system is designed such that a sub-system, known as a sub-system column (SSC) operates on one of these disjoint memories within the 3D-DRAM (see figure 2.2).

When the sub-system columns need to share data or neuron activations, the data is passed between SSCs using a network-on-chip (NoC).

An overview of the various blocks and interconnects are given below:

2.1.1 3D-DRAM

The targetted 3D-DRAM, the Tezzaron DiRAM4 is a 3D-DRAM employs multiple memory array layers in conjunction with a control and IO layer. The memory is formed from 64 disjoint sub-memories each providing upwards of 1Gigabit with a total capacity of at least 64 gigabit.

2.1.2 Manager Layer

The Manager block is the main controller in the system. The operations required to process an ANN are formed from individual instructions which are decoded by the Manager. These instructions include descriptors to describe memory read operations, processing engine operations and memory write operations. The manager reads these system instructions from an instruction memory, decodes the instruction and configures the various blocks in the system. The configuration includes:

- initiate operand reads from DRAM

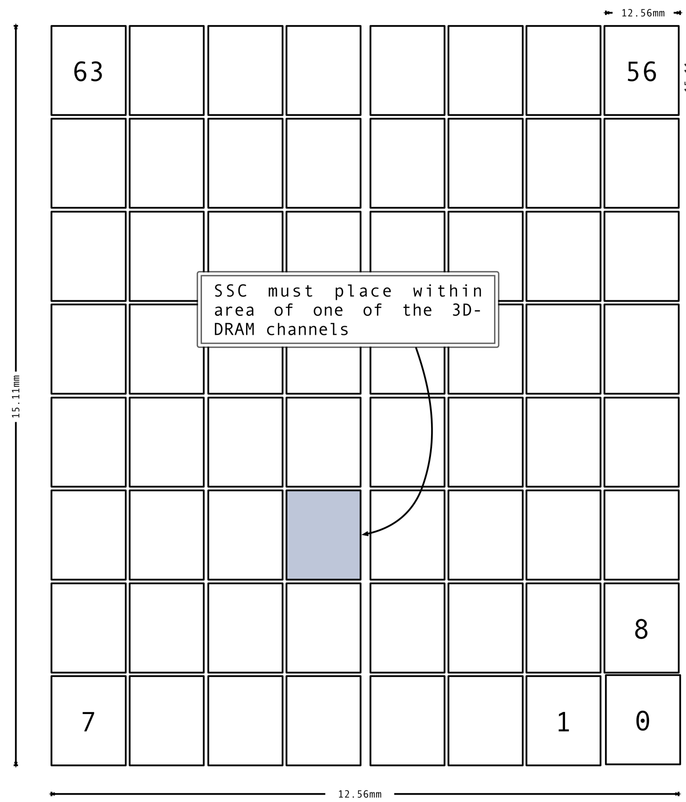


Figure 2.2 DRAM Physical Interface Layout showing area for SSC

- prepare the processing engine (PE) to operate on the operands
- prepare the result processing engine to take the resulting neuron activations from the PE and write those results back to the DRAM
- replicate the resulting neuron activation's to neighbor managers for processing of other ANN layers

2.1.3 Processing Layer

The PE is able to operate on data streamed directly from the DRAM via the Manager layer. The PE is configured by the manager to perform operations on the operand data streamed from the manager. In the baseline system, the main operation is to perform multiply-accumulates on 32 execution lanes of two operands. These operands typically are the pre-synaptic neuron activation's and the connection weights. The PE also performs the activation function on the result of the MAC to generate the neuron activation value. These 32 activation values are sent back to the Manager layer.

2.1.4 Layer Interconnect

The layers are connected using through-silicon-vias (TSVs) which provide high connection density, high bandwidth and low energy. By ensuring the system stays within the 3D footprint ensures we can take advantage of the huge benefits provided by TSVs.

2.1.5 Inter-Manager Communication

During configuration and/or computations, data must be transported between managers. This inter-manager communication is provided by an NoC. When computing an ANN across multiple processing sub-systems, often neuron activation data must be shared between these SSCs. The SSC includes the DRAM port, the manager and the PE. An NoC within each management block communicates with each adjacent manager using a mesh network. This NoC has a forwarding table that can be reconfigured to provide more efficient routing for a given processing step.

A control and data flow diagram of the stack showing the 64 sub-system columns can be seen in figure 2.3.

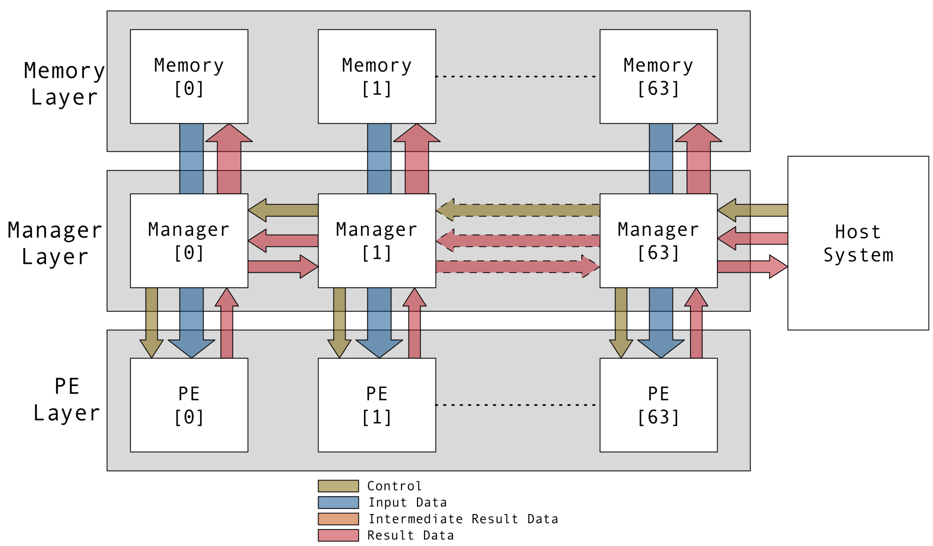


Figure 2.3 System Flow Diagram

CHAPTER

3

SYSTEM OPERATIONS

In the context of this system and AN state calculation, the basic operations to determine the state of a neuron is to:

- Inform the Manager and PE which operations are to be performed
- Tell the manager to access the states of the pre-synaptic neurons
- Tell the manager to access the weights of the connections from the pre-synaptic ANs
- Provide the pre-synaptic neuron weights and states to the processing engine execution lanes
- Tell the manager where to store the resulting AN state back to memory

This work has researched an instruction architecture to describe the above operations which are interpreted by the manager.

