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.)

This dissertation explores employing a Three-Dimensional Integrated Circuit (3DIC) in the acceleration of Artificial Neural Networks (ANNs) for systems deployed in customer facing applications. Assuming ANNs fulfill their potential, it is this works belief that these systems will employ ANNs for various functions, such as engine monitoring, anomaly detection, navigation etc. and that the various system functions are implemented with a set of disparate ANNs. A further assumption is that these customer facing systems may not have access to cloud servers or the cloud servers do not provide the necessary turn-around time for processing the ANN.

Although ANNs have been known of 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.

Artificial Neurons (ANes) 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 ANes that simply encode the spiking behavior in the form of a number or rate.

The ANNs that have demonstrated most efficacy are a family of neural networks that can be described as Deep Neural Networks (DNNs). These DNNs are created by cascading layers of rate-based ANes to form a large, layered ANN employing ten's of thousands or more of ANes.

Considering the storage required for the input and the ANN parameters, the storage requirements result in gigabytes of memory. When these ANNs are required to be solved in fractions of a second, the processing and memory bandwidth becomes prohibitive.

Unfortunately, to achieve a high performance, existing implementations rely on processing a batch of inputs, such as processing a batch of images or voice recordings which all use the same ANN

or by employing a sub-family of DNNs, known as Convolutional Neural Networks (CNNs), which reuse portions of the ANN parameters. These techniques allow these implementations to hold and reuse data in fast local Static Random Access Memory (SRAM). With this works target application, the assumption is there is little opportunity for batch processing or reuse therefore data must be drawn constantly from main memory, which generally is Dynamic Random Access Memory (DRAM).

One area of integrated circuit technology that hasnt been widely used in ANNs is 3DICs. 3DICs have the potential to increase connectivity, and thus bandwidth and keep power dissipation to within acceptable levels.

This work demonstrates how a customized 3DIC DRAM can be combined with application-specific layers to produce a system meeting the required level of performance in systems with multiple instances of disparate 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 high school he took a job in a local electronic engineering firm under a vocational program. While working on the manufacturing floor and 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. The journey to the "white coat" 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.

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

1.1 Overview

Machine Learning in the form of Deep Neural Networks (DNNs) have gained traction over the last few years. They get good press in applications such as image recognition and speech recognition. DNNs are constructed from a basic building block, the Artificial Neuron (ANe). With popular DNNs, the Artificial Neural Network (ANN) is often formed from tens of layers with each layer containing many ANes. In most cases, these layers are processed in a feed-forward manner with one layer being the inputs to the next layer. Therefore, useful DNNs often require hundreds of thousands of ANes and within the network, each ANe can have hundreds, even thousands of feeder or pre-synaptic ANes.

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 have significant memory requirements to store the connection weights (parameters) therefore requiring Dynamic Random Access memory (DRAM) to store the ANe parameters.

There have been many successful attempts to accelerate ANNs, but in most cases the focus is on a subset of the DNN known as the Convolutional Neural network (CNN). CNNs assume a significant amount of reuse of the weights connecting ANes and thus they can take advantage of local memory (SRAM).

Much of the ASIC and ASIP ANN research has focused on taking advantage of the performance and ease of use of SRAM. These implementations can be shown to be effective with specific ANN architectures, such as CNNs where the ANN parameters can be stored in SRAM in a cache-like architecture avoiding constant accessing of the "slower" DRAM. In addition, to achieve a high performance, these rely on processing a batch of inputs, such as processing a batch of images or voice recordings using the same ANN.

The work in this paper considers "edge" applications that require the processing of a disparate set of useful sized ANNs. The work assumes that the application system is utilizing ANNs for the processing of various sub-systems, such as navigation, engine monitoring etc.. This work also does not assume the ANN is specifically a CNN but a DNN where there may not be opportunities to store and reuse portions of the ANN in SRAM. A further assumption is that the target edge devices do not include opportunities to perform batch processing. Under these circumstance, when these implemenations need to contantly load ANN parameters directly from main memory, the performance is constrained to the DRAM interface bandwidth. Therefore, the performance of SRAM-based ASIC/ASIP implementations are severely degraded to the point of being unacceptable.

This work uses the DRAM as the primary processing storage and employs minimal SRAM for the processing of the ANe. In addition, the work considers 3D integrated circuit technology and a custom 3D-DRAM. By employing 3DIC technology, this work takes advantage of the reduced energy and area and increased connectivity and bandwidth to allow the DRAM to be employed efficiently without the

need for local SRAM. This work demonstrates that a 3DIC system based on a customized 3D-DRAM could be used in edge applications requiring at or near real-time performance for systems running multiple ANNs.

It should be noted that this work does not design a custom 3D-DRAM but answers the question "if such a device were available, can we employ it within a useful ANN system".

1.2 Abbreviations

Acronyms

3D Three-Dimensional

3D-DRAM Three-Dimensional Dynamic Random Access Memory

3DIC Three-Dimensional Integrated Circuit

ANe Artificial Neuron

ANN Artificial Neural Network

ASIC Application-Specific Integrated Circuit

ASIP Application-Specific Instruction-set Processor

binary32 Single-precision floating-point format

CNN Convolutional Neural Network

CPU central processing unit

DDR Double Data Rate

DiRAM4 Dis-Integrated 3D DRAM

DNN Deep Neural Network

DRAM Dynamic Random Access Memory

EOM end of message

ESD Electrostatic discharge

FIFO first-in first-out queue

GPU graphics processing unit

HBM High Bandwidth Memory

HMC Hybrid Memory Cube

IC Integrated Circuit

IP Intellectual property

KGD Known Good Die

LSTM Long Short-term memory

MAC multiply-accumulate

MOM middle of message

NoC Network-on-Chip

OOB Out of Band

PC Program Counter

PE Processing Engine

QDR Quad data rate

ReLu Rectified Linear Unit

ROI region-of-interest

SIMD Single-Instruction Multiple-Data

SoC System-on-Chip

SOM start of message

SRAM Static Random Access Memory

SSC Sub-System Column

StOp Streaming Operation block

TDP total design power

TSV Through-Silicon Via

CHAPTER

2

ARTIFICIAL NEURAL NETWORKS

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

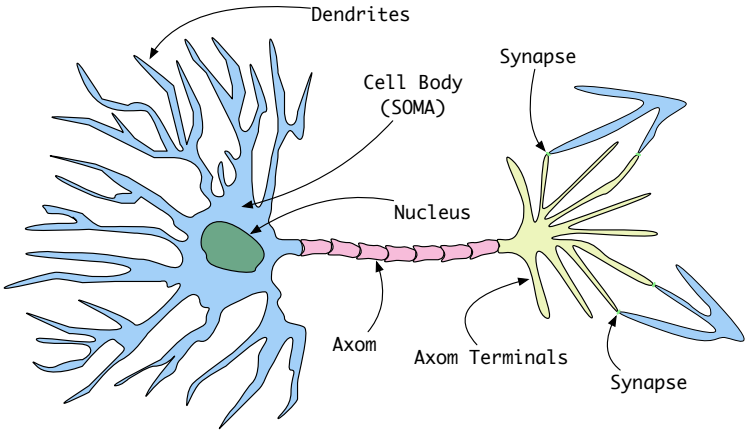


Figure 2.1 Artists impression of a Mammalian Neuron

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 2.1.

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 discuss next.