In the baseline system, the manager is not responsible for performing specific algorithm operations but is responsible for coordinating the various data flows and configuration of the modules that make up the system.

The managers primary responsibility is:

- Instruction decode
- Internal Configuration messages
- Operand read
- Result write

In the baseline system, the PE is responsible for the main algorithm operations.

The PE has three major blocks:

- Streaming operation function (stOp)
  - Processes data from the manager on-the-fly without storing in local SRAM
- SIMD
  - processes the data from the stOp function, usually neuron activation such as ReLu
- DMA/local memory controller
  - transfer configuration data to PE conroller or to store stOp results to a small local SRAM which can be used for access by SIMD or by the stOp function

3.1 Manager Operations

3.1.1 Instructions

The instructions communicate:

- To the Manager
  - ROI Storage descriptor
  - Parameter/Weight Storage Descriptor

    Broadcast or Vectored
  - Result write storage descriptor

    include descriptors for all destination managers
- To the PE
  - stOp operation
  - SIMD operation
  - Number of active lanes
  - Operand Vector length

Instructions include information to control the above operations.

Instructions contain sub-instruction called descriptors. These descriptors contain the information to control the various operations associated with the processing of a group of ANs.

The group size is related to the number of execution lanes which for the baseline system is 32. So a group can be anywhere from 1-32. It should be said that unless the group size consistently approaches 32 the system performance will be poor.

An instruction will typically have four descriptors:

1. Operation
2. Memory read for operand stream 0
3. Memory read for operand stream 1

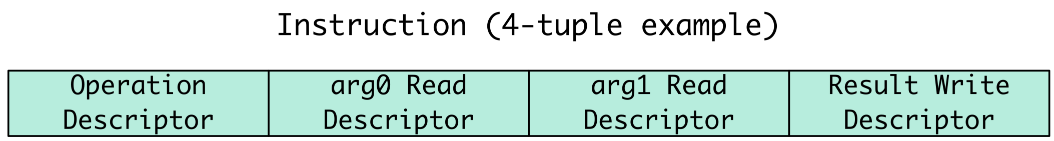


Figure 3.1 Instruction 4-tuple

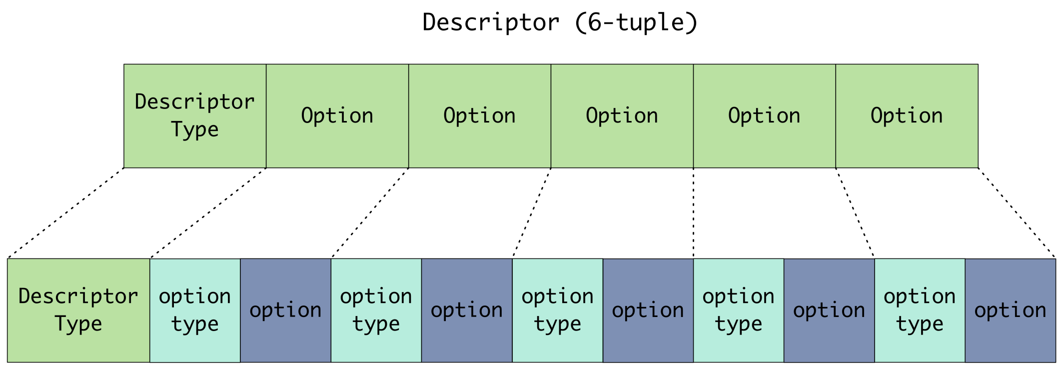


Figure 3.2 Descriptor 6-tuple

4. Result write

Note: An operand stream will be reffered to as an argument.

The instruction is actually an n-tuple where the tuple elements are descriptors and the number of elements can vary based on the operation being performed. In figure 3.1 we see the format of a 4-tuple instruction which we use to perform an activation calculation for a group of neurons.

Now within a descriptor, we need to describe the various options such as storage descriptor pointer, number of operands etc..

Again, we employ a n-tuple format where the first tuple element describes the descriptors operation followed by an m-tuple whose elements contain the options required for the operation.

These option elements are a two-tuple with option and associated value. In figure 3.2 we see the format of a 6-tuple descriptor.

3.1.2 Accessing of Pre-synaptic AN states and connection weights

As was discussed previously, the ANN input and configuration is stored in main DRAM memory. A part of the research is determining how to store the ANN input and parameters in such a way to effectively make use of main DRAM bandwidth. To provide parameters for the up to 32 execution lanes within the PE, we store the AN parameters in consecutive address locations. With one read to the DRAM, we access 128 words. This provides four weights for each of the 32 ANs being processed. These weights are sent to each lane of the PE over four cycles. We will discuss memory efficiency later, but by taking advantage of the multiple DRAM banks along with pre-fetching and buffering, we are able to achieve relatively high efficiency of the available maximum bandwidth.

Although AN parameters (weights) are stored in contiguous memory locations, providing the input state to a particular AN presents us with an interesting problem.

Most often DNNs are represented by layers of ANs whose pre-synaptic neurons are from the previous layer. These previous layers represent the input to a given layer. The first layers input is the actual input to the ANN.

The input can be represented in the form of a 2-D array of AN states. For the sake of generality, the input array elements are considered as AN states.

Any given AN operates on a region of interest (ROI) within the input array.

In figure 3.3, an input to a ANN layer in the form of a 2-D array along with the ROI of two ANs.