2.1 ANN Overview

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. 2.4a)

The Rate-based neuron model is characterized by:

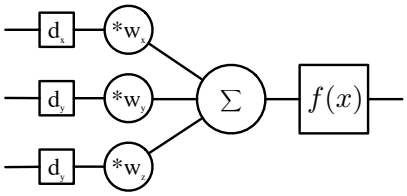
- Connections between neurons have only a strength
  - The pre-synaptic neuron output is multiplied by the connection weight
- The weighted inputs from all pre-synaptic neurons are accumulated
- The accumulated inputs drives an activation function
  - the activation function $f(x)$ is a non-linear function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

    examples are (see Fig. 2.3):

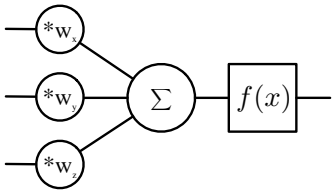
    · sigmoid

    · rectified linear unit

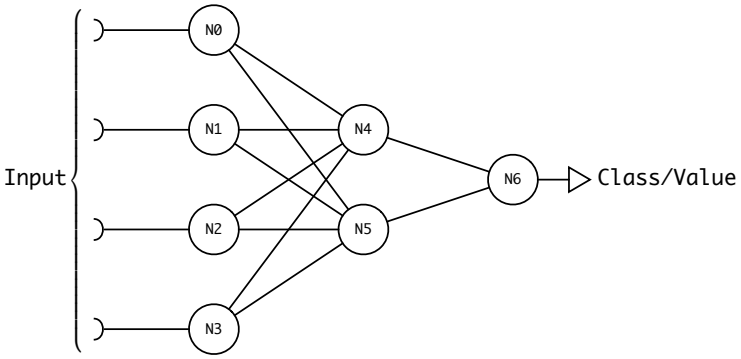
To emulate complex behavior, the artifical neurons are connected in networks, typically with layers of sub-networks which are in affect separated by the non-linear activation function. Examples of both rate-based and spiking artificial neural networks can be seen in Fig. 2.5a and Fig. 2.5b respectively. Typically neural networks process in a feed-forward fashion. Considering Fig. 2.2, 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

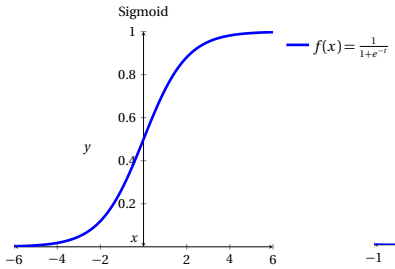


(b) Rate-Based Model

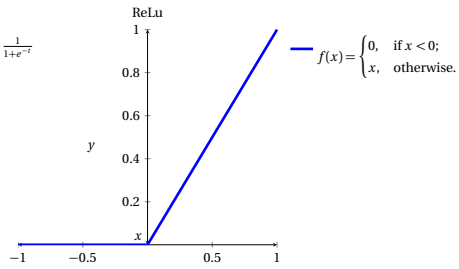


(c) Network of Artificial Neurons

Figure 2.2 Artificial Neurons and Network



(a) Sigmoid

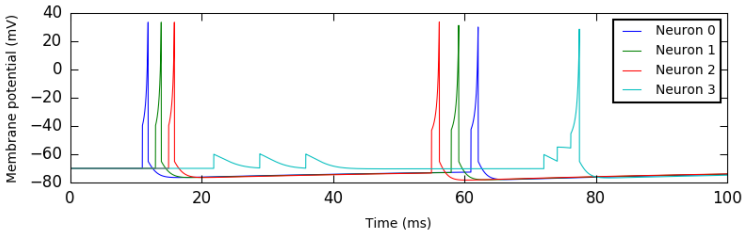


(b) Relu

Figure 2.3 Example Rated-Based Model Activation functions

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

(a) Izhikevich Model[Izh04]



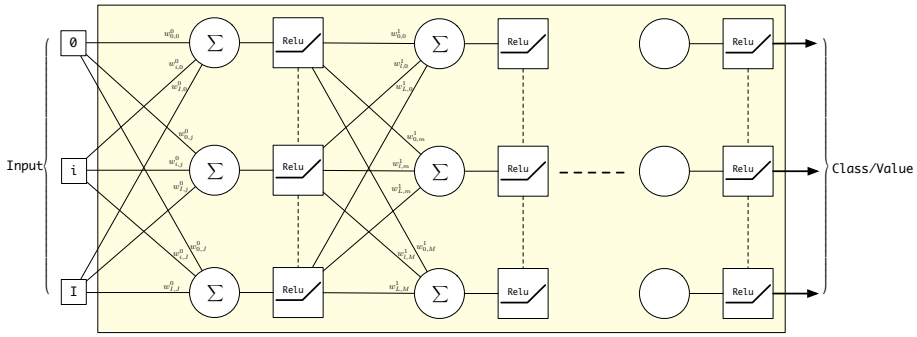
(b) Izhikevich Model Simulation [Izh04][CH06]

Figure 2.4 Example Spiking Activation Function Model

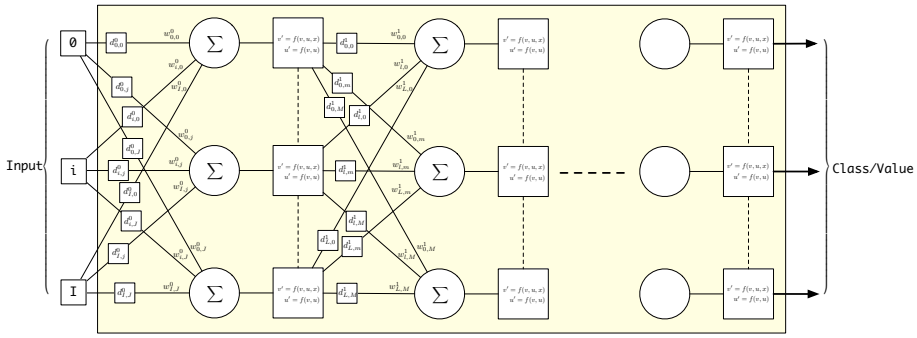
recursion where for example neurons N0 through N4 are not only influenced by the input, but also by themselves. Many ANNs operate only in feed-forward fashion but some popular ANNs, such as Long short-term memory (LSTM), employ recursion.

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 ANes with each layer containing many ANes. 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 ANes and within the network, each ANe can have hundreds, even thousands of feeder or pre-synaptic ANes. 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 2.5 Layered 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.

This work does not address the training of these rate-based ANNs, the training is mostly performed offline. This work is addressing the inference of ANNs. During inference, the most computationally intensive operation is the multiply accumulate associated with the ANe activation, which can involve hundreds or thousands of multiply-accumulates. The ANe activation calculation for the rate-based ANe in figure 2.2b is shown in equation (2.1).

$$\text{ANe Activation} \quad A = f\left(\sum_{n=0}^{C_n} W_n \cdot A_n\right) \tag{2.1}$$

$\mathbf{C_p}$ is the number of pre-synaptic connections

$\mathbf{W_p}$ is the weight of a connection

$\mathbf{A_p}$ is the activation value of the pre-synaptic neuron

and $f(x)$ is the activation function such as ReLu

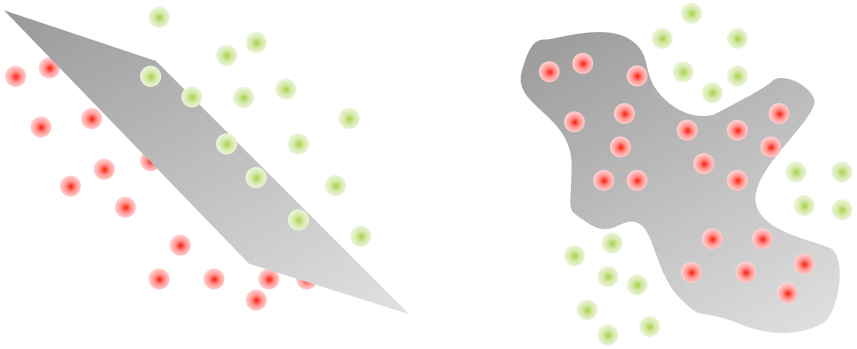
2.2 ANN Layers

In figures 2.2c and 2.5a, the ANN is shown to be constructed using layers of ANes. It has long been known that a single layer of ANes can be used linearly partition an n-dimensional input, as shown in figure 2.6a. However, if a more complex partition is required, this cannot be achieved using a single layer of ANes. A higher order classification, as shown in figure 2.6a can only be achieved using multiple layers of ANes. In addition, to ensure the multiple layers cannot be mathematically collapsed into a single layer, the activation function $f(x)$, as shown in figure 2.2b must be a non-linear function.

2.2.1 Deep Neural Networks

As mentioned, a single layer of neurons can be used as a linear classifier as long as the classes can be separated using a linear function. Even some simple cases cannot be linearly separated, an example often used is an exclusive-OR gate.

Even with a layered ANN, the final output comes from a single layer. To allow this final layer to linearly separate classes, the original input needs to be transformed into a space where the classes can be linearly separated. Deep Neural Network (DNN) are ANNs that incorporate many layers of ANes, often which are often up to tens of layers deep. The additional layers are incorporated to translate the



(a) Linear Classification using a single layer

(b) Complex Classification requires multiple layers

Figure 2.6 Classification using ANNs layers

space of the input so the various classes being identified can be separated using linear classifiers in the later layers.

Recently, an example of a DNN known as a Convolutional Neural Network (CNN) demonstrated high levels of efficacy when used to classify objects in images [Kri12]. These CNNs use the early layers to identify low-level features and later layers are used combine these features into yet more higher-level features. Finally, the combination of high-level features are used to identify the required classes. This layering is shown in Fig. 2.7.

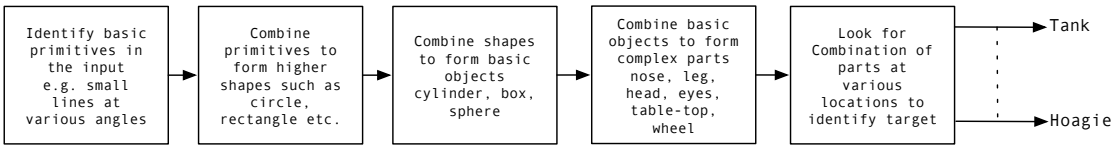


Figure 2.7 Deep network showing feature layers

In figure 2.7, the final layer is often a fully connected linear classifier with the output representing the probability of a particular class being present in the image. In practice, these DNNs can be used as classifiers or as function approximators.

2.2.2 Feature Layers

For the most part, different ANNs are characterized by how the ANes are interconnected and the activation function employed.

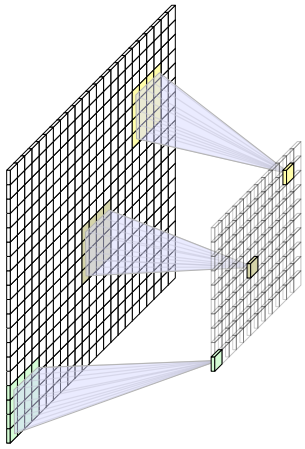
The typical DNN layers are processed in a feed-forward fashion where the pre-synaptic ANes are formed from ANes in the previous layer. There are some types of DNN which also include recursive connections where the the pre-synaptic ANes include ANes from the current layer. A popular recursive DNN is Long Short-term memory (LSTM). Although this work does not preclude supporting LSTM in the future, the focus of this work is on the feed-forward type DNN.

As described in 2.2.1, a DNN layer transforms the previous layer with each higher layer providing a coarser grained transformation. This is best seen in image recognition application were the early layers identify low level shapes or features, such as angled lines. The following layers are used to identify higher order shapes such as circles, blocks etc. Although the features detected during the image recognition application are somewhat intuitive, it is believed that in less intuitive applications the DNN performs a similar fine to coarse feature extraction.

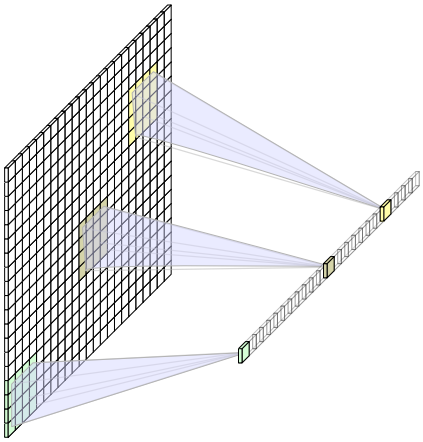
The connections between layers can be locally-connected or fully-connected. With locally-connected layers as shown in figure 2.8, a layers pre-synaptic ANes are formed from regions of the previous layer. With fully-connected layers as shown in figure 2.9, a layers pre-synaptic ANes are formed from all ANes in the previous layer. In many cases, a DNN is constructed with lower layers being locally-connected and higher layers being fully connected.

In early uses of locally-connected ANNs, the first layers weights were often hand-generated, an example being Gabor filters. With automatically trained ANNs, the feature detectors at each layer are often created during training. Some contrived examples of locally-connected feature detectors are shown in figure 2.10.

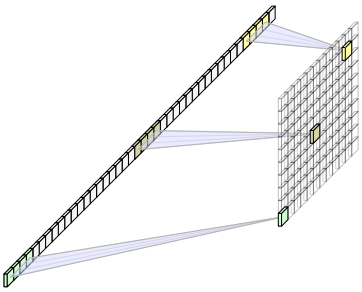
The pre-synaptic ANes of a locally-connected ANe are formed from particular region-of-interest (ROI) of the previous layer using the weights from a feature filter. Another locally-connected ANe



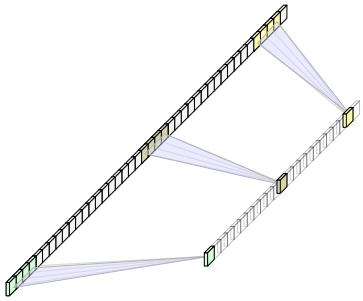
(a) Local Array to Array



(b) Local Array to Linear



(c) Local Linear to Array

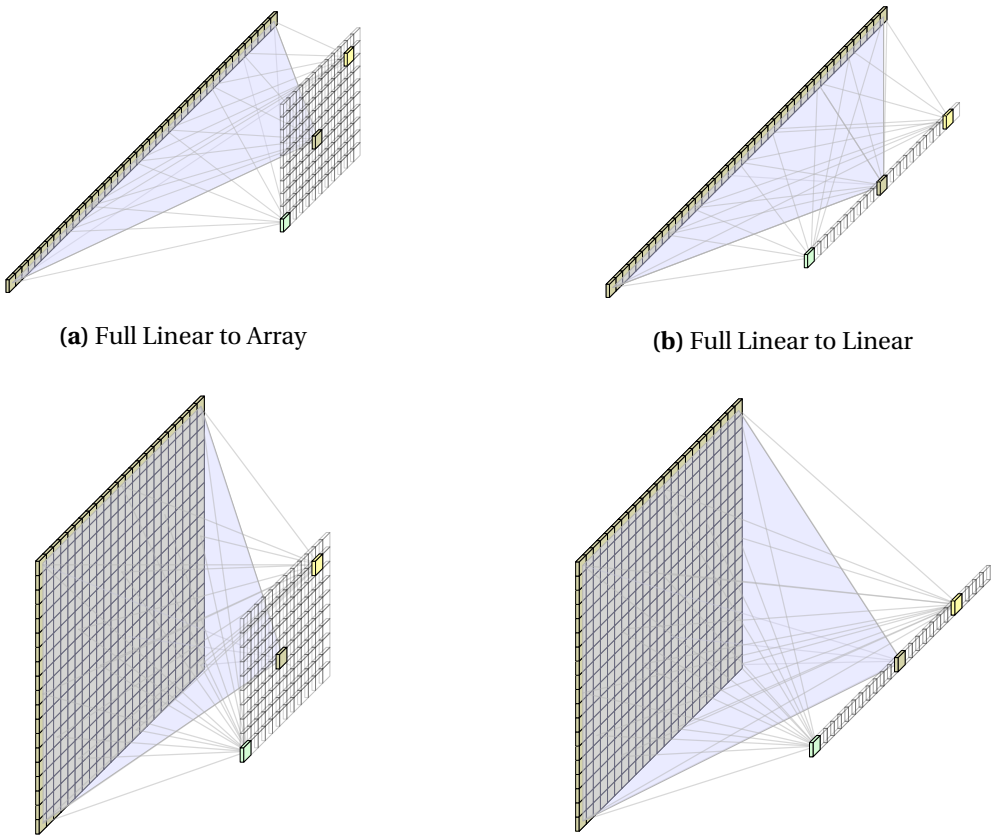


(d) Local Linear to Linear

Figure 2.8 Locally Connected Layer to Layer Connection Types

may use the same ROI but employs a different filter. In practice, for a particular ROI, a number of feature filters are employed resulting in a number of ANes being associated with the same ROI in the previous layer. To reiterate, these feature filters all operate on the same ROI. A different ROI will result in another group of ANes all using their own feature filters. The resulting locally-connected layer becomes a Three-Dimensional (3D) layer with its X-Y coordinates representing a reference to a particular ROI and the Z-dimension representing the various filters applied to that ROI. A example 3D locally-connected layer can be seen in figure 2.11.