The various connection weights are stored in multiple contiguous sections. However, its not possible to arrange the input in such a way that each ANs ROI can be stored in contiguous memory locations. The figure above shows a typical ROI arrangement. Assuming the input array is stored in row-major order, an ROI is drawn from disjoint sections of memory. These disjoint sections contain a number of AN states and the sections are separated by a gap of a number of memory addresses. When the parameters are accessed when performing a particular operation, the memory controller within the manager must be informed of the start address and the lengths of the sections and gaps. Now this

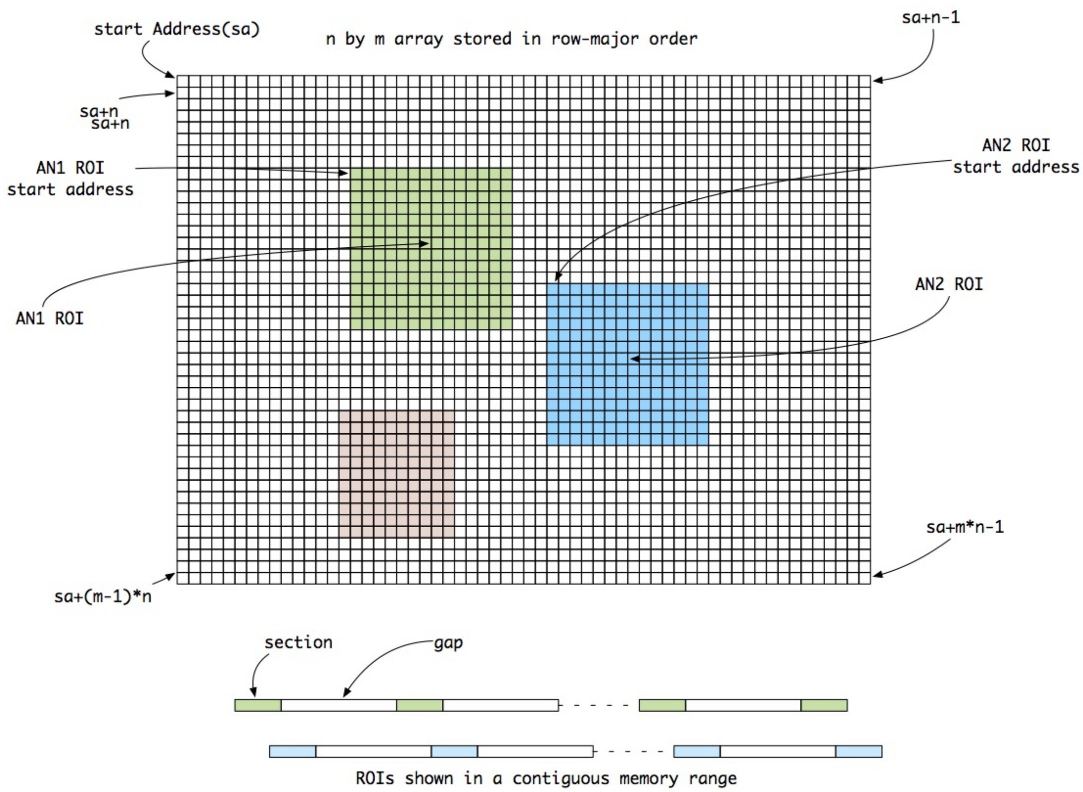


Figure 3.3 ROI Storage

looks problematic, and it is, but in practice groups of ANs share a common ROI. So once we solve the problem of efficiently reading an ROI from the DRAM, that ROI can be shared across a group of ANs

The read efficiency problem is solved by again taking advantage of the DRAMs banks and pages.

This work proposes a data structure to describe these ROI storage locations.

Although disparate groups of ANs may have a different start addresses for their ROI, a commonality is observed in the ROI section lengths and gaps. So for each AN group, the groups ROI starting address is stored along with a pointer to a common set of section length/gaps. This structure is termed a storage descriptor.

This storage descriptor contains, amongst other things the start address of the ROI and a pointer

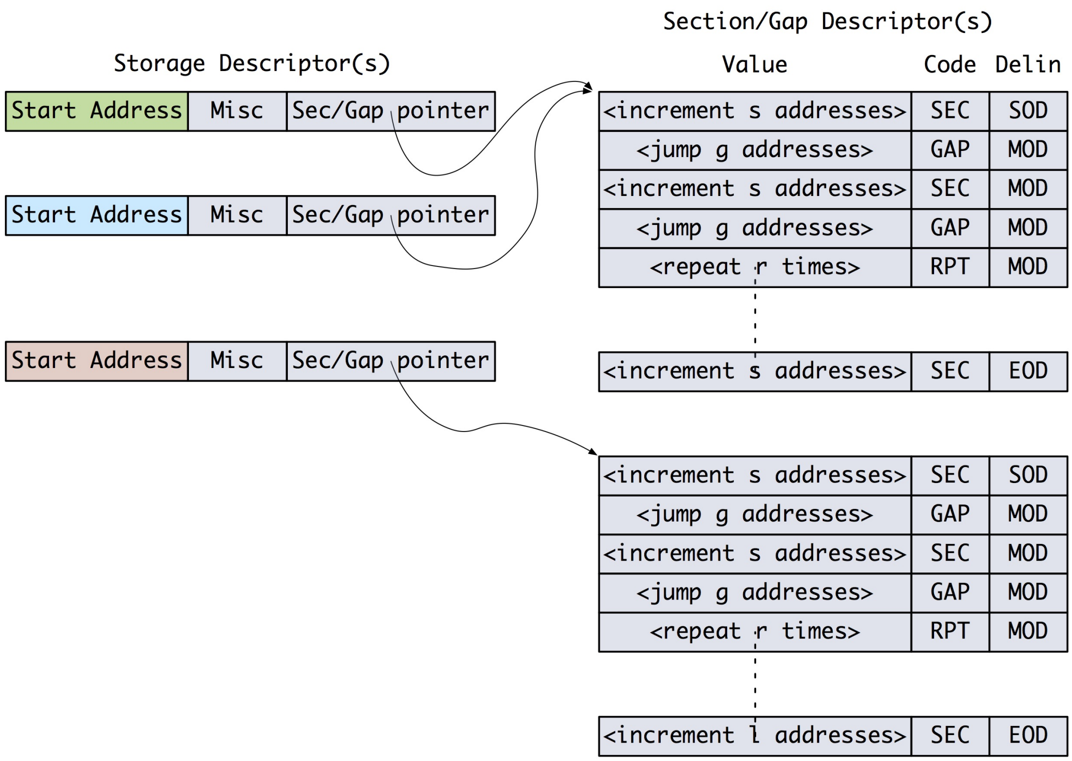


Figure 3.4 Storage Descriptor

to a section/gap descriptor. Many storage descriptors point to a common section/gap descriptor. This avoids having to have a unique section/gap descriptors for each AN group.

Figure 3.4 shows the structure of the storage descriptor. The SOD, MOD and EOD are used to delineate each descriptor in memory and stand for start-of-descriptor, middle-of-descriptor and end-of-descriptor.

3.1.3 Writing AN state results to memory

When the PE has processed the group of ANs, the new AN states are sent back to the manager. The manager will store these back to DRAM most likely in the array format as described earlier.

A significant difference taken advantage of is that for any given operation, the system is writing far less than is being read. For example, the ROI and parameters are usually vectors that will typically exceed 100 elements and in many cases much higher. When an operation is complete, in almost all cases one word per lane is writen back to main memory. Now that sounds like writing back has a very small impact on performance but with DRAMs that's not always true.

When the system writes the result of an operation back to memory, it is often writing a small portion of a DRAM page and the nature of the DRAM protocol means this is a very inefficient use of DRAM bandwidth. So although the amount of data written is small the performance impact cannot be ignored.

In addition, in many cases the results from a particular PE has to be provided not only to the PEs local manager but also to other managers. This is handled with a network-on-chip (NoC).

The result storage directives are communicated by using the same storage descriptor mechanism. However, the added complication is because the result will likely have to be replicated to other managers, the storage descriptors must be sent to all destination managers.

3.2 PE Operations

3.2.1 Streaming Operations (stOp)

The operations performed by the stOp are primarily multiple-accumulate with a transfer to the SIMD or to local memory.

Even though the baseline system focuses on the AN multiply-accumulate followed by a ReLu activation function, the system has built in flexibility into the stOp function to allow other functions to be added

In most cases, the stOp module will operate on the AN state and weights provided by the manager and provide the result to the SIMD.

3.2.2 SIMD

The SIMD is a 32-lane processor with some builtin special functions, such as the ReLu operation.

The SIMD will take the result provided by the stOp and perform a ReLu. The result will, in most cases, then transmitted back to the manager.

3.2.3 Configuration

To configure these operations, two pointers are sent to the PE. These pointers index into a small local memory which provides a program counter (PC) to the function to be performed by the SIMD and a configuration entry for the operation to be performed by the stOp.

The PE is able to perform its operation concurrently on 32-lanes. However, there are cases when less than 32-lanes will be employed. This may occur if the number of ANs being processed is not modulo-32. In this case, the manager provides the number of lanes being processed for any given operation. In addition, the length of the vector of operands is also sent by the manager to the PE.

CHAPTER

4

SUGGESTED DRAM CUSTOMIZATIONS

Accessing a "typical" DRAM involves opening a page in a bank, reading or writing a portion of the contents of the page then closing the page.

Typically a bank may contain of the order of a few thousand pages and a page may contain of the order of a few thousand bits.

Once the page is open, the user accesses a portion of the requested page over a bus. With PCB based DRAMs the bus might vary from four to 16 bits wide, but with 3D DRAMs, such as HBM the bus might be up to 128 bits wide.

Figure 4.1 shows a block diagram of a typical DRAM.

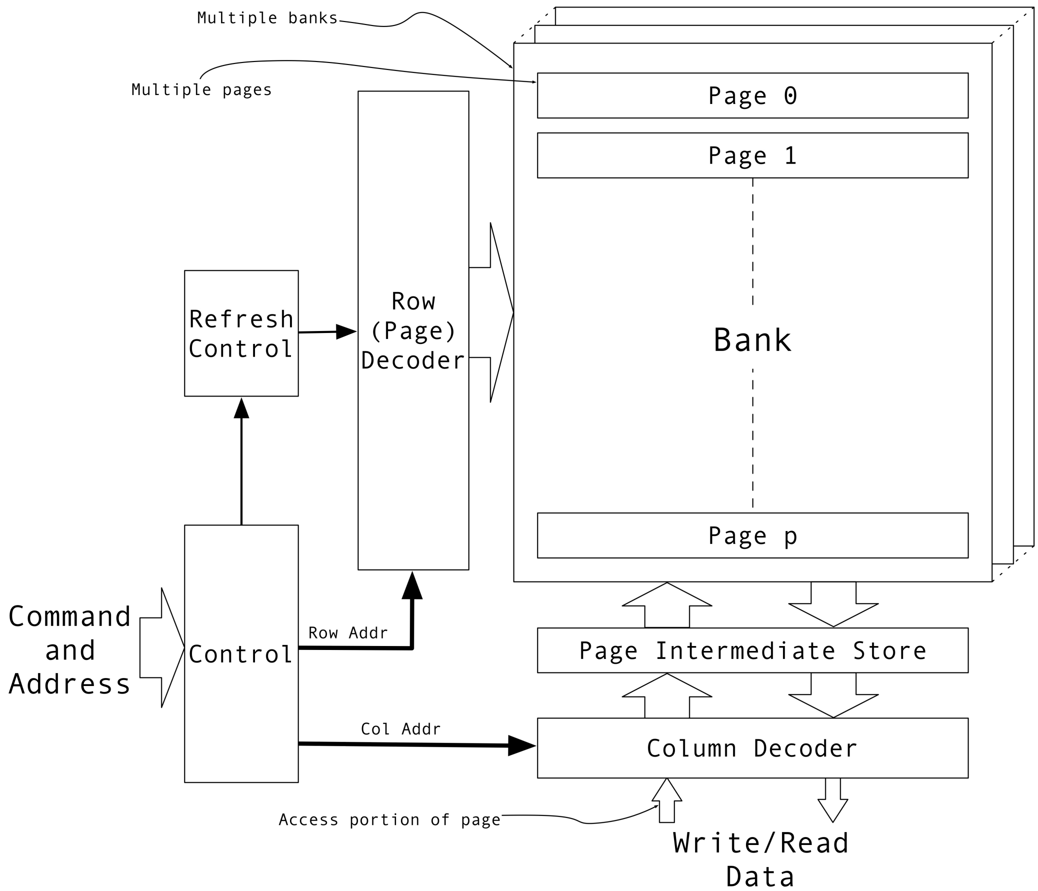


Figure 4.1 Typical DRAM Block Diagram

4.1 Expose more of the Page

This work achieves the increase in bandwidth by proposing that the DRAM expose more of its currently open page.