So these locally-connected layers have multiple filters applied to the same ROI and the next layer



(a) Full Linear to Array

(b) Full Linear to Linear

(c) Full Array to Array

(d) Full Array to Linear

Figure 2.9 Fully Connected Layer to Layer Connection Types

becomes a 3D array with the Z-axis representing the features. The number of feature filters applied at each layer can be tens to hundreds of filters. The filters employed in a layer following one of these 3D locally-connected layers are themselves 3D. With tens to hundreads of features in the previous layer the number of weights associated with each filter is usually hundreds to thousands of weights.

The feature filters employed in the locally-connected layers can be unique to the regions of the previous layer or the same filters can be employed across the entire previous layer. In the case of employing the same filters across the entire input layer the ANN is known as a Convolutional Neural Network (CNN). The CNNs are examples of ANNs which can take advantage of reuse described in

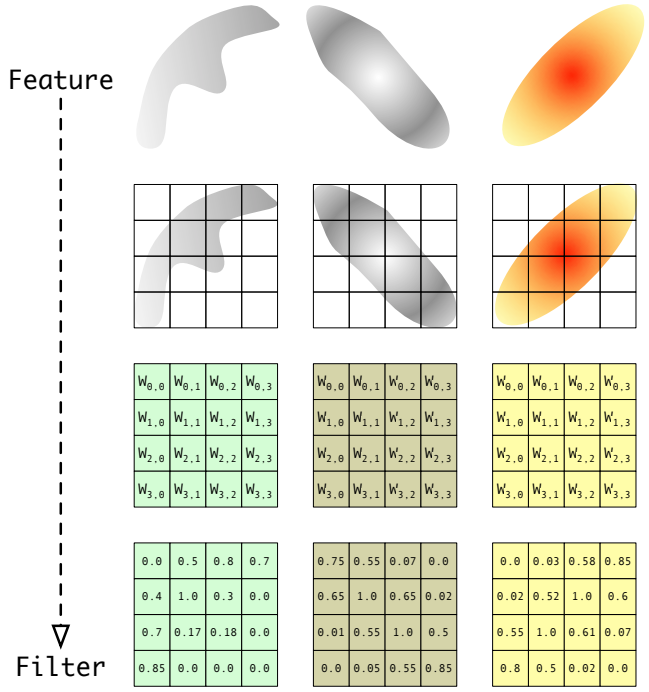


Figure 2.10 Features and locally-connected filters (kernels)

chapter 1. These CNNs can store the filter parameters in local SRAM and construct an entire feature plane. These CNNs are considered a subset of the generic case of DNN. This work considers the more general DNN case and supports acceleration of generic DNNs which includes CNNs.

An example of a DNN can be seen in figure 2.12b with the layer configurations shown in table 2.12a. A CNN similar to this has demonstrated high levels of efficacy in image recognition applications. **Therefore, this work will use the parameters from the table shown in figure 2.12a as a template for a baseline ANN for estimating the storage and processing requirements and the range of pre-synaptic fanins.**

To approach the capabilities observed in human behavior, such as object recognition ANNs have become very large. The example shown in figure 2.12, which is based on the work from [Kri12], has hundreds of thousands of ANes and hundreds of millions of connection weights (see table 2.12a).

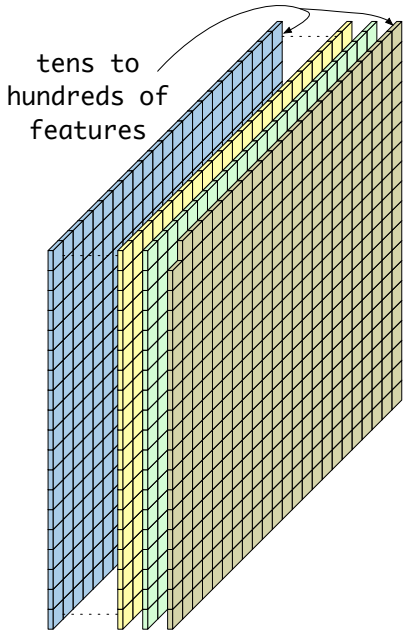


Figure 2.11 Single Layer constructed from 3D layers of Features

These ANNs utilize these hundreds of thousands of ANes to implement what a human would consider a relatively straightforward task. For example, a "useful" ANN similar to that described in [Kri12] which was used to recognize up to 1000 different object classes, has a network size of approximately 650,000 ANes and 630 million synaptic connections [Kri].

The increased performance of ANNs over classical methods in image recognition and voice recognition might suggest that ANNs will out-perform operations performed in other applications.

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 figure 2.12 and [Kri12], these implementations will be processing multiple large ANNs at or near real-time.

2.3 ANN Processing

Considering the storage required for the input, the ANe states and most significantly the weights for connections, 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 assumes that any useful ANN will be similar to that shown in figure 2.12 which utilizes $> 900 \times 10^3$ANes and $\approx 200 \times 10^6$ parameters.

When it comes to estimating storage requirements for ANNs there is a lot of debate regarding the precision of number format for the parameters. There has been work on the impact of changing the precision of the number format employed during training and inference. These formats can vary between eight bit fixed point to 64-bit double precision. However, for the baseline requirements this work assumes 32-bit single-precision floating point.

Therefore, assuming an ANN similar to that shown in table 2.12a with 775×10^3ANes and an average fanin to each ANe of 1650, a system employing 10 ANNs for various disparate functions and an average processing time of 16 ms suggests a average bandwidth of 32 Tbit/s (see equation 2.2).

$$
\begin{aligned}
\text{Maximum Bandwidth} &= \sum_{\mathbf{n}=0}^{\mathbf{N_n}} \Big(\frac{\overline{\mathbf{N}}_\mathbf{a} \cdot \overline{\mathbf{C}}_\mathbf{p} \cdot \mathbf{b_w}}{\overline{\mathbf{T}}_\mathbf{p}} \Big) \text{bit/s} \\
&= \sum_{\mathbf{n}=0}^{9} \Big(\frac{772 \times 10^3 \cdot 1.65 \times 10^3 \cdot (32+1)}{16 \times 10^{-3}} \Big) \\
&= \sum_{\mathbf{n}=0}^{9} 2.63\,\text{Tbit/s} \\
&\approx 26\,\text{Tbit/s} \qquad (2.2)
\end{aligned}
$$

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

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

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

$\mathbf{b_w}$ is the number of bits per parameter

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

Note: assumes ROI streamed to all lanes

When implementing ANNs, the memory requirements are also significant. The storage is required for the input, the ANe states and most significantly the parameters for each of the ANes pre-synaptic connections. For the case shown in table 2.12, there are 202×10^6 parameters requiring 0.81 GB and 772×10^3 ANes requiring 3.88 MB storage. The storage required for 10 ANNs is of the order of 8.0 GB (2.3).

$$
\begin{aligned}
\text{ANN Memory} &= \sum_{\mathbf{n}=0}^{\mathbf{N_n}} \left((\overline{\mathbf{N}}_{\mathbf{p}} + \overline{\mathbf{N}}_{\mathbf{a}}) \cdot \mathbf{b_w} \right) \text{Gbit} \\
&= \sum_{\mathbf{n}=0}^{9} \left(\left(202 \times 10^6 + 772 \times 10^3\right) \cdot 32 \right. \\
&= \sum_{\mathbf{n}=0}^{9} 6.49\,\text{Gbit} \\
&= 64.9\,\text{Gbit} \equiv 8.1\,\text{GB}
\end{aligned} \tag{2.3}
$$

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

$\mathbf{N}_a$ is the number of ANEs per ANN

and $\mathbf{b_w}$ is the number of bits per parameter

The approximate system bandwidth and storage requirements are shown in table 2.1.