Without the limitations of having to transfer data beyond the chip stack, this work suggests exposing a larger portion of the page over a very wide bus. By staying within the 3D footprint, this bus can be implemented using fine pitch through-silicon-vias. (see figure 4.2).

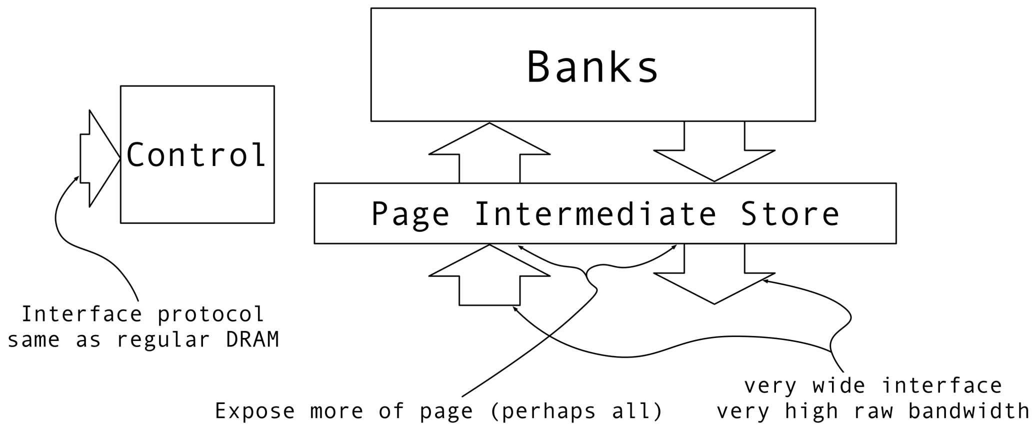


Figure 4.2 Exposing more of the DRAM page

4.2 DRAM Write Mask

When processing an ANN, to compute the activation of an individual AN involves reading the pre-synaptic AN activation's and the weights of the connections between the pre-synaptic ANs and the AN being processed. The activation of the processed AN is written back to memory. The ratio of reads to writes is high, 100's or 1000's to one. Therefore, the system often needs to write a portion of the page back to memory. To avoid a read/modify/write, a customization to the DRAM is the addition of a write data mask to the DRAM write path.

CHAPTER

5

DETAILED SYSTEM DESCRIPTION

A detailed flow diagram and block diagram of the sub-system column can be seen in figures 5.1 and 5.2 respectively.

5.1 Manager

5.1.1 Operation Decode

In figure 5.2, instructions are read from instruction memory and passed to the instruction decoder.

The operation tuple is decoded and a streaming operation (stOp) pointer and a SIMD operation pointer are sent to the PE inside an OOB control packet.

The stOp pointer specifies what streaming operation is to take place on the data directly streamed

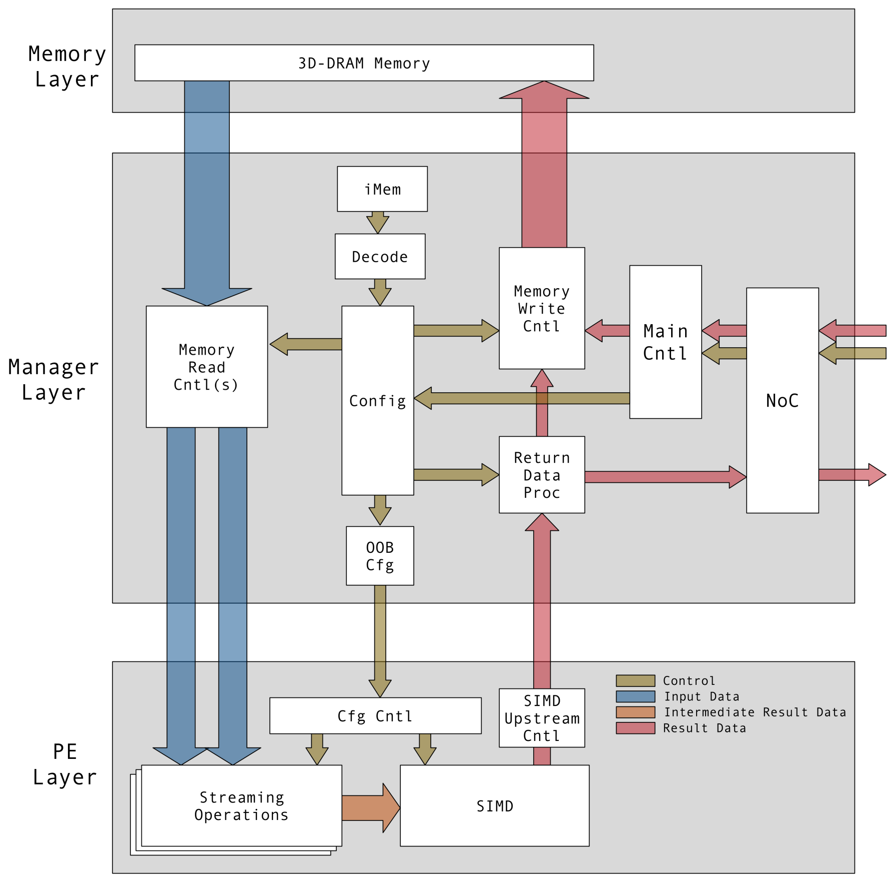


Figure 5.1 Sub-System Column (SSC) Detailed Flow Diagram

to the PE. In the baseline system, typically this would be a floating-point multiply accumulate on two arguments, the pre-synaptic neuron states and the pre-synaptic weights.

The SIMD pointer is essentially a program counter that will be invoked when the stOp result is passed to the SIMD.

Note that other types of stOp includes a NOP with a destination of local memory. This allows us to transfer block of instruction or data from the manager to the PE.

5.1.2 Argument Decode

The instruction also includes argument descriptors. These descriptors include a storage descriptor pointers that point to a storage descriptor stored in local memory that encodes where data should be read from for the one or two arguments that will be streamed from DRAM to the stOp within the PE. In the case of a AN activation calculation, there are two arguments, the pre-synaptic neuron states and the pre-synaptic weights. The read storage descriptor pointers are passed to the Memory Read Controllers (MRC). The MRCs read the actual storage descriptor from their local memory and immediately start sending read commands to the memory via a Main Memory Controller (MMC). The MMC is not shown in the diagram but essentially takes the memory read requests and converts them into the DRAM read protocol.

As soon as read data is sent back to the MRC via the MMC, that data is aligned with the downstream bus and sent to the 32 Streaming Operations inside the PE.

5.1.3 Result data Processing

The instruction also includes argument descriptors. These descriptors include a storage descriptor pointers that point to a storage descriptor stored in local memory that encodes where data should be read from for the one or two arguments that will be streamed from DRAM to the stOp within the PE. In the case of a AN activation calculation, there are two arguments, the pre-synaptic neuron states and the pre-synaptic weights. The read storage descriptor pointers are passed to the Memory

Read Controllers (MRC). The MRCs read the actual storage descriptor from their local memory and immediately start sending read commands to the memory via a Main Memory Controller (MMC). The MMC is not shown in the diagram but essentially takes the memory read requests and converts them into the DRAM read protocol.

As soon as read data is sent back to the MRC via the MMC, that data is aligned with the downstream bus and sent to the 32 Streaming Operations inside the PE.

5.1.4 Memory Write Controller

The Memory Write Controller (MWC) receives data from to sources, the NoC via the MCNTL and the RDP.

In both cases, the MWC read the actual storage descriptor from their local memory and immediately start forming data that will be written back to main memory.

When the data is formed, a write command is sent to the memory via the MMC. Again, the MMC is not shown in the diagram but takes the memory write requests along with the data and converts them into the DRAM write protocol.

The MWC can only operate on one of the two sources at any one time. However, there are four 4096-bit holding registers where data is formed prior to the write request.

The holding registers have the potential in future to allow aggregation of data from one or more operations to allow a coalesced write back to main memory.

5.2 Processing Engine

5.2.1 Configuration

A configuration controller within the PE (PE_CNTL) takes the OOB packet from the Manager and extracts the stOp and SIMD operation pointers.

The stOp pointer is used to point to a local stOp configuration memory. The memory contains the various configuration data required by the streaming operation controller (stOp_CNTL). The stOp_CNTL is not shown.

The stOp_CNTL configures the:

- Operation type
- Number of active execution lanes
- Source of the argument data, which can be downstream data from the manager or from the small local SRAM
- Destination of the result data, which can be the SIMD or the small local SRAM

The SIMD operation pointer is sent to the SIMD.

5.2.2 Streaming Operations

The streaming Operations (stOp) are designed to operate on data passed from the Manager at or near line-rate. If line-rate cannot be maintained, a flow-control mechanism is employed to slow the data from the Manager.

Once the stOp has processed the data, it passes the result to the SIMD. Note in some cases the result can be placed in local SRAM or sent to both SIMD and SRAM.

It should also be stated that while the stOp is processing the current data, the SIMD may be operating on the result of the previous operation. It is expected the SIMD will have completed the previous operation before the stOp completes the current operation, but again, if necessary a flow control mechanism between SIMD and stOP will be engaged if the SIMD is not ready.

5.2.3 SIMD

The SIMD takes the result data and performs the operation starting at the program counter (PC) indicated by the SIMD operation pointer provided by the PE_CNTL.

The stOp provides the result to the SIMD via a local register. The result is also written, in most cases to the small local SRAM.

The SIMD performs the specified operation on the data provided by the stOp.

In most cases this will be the AN activation function and in the baseline system is the Rectified Linear function (ReLu).

When the SIMD has completed its operation, it passes the result to the SIMD Upstream controller to be returned to the Manager.

5.2.4 Result Data

The SIMD Upstream Controller (SUI) takes the data and encapsulates it in an Upstream packet. Included in the packet is the tag required by the Return Data processor within the Manager.