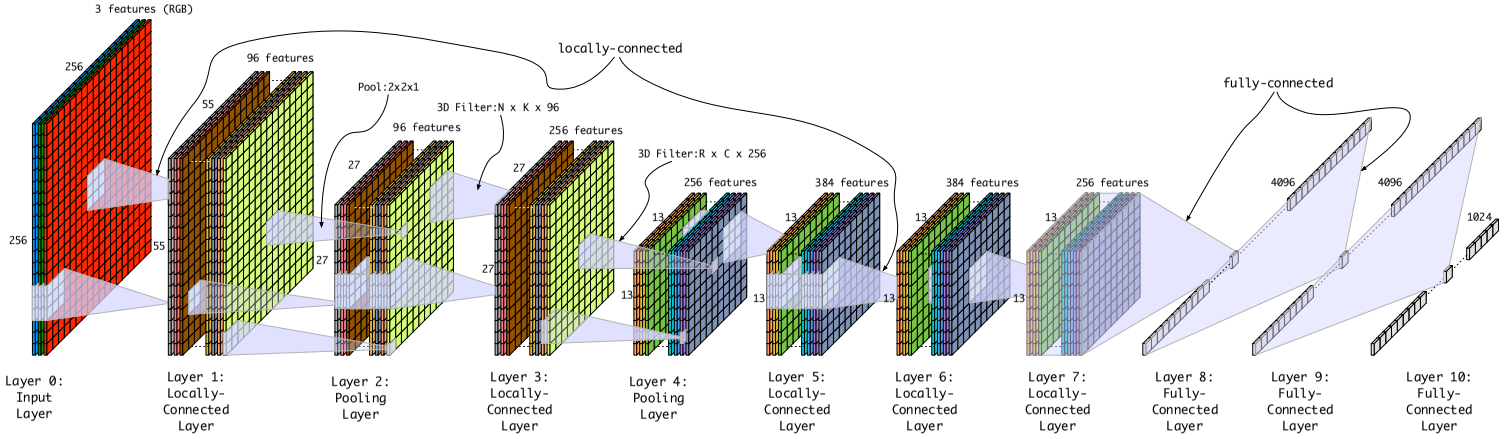
Table 2.1 Estimated System Bandwidth and Storage Design Requirements

Parameter	Value
Bandwidth	26 Tbit/s
Storage	8.0 GB

Given the bandwidth and storage requirements shown in table 2.1, 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.**"

		Layers											
		1	2	3	4	5	6	7	8	9	10	11	
Type		Input	Locally	Pooling	Locally	Pooling	Locally	Locally	Locally	Fully	Fully	Fully	
Dimensions	X	256	55	27	27	13	13	13	13	4096	4096	1024	
	Y	256	55	27	27	13	13	13	13	1	1	1	
	Z	3	96	96	256	256	384	384	256	1	1	1	
Filter Dimensions	X	na	11	2	5	2	3	3	3	13	4096	4096	
	Y	na	11	2	5	2	3	3	3	13	1	1	
	Z	na	3	1	96	1	256	384	384	256	1	1	
Stride		na	4	2	2	2	1	1	1	na	na	na	Aggregate
Pre-synaptic Fanin		na	363	4	2400	4	2304	3456	3456	43264	4096	4096	1650
Number of ANe		196608	290400	69984	186624	43264	64896	64896	43264	4096	4096	1024	772544
Number of Weights		na	34848	na	614400	na	884736	1327104	884736	177209344	16777216	4194304	2.02×10^8

(a) Baseline ANN Layer Configuration [Kri12]



(b) DNN showing layer order [Kri12]

Figure 2.12 Baseline DNN [Kri12]

CHAPTER

3

MOTIVATION

3.1 The Problem

As mentioned in chapter 1, this work focuses on edge applications employing disparate ANNs and therefore assumes there are limited opportunities for both weight reuse and batch processing.

Given the storage requirements shown in table 2.1, it is generally accepted that DRAM is required to store the ANN parameters.

When considering systems that will employ multiple DNNs simultaneously suggests that these edge systems will require usable memory bandwidth of the order of 10s of Tbit/s (2.2).

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

3.1.1 Why not SRAM?

Why is it that much of the ASIC and ASIP ANN research employs SRAM as an intermediate store?

In practice there are benefits if the processing elements can operate out of SRAM. Certainly good performance and potentially low power.

Perhaps also because its easy to use?

When compared to DRAM, SRAM has low latency. Also, the DRAM access protocol is much more complicated to implement than SRAM.

Given that DRAM is used for the main memory storage, having the processing elements operate out of SRAM requires that the high cost of transferring data from the DRAM to the SRAM be absorbed by using that data multiple times or "reused".

But using SRAM for intermediate storage makes assumptions on the type of ANNs that can be supported and the application in which the ANN is being deployed. The primary requirement of the type of ANN and the deployed application to allow effective use of SRAM is "reuse", so once parameters are transferred and stored in SRAM, these parameters can 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. A prime example is CNNs, where the connection weights are reused. In CNNs, common feature filters are 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 different filters may be used for different ROIs some filter reuse may be available. Another form of reuse is in cloud applications or in training where 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 ANN processors, they are dominated by the silicon area of the SRAM. Because of the relatively large area required for SRAM, companies attempt to create custom SRAMs to minimize the area impact.

So ASIC and ASIP ANN implementations that target applications that have considerable weight

reuse and/or batch processing opportunities can effectively use SRAM as an intermediate store.

But to reiterate, this work assumes the target application have limited or no opportunities for weight reuse or batch processing.

3.2 Alternatives

3.2.1 Graphics processing units (GPUs)

Some might suggest the requirements of these applications would be satisified by employing multiple GPUs. In practice, GPUs are used to implement large ANNs and in some ANN architectures, such as CNNs, they are quite effective. However, we should not forget they are a) not optimized purely for ANN processing, b) are restricted by available SRAM and c) 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 technology. For example, a 2.5D solution employing High bandwidth Memory (HBM) would be limited to a maximum raw bandwith of the order of 6 Tbit/s [Nvi]. Also, it has proven very difficult, if not impossible to take advantage of the available memory bandwidth [Far11] [Aba15].

A solution could employ multiple devices, but there would be significant power and real-estate issues. The typical high performance GPU consumes between the order of 100 W and 200 W. A multiple GPU implementation would have a high real-estate impact and a system power approaching a 1 kW.

Overall GPUs have limited suitability to meet this works target application requirements.

3.2.2 ASICs/ASIPs

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 (SRAM). These implementations can be shown to be effective with specific ANN architectures (e.g. CNN), server applications or the

"toy examples" but when a system requires multiple disparate ANNs in an edge application, **existing implementations do not provide the required flexibility, storage capacity and deterministic performance**.

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 ANNs, but of all the papers at ISCA 2016 on hardware accelerators for ANNs, alas, all nine papers looked at CNNs, and only two mentioned other ANNs. 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.

Considering this work focuses on edge applications employing disparate ANNs and assumes both weight reuse and batch processing do not apply, regardless of how implementations employ SRAM as an intermediate store, **DRAM bandwidth is the bottleneck**.

3.3 The Solution

It is generally accepted that realistically, DRAM is required to meet the main storage requirements of useful sized ANNs.

This work also believes that to support all types of disparate ANNs, the system needs to be able to operate directly from the DRAM.

If an implementation can ensure the DRAM bandwidth can meet the system requirements, why use SRAM as an intermediate memory and waste the significant silicon area they consume.

The question becomes, can an implementation employ DRAM with minimal SRAM and meet the system requirements?

This works implementation operates directly out of DRAM, but not just DRAM, Three-Dimensional Dynamic Random Access Memory (3D-DRAM). This work has designed a system that can stay within the physical footprint of the 3D-DRAM and thus can leverage the benefits of 3DIC. The benefits of 3DIC, which are reviewed in chapter 4 include reduced energy, reduced area and increased connectivity and bandwidth.

Given the problem description the primary design considerations that drove the architecture of this work are :

- DRAM is required for storage of ANN parameters
- Target applications are unable to take advantage of memory reuse opportunities and therefore not able to achieve high performance using local SRAM
- Target application will likely apply many disparate ANNs to perform various system functions
- Target application will have space and power limitations

When performing inference in ANNs, the computational hotspot is the ANe pre-synaptic summation shown in figure 2.2b and equation (2.1). This ANe summation involves hundreds or thousands of multiply-accumulates of the pre-synaptic ANe activations and corresponding connection weights. In this work, the ANe activations and weights are stored in DRAM with minimal local SRAM. Therefore, because of the complex access protocol associated with DRAM, one of the main objectives is to demonstrate the 3D-DRAM can be accessed while maintaining the required average bandwidth to the processing elements.

The system has to process thousands of ANes concurrently and do this with minimal unused bus cycles. Therefore, the system must decode instructions, configure the various functions, pre-fetch and pipeline DRAM data and perform the actual activation calculation.

To maximize the processing bandwidth, these operations are all performed concurrently enabling this work to demonstrate the ability to meet and exceed the required processing bandwidth as shown in equation (2.2).

3.3.1 Novelty

The novelty of this work includes:

- A system that can simultaneously process multiple disparate ANNs at or near real-time
  - with low power and real-estate demands
- A custom 3D-DRAM providing a ~64X bandwidth benefit compared to standard 3D-DRAM
  - the 3D-DRAM could be employed in other applications
- A system that employs pure 3DIC technology
  - providing power and performance benefits of remaining within a 3DIC stack
- Custom instructions and data structures that facilitate operating directly out of 3D-DRAM
  - maximizing processing bandwidth by ensuring effective use of the 3D-DRAM
  - instruction format allow system functions to operate concurrently

3.3.2 Summary

This research explores a 3DIC solution using a custom organized 3D-DRAM 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 disparate ANNs whilst staying within acceptable area and power limits and will provide greater than an order of magnitude benefit over comparable solutions.

An overview of 3DIC technology is given in chapter 4. An overview on the pros and cons of DRAM and SRAM along with some proposed DRAM customizations are given in chapter 5. Some state-of-the-art implementations are reviewed in chapter 6. An overview of the proposed system is described in chapter 7 with more details in chapter 9. An overview of the instruction architecture is given in chapter 8. Simulation results are shown in chapter 10. The conclusion and further work are discussed in chapter 11.

CHAPTER

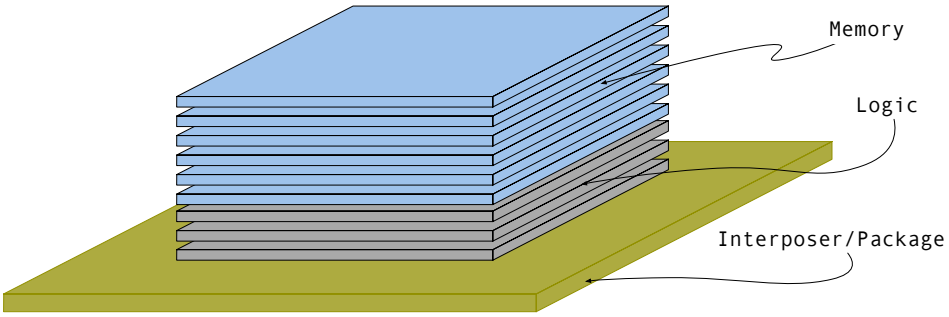
4

THREE-DIMENSIONAL INTEGRATED CIRCUITS (3DIC)

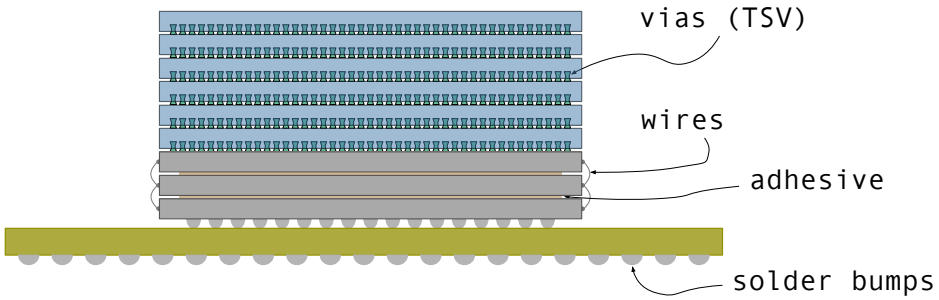
Over the last couple of decades, the ever shrinking world of Integrated Circuits (ICs) has enabled the introduction of devices society takes for granted such as personal computing and cell phones.

As IC technology has shrunk, design complexity has grown to take advantage where hundreds of millions of transistors are placed on a typical IC. These ICs have evolved from performing small functions to becoming systems on a chip.

However, at some point, an IC has to interface with another function and often that communication involves the moving of data to and from memory and the increase in complexity often drives a need for higher memory data bandwidth.



(a) Different Die types stacked mounted on an interposer/package substrate



(b) Connection Types

Figure 4.1 3DIC Stack of Die

The IC complexity has been doubling approximately every two years but the external interfaces are restricted by physical limitations. Within the System-on-Chip (SoC), the designer can take advantage of very wide interfaces, often thousands of bits wide to increase bandwidth, but when data is moved to off-chip memory, the widest buses are usually hundreds of bits wide.

One way to avoid this limitation is to employ 3DIC (see figure 4.1). The advantages of 3DIC are well understood. reducing the amount of off-chip communication increases bandwidth and reduces power. The power reduction comes from not having to drive the relatively high capacitance inputs and outputs.

4.1 Pros and Cons

Taking advantage of 3DIC means stacking die on top of one another and making connections directly between the die. These connections can be in the form of wire made at the edges of the die or using vias buried in the die itself.

Below is a summary the benefits of 3DIC :

- Reduced Power
  - mainly from not having to drive external outputs and receiving external inputs
- Increased Connectivity
  - maintaining very wides buses through the SoC increases bandwidth
- Ability to mix heterogeneous technology
  - Mixed Analog/Digital
  - Mixing memory technology and logic technology
- Increase density and mitigation against the slowing of Moores Law
  - using the vertical domain to increase perceived transistor per mm^2
- Potentially lower costs by combining simpler die rather than build a large die
  - yield benefits from combining higher yield die
- Possibility of novel architectures [Kim16b]

Some disadvantages of 3DIC are:

- Reliability

- Cost
  - being a relatively new technology it is still expensive
  - TSV technology is still unreliable

There is still some reluctance to fully embrace 3DIC but undoubtly the various barriers will be broken down.

The ability to mix heterogenous technology is of particular interest to this works target application because mixing technology targeted toward DRAM with cmos logic technology is a heterogenous mix of which this work takes advantage.