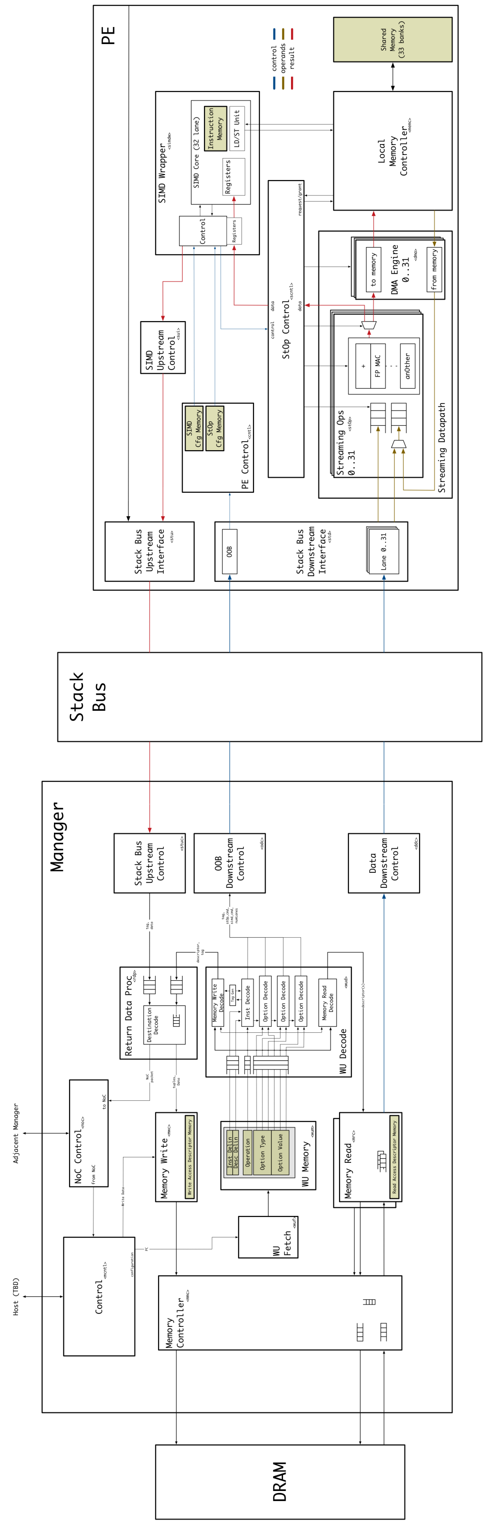


Figure 5.2 Sub-System Column (SSC) Detailed Block Diagram

CHAPTER

6

RESULTS

The objectives of this work was to design a system able to accelerate ANNs in customer facing systems implemented at the edge. This means systems that are not designed to process multiple requests of essentially the same operation. One assumption is that these systems implement disparate ANNs performing various functions. These assumptions imply that the system is not able to take advantage of local SRAM when processing the ANN. Another assumption is that the target systems will be space and power constrained. Finally, this work assumes that implementing multiple disparate ANNs cannot be implemented purely with SRAM and that DRAM is required to store the ANN parameters.

To demonstrate such a system, this work targeted 3DIC technology including 3D-DRAM. This work proposes that if a system can be purely 3DIC, the system can take advantage of the benefits of 3DIC which inlcudes reduced energy, area and additional bandwidth due to high levels of connectivity.

In addition, this work proposes that if the system is purely 3DIC, then a customized DRAM would provide a significant bandwidth boost over typical implementations using standard DRAM. This work targeted the Tezzaron DiRAM4 3D-DRAM.

To ensure the system was purely 3DIC, the area of the system Manager and Processing Engine has to stay within the physical footprint of the DRAM.

The target technology node was 28nm. This was chosen because its the technology node employed for some recent GPUs and other ASICs such as [Jou17]. The design was synthesized using an available 65nm technology node and then scaled to 28nm to demonstrate fitting within the 3DIC footprint.

The primary control and datapaths of the system have been simulated in a system verilog environment. It has been synthesized using a 65nm technology node. The design has been coded targeting a frequency of >500 MHz. Timing closure and place and route is ongoing. The system has been designed throughout to meet the timing target so minimal changes are expected to meet timing.

As mention previously (1.1), to process multiple useful sized ANNs requires a sustained bandwidth to the PE of the order of ten's of Tbit/s.

The Manager and PE have been place and routed as shown in figure 6.1. Although the design is yet to close timing, the parasitics were extracted from these layouts and simulated against a group of operations. The operations simulated were based on the expected lower and upper limits of pre-synaptic fanin. These were based on layers similar to CONV2 and FC-7 from [Kri12] and represent a pre-synaptic fanin of 225 and 4000 respectively. The simulation generated an activity file which was then used by the Synopsys® Primetime-PX™ power analysis tool to obtain power and bandwidth estimates. The DRAM accesses were captured and calculated from [Tez]. The power dissipated in the TSVs were estimated from [Liu12]. These estimates were used to estimate power dissipation for operating frequencies of 500 MHz and 700 MHz which are shown in table 6.1.

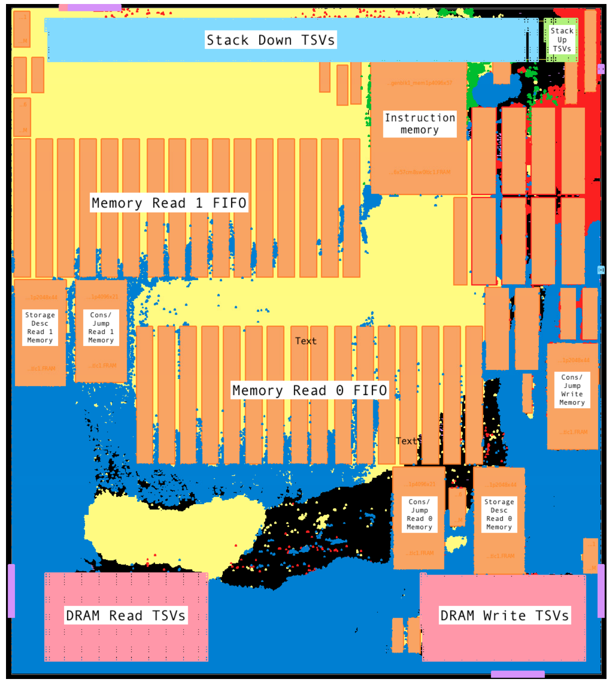
Table 6.1 Simulation-based estimates

Technology Node	Clock Frequency	Expected Power	Bus Efficiency
28nm	500 MHz	53W	$\sim$ 70 %
28nm	700 MHz	73W	$\sim$ 70 %