In [ITR15] there are four definitions of 3DIC interconnects:

- 3D-Wafer level package [ITR15]
  - In this case, different die are stacked and then connected using traditional bond bumps and/or bond wires at the periphery of the chip.
  - This technique provides better transistor density compared to traditional 2D-IC with improvements in interconnect density.
- 3D-Stacked SoC [ITR15]
  - In this case, different die are stacked and then connected using TSVs. The TSVs connect the dies to intermediate metal layers known as global metal layers. This allows the indivial die to maintain a high level of functionality and thus is similar to connecting functional building blocks meaning the individual die are likely to be significant functional pieces of Intellectual property (IP).
  - using TSVs provides a medium level of interconnect
- 3D-Stack IC [ITR15]

  - In this case, different die are stacked and then connected using TSVs. The TSVs connect the dies to intermediate higher metal layers known as global metal layers. This infers the individual die are not large functioning pieces of IP.
  - using TSVs provides a high level of interconnect

- 3D-Integrated Circuit [ITR15]
  - In this case there are not multiple dies. Instead the additional silicon layers are deposited on top of each other with the final 3DIC device having multiple layers of transistors
  - Local metal layers are used which along with TSVs provides a very high level of interconnect

A die stack with TSVs can be seen in Fig. 4.2.

4.2 Construction

There are other definitions on how the dies are bonded together:

- Wafer-to-Wafer
  - current Electrostatic discharge (ESD) mitigation allows implementation of unbuffered IO
  - potential low yield because of lack of knowledge regarding Known Good Die (KGD)
- Die-to-Wafer
  - will need additional ESD mitigation support
  - higher yield because of KGD
- Die-to-Die
  - will need additional ESD mitigation support

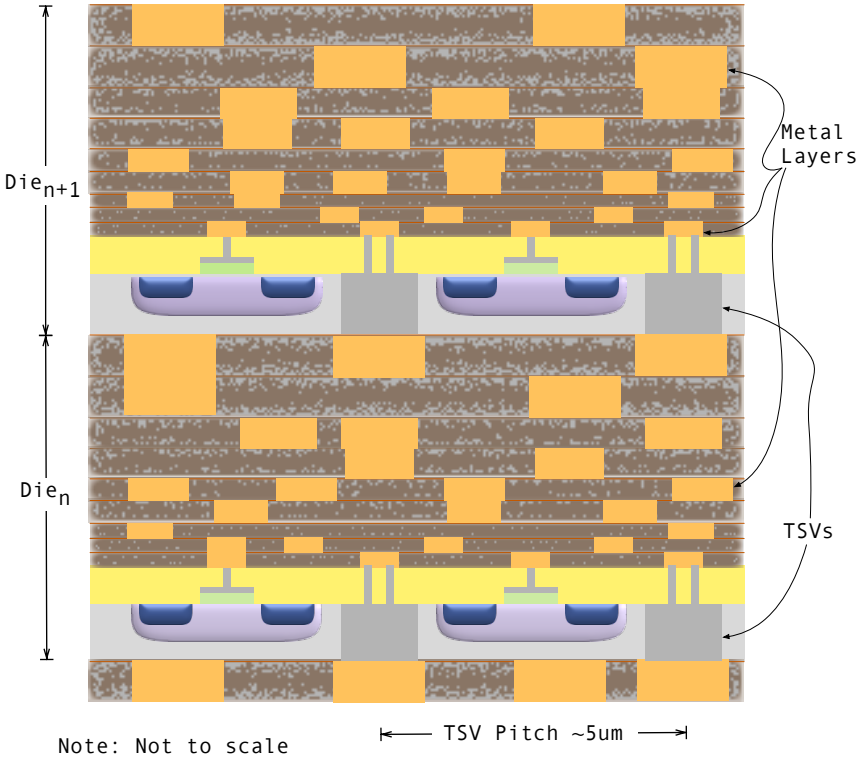


Figure 4.2 Die Stack profile [ITR15] with TSVs

– higher yield because of KGD

This work is targeting 3DIC technology that supports 3D-Stacked SoC or 3D-Stack IC with high levels of interconnect. To avoid using large IO buffers for the TSV interconenct, this work assumes that the 3DIC technology supports unbuffered interconnects. This would suggest wafer-to-wafer bonding because of the existing ESD mitigation during wafer handling although it is anticipated that improved ESD mitigation will be introduced in future manufacturing steps.

4.3 Design Guidelines

The technology roadmap in [ITR15] and the information in [Pat14] suggests 5 µm pitch TSVs is a reasonable design goal. This work assumes a one-to-one ration of signal TSVs to power/ground TSV so when accounting for area associated with TSVs, the number of signal TSVs are doubled.

As a large amount of TSVs are employed, TSV energy cannot be ignored. Most of the energy dissipated in the TSV is associated with the charging and discharging of the TSVs capacitance. For a 5 µm pitch and 2 µm radius TSVs, [BGO17] table I suggests an average capacitance of 4.2 fF [1].

Based on (4.1) and assuming a supply voltage of 1.0 V, the power associated with a TSV is shown in equation (4.2).

$$\text{Energy to charge a TSV,} \quad E_{tsv} = \frac{1}{2} \cdot C_{tsv} \cdot V^2 \tag{4.1}$$

$$\begin{aligned} \text{Energy to charge a TSV,} \quad E_{tsv} &= \frac{1}{2} \cdot C_{tsv} \cdot V^2 = \frac{1}{2} \cdot 4.2 \times 10^{-15} \cdot 1.0 = 2.1\,\text{fJ} \\ \text{Power per TSV,} \quad P_{tsv} &= E_{tsv} \cdot \text{bit rate} \\ \text{normalizing to a clock of 1.0 GHz} & \\ \text{Power per TSV per Hz} &= 2.1\,\mu\text{W/Gbit/s/TSV} \end{aligned} \tag{4.2}$$

The TSV design guidelines used by this work are summarized in table 4.1.

[1] [Tezb] suggests a lower capacitance

Table 4.1 TSV Design Characteristics

	Dimensions		
Parameter	Pitch	Radius	Power
Value	5 µm	2 µm	2.1 µW/Gbit/s/TSV [BGO17]

CHAPTER

5

DRAM OVERVIEW AND CUSTOMIZATIONS

There are two types of memory employed in ASICs and ASIPs, Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM). Both of these technologies have a similar top level block diagram which contains an array of storage elements, a means to address into a particular row of memory cells and a means to read and write a column of those cells. A basic block diagram is shown in figure 5.1.

The major difference between the SRAM cell and DRAM cell is the number of elements it takes to implement. The SRAM cell takes six transistors and the DRAM cell takes one transistor and one capacitor. The size of the DRAM cell means DRAM arrays provide five to six times more storage density

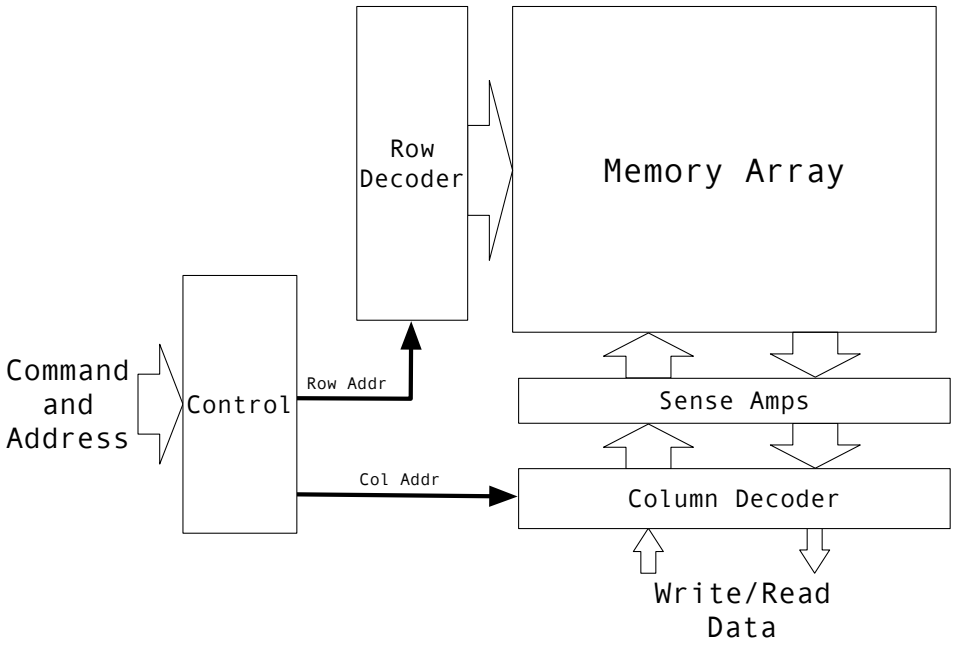


Figure 5.1 Typical Memory Block Diagram [Jac07]