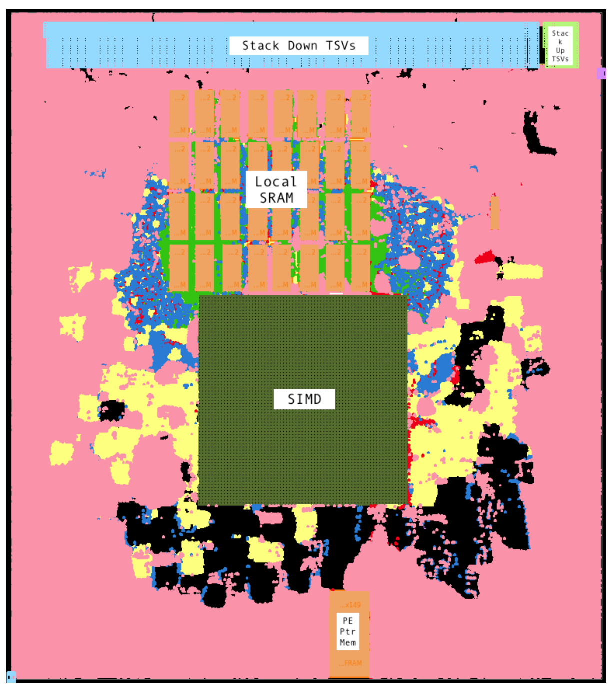
(a) Power Dissipation

Test	Downstream Stack Bus Efficiency	Average Downstream Bandwidth At Frequency 500 MHz	Average Downstream Bandwidth At Frequency 700 MHz
CONV2 [Kri12]	$\sim$ 65 %	$\sim$ 43 Tbit/s	$\sim$ 60 Tbit/s
CONV-294	$\sim$ 67 %	$\sim$ 44 Tbit/s	$\sim$ 61 Tbit/s
CONV-300	$\sim$ 73 %	$\sim$ 48 Tbit/s	$\sim$ 67 Tbit/s
CONV-500	$\sim$ 82 %	$\sim$ 54 Tbit/s	$\sim$ 75 Tbit/s
CONV-1000	$\sim$ 89 %	$\sim$ 58 Tbit/s	$\sim$ 82 Tbit/s
FC-350	$\sim$ 78 %	$\sim$ 51 Tbit/s	$\sim$ 72 Tbit/s
FC-500	$\sim$ 84 %	$\sim$ 55 Tbit/s	$\sim$ 77 Tbit/s
FC-1000	$\sim$ 91 %	$\sim$ 60 Tbit/s	$\sim$ 83 Tbit/s
FC-7 [Kri12]	$\sim$ 94 %	$\sim$ 62 Tbit/s	$\sim$ 86 Tbit/s

(b) Bus Efficiency



(a) Manager



(b) PE

Figure 6.1 Manager and PE Die layouts

BIBLIOGRAPHY

[Aba15] Abadi, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[Boj16] Bojarski, M. et al. "End to End Learning for Self-Driving Cars". *arXiv preprint arXiv:1604.07316* (2016).

[Che14] Chen, T. et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning". *ACM Sigplan Notices*. Vol. 49. 4. ACM. 2014, pp. 269–284.

[Che16] Chen, Y.-H. et al. "14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks". *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. 2016, pp. 262–263.

[Tez] *DiRAM4-64Cxx Cached Memory Subsystem*. Rev. 0.04. Tezzaron Semiconductor. 2015.

[Esm05] Esmaeilzadeh, H. et al. "NnSP: embedded neural networks stream processor". *48th Midwest Symposium on Circuits and Systems, 2005.* IEEE. 2005, pp. 223–226.

[Far11] Farabet, C. et al. "Neuflow: A runtime reconfigurable dataflow processor for vision". *Cvpr 2011 Workshops*. IEEE. 2011, pp. 109–116.

[Fra16] Franzon, P. D. et al. *Hardware Acceleration of Sparse Cognitive Algorithms*. FA8650-15-7518. AFRL. North Carolina State University. 2016.

[GH11] Galal, S. & Horowitz, M. "Energy-efficient floating-point unit design". *IEEE Transactions on Computers* **60**.7 (2011), pp. 913–922.

[Gok14] Gokhale, V. et al. "A 240 g-ops/s mobile coprocessor for deep neural networks". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2014, pp. 682–687.

[Izh04] Izhikevich, E. M. "Which model to use for cortical spiking neurons?" *IEEE Transactions on Neural Networks* **15**.5 (2004), pp. 1063–1070.

[Jou17] Jouppi, N. P. et al. "In-datacenter performance analysis of a tensor processing unit". *arXiv preprint arXiv:1704.04760* (2017).

[Kim16] Kim, D. et al. "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory". *Proceedings of ISCA*. Vol. 43. 2016.

[Kri] Krizhevsky, A. et al. *ImageNet Classification with Deep Convolutional Neural Networks*. `http://image-net.org/challenges/LSVRC/2012/supervision.pdf`. Accessed: 2016-08-30.

[Kri12] Krizhevsky, A. et al. "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[KO11] Kyo, S. & Okazaki, S. "IMAPCAR: A 100 GOPS in-vehicle vision processor based on 128 ring connected four-way VLIW processing elements". *Journal of Signal Processing Systems* **62**.1 (2011), pp. 5–16.

[Liu12] Liu, Y. et al. "A compact low-power 3D I/O in 45nm CMOS". *2012 IEEE International Solid-State Circuits Conference*. IEEE. 2012, pp. 142–144.

[Mad14] Maddison, C. J. et al. "Move evaluation in go using deep convolutional neural networks". *arXiv preprint arXiv:1412.6564* (2014).

[Mni13] Mnih, V. et al. "Playing atari with deep reinforcement learning". *arXiv preprint arXiv:1312.5602* (2013).

[Par15] Park, J. B. *Three-Dimensional DRAM Area, Timing and Energy Model*. ECE Dept., North Carolina State University, Box 7911, Raleigh. Box 7911, Raleigh, NC, 27695-7911, 2015.

[Pat14] Patti, R. "2.5 D and 3D Integration Technology Update". *Additional Papers and Presentations* **2014**.DPC (2014), pp. 1–35.

[Qiu13] Qiu, Q. et al. "A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster". *IEEE Transactions on Computers* **62**.5 (2013), pp. 886–899.

[Sch14] Schabel, J. et al. *Predictive energy-Per-Op scaling for exploring the design space*. ECE Dept., North Carolina State University, Box 7911, Raleigh. Box 7911, Raleigh, NC, 27695-7911, 2014.

[Tai14] Taigman, Y. et al. "DeepFace: Closing the Gap to Human-Level Performance in Face Verification". *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[Tec] Techpowerup. *NVidia Tesla K20c*. `https://www.techpowerup.com/gpudb/564/tesla-k20c`. Accessed: 2016-09-08.

[Tsa08] Tsai, Y.-F. et al. "Design space exploration for 3-D cache". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **16**.4 (2008), pp. 444–455.

APPENDICES