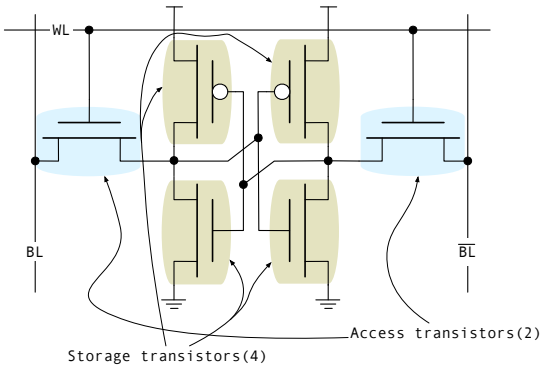
when compared to a similar sized SRAM array.

The major disadvantage is the capacitor cannot hold a indefinitely because the leakage currents in ICs cause the charge to leak away. If kept unchecked, the stored value will dissipate and it is this behavior that makes accessing a DRAM array more complicated than a similar SRAM array.

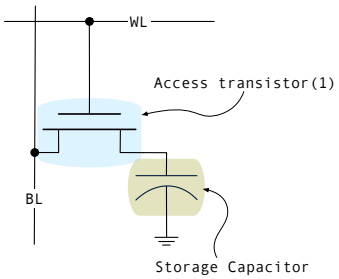
5.1 Accessing DRAM and SRAM

Accessing a typical SRAM involves providing an address and either reading or writing the contents of that location. The read or write often completes in one or two clock cycles depending on whether the SRAM employs internal registers which are used run the SRAM with a faster clock.

The storage cell inside the SRAM is formed from cross-coupled transisters (see figure 5.2a) which latch the contents and hold the contents indefnitely or until power is removed from the device. The storage structure employs six transistors and allows the access logic to be relatively simple and fast



(a) SRAM Storage Cell [Jac07]



(b) DRAM Storage Cell [Jac07]

Figure 5.2 RAM Storage Cell Types

but has a relatively low density because of the number of transistors employed.

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

When an SRAM cell is read, the cross-coupled transistors retain the stored value. The reasons behind this added complexity is the memory cell inside the DRAM which is formed from a capacitor (see figure 5.2b) which holds a charge reflecting a logic zero or one. When a page(row) of a DRAM memory array is read by the sense amps (see figure 5.3), the process of sensing the charge on the capacitor causes the capacitor to discharge and lose its contents. To alleviate this problem when the read occurs, the entire contents of the page are transferred to registers, refered to as "page intermediate store" in figure 5.3. The process of transferring a page to the intermediate store is known as a "page open". Once this transfer is complete, portions of the open page can be read similar to reading an SRAM.

The problem is that if another read wants to access data that is not in the page, the page has to be closed and another page opened. This involves transferring the previously registered page back to the array to recharge the capacitors in the memory array storage elements. The next page can then be opened and transferred to the page intermediate registers.

In practice, the DRAM protocol is separated into "page" commands and "cache" commands. The page commands open and close pages and the cache commands read and write to the page intermediate store.

The process of opening and closing pages is a relatively long time, typically 10-20ns. Once a page is open, accessing the intermediate store is much faster. So if the accesses are somewhat random and different pages are constantly accessed and pages are constantly being opened and closed, the average time to complete reads and writes are very large when compared to SRAM. To alleviate this issue, the DRAM is formed from more than one array of storage elements, between 8 and 32, known as banks. The idea is that while a page from one bank is being accessed, another banks page can be opened in preparation for a future read (or write). This access protocol is rather complicated and involves interleaving page commands and cache commands to multiple banks. The system memory controller logic must keep track of which pages are open inside the DRAM and for each memory request must determine if a page needs to be closed and another opened before reading (or writing) the intermediate store. If consequtive accesses are not sequenced carefully, the performance of DRAM can be poor.

5.1.1 Access Locality/Reuse and SRAM as a Cache

In general purpose computing, the sequence of accesses cannot always be controlled, so SRAM is typically used as the first level of memory with DRAM used as the primary storage. Using SRAM as this first level memory is called a cache and acts like a mirror of the DRAM contents. These caches have been used for decades to isolate the computing system from unpredictable access behavior of the DRAM. The general idea behind caches is that most data exhibits spatial and temporal locality. This "locality" means that when a computer program uses a piece of data in memory, it is very likely that soon after other data "close" to that data will be used and/or the same data will be reused. So when a piece of data in memory is accessed, that data and a large block of data in close proximity to the requested data are transferred to the cache. The cache is designed to hold multiple of these

blocks of data often resulting in tens of kB of SRAM. When other memory requests are made, the memory controller checks to see whether the block associated with the requested data is present in the cache. If the requested data is contained in a block present in the cache, this is considered a "hit" and the data in the cache is used. If the requested data's block is not present, this is known as a "miss" and the slower main memory must be accessed. This access results in another "block" of data being transferred to the cache. If all the blocks in the cache are currently fully employed, one of the blocks must be freed up to make space in the cache for the new block. A block is chosen and transferred back to the main memory and the new block is read and transferred to the cache.

To make effective use of the cache, the access behavior of the computer program must exhibit this locality behavior. If the cache blocks are large enough and the programs access behavior exhibit locality, then employing SRAM is effective and the slower DRAM access times can be somewhat hidden.

As mentioned in section 1.1 much of the ASIC and ASIP ANN research has focused on taking advantage of the performance and ease of use of SRAM. If the target application means the memory access behavior exhibits some locality and the SRAM cache can be made large enough to avoid high levels of cache misses, then this use of SRAM can be effective.

But this works target application is such that the access behavior exhibits no or little locality (reuse) and block transfers between the cache and main memory would be constantly occuring. Under these circustances, **DRAM bandwidth will be the bottleneck**.

This work focuses on using DRAM as the primary storage and managing the accesses to ensure the DRAM is used effectively. With the increased bandwidth achieved from the additional DRAM customizations discussed in section 5.2.1 and 5.2.2, this work demonstrates DRAM bandwidths 10X faster than what is available with 2 or 2.5D solutions.

This high level of DRAM bandwidth provides this work the ability to process multiple disparate ANNs at or near real-time whilst being **10-100X faster than state-of-the-art solutions**.

5.2 DRAM Customizations

In a typical DRAM, 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. An ASIC, ASIP or GPU implementation may combine multiple devices to generate bus widths of the order of 1 kbit wide. When using 2.5D technology and High Bandwidth Memory (HBM) with their Pascal™GPU accelerator device, NVidia® achieve a raw DRAM bandwidth approaching 6 Tbit/s [Nvi][1]. However, experience has shown [Far11] [Aba15] that usable bandwidth will likely be much lower. Regardless, this existing technology does not achieve the required bandwidth (2.2).

To achieve increased DRAM bandwidth this work is proposing two changes to the Tezzaron®DiRAM4 [Teza] 3D DRAM. The most significant customization is to widen the databuses to generate more raw bandwidth. This is discussed further in section 5.2.1.

With the customizations discussed in sections 5.2.1 and 5.2.2, this work demonstrates DRAM bandwidths >10X faster than what is available with 2 or 2.5D solutions.

5.2.1 Customization One: Very-Wide Bus

Figure 5.3 shows a block diagram of a typical DRAM.

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 5.4).

This work assumes the DRAM interface protocol uses Double Data Rate (DDR) with a bus width

[1] datasheet also shows a total design power (TDP) of 300 W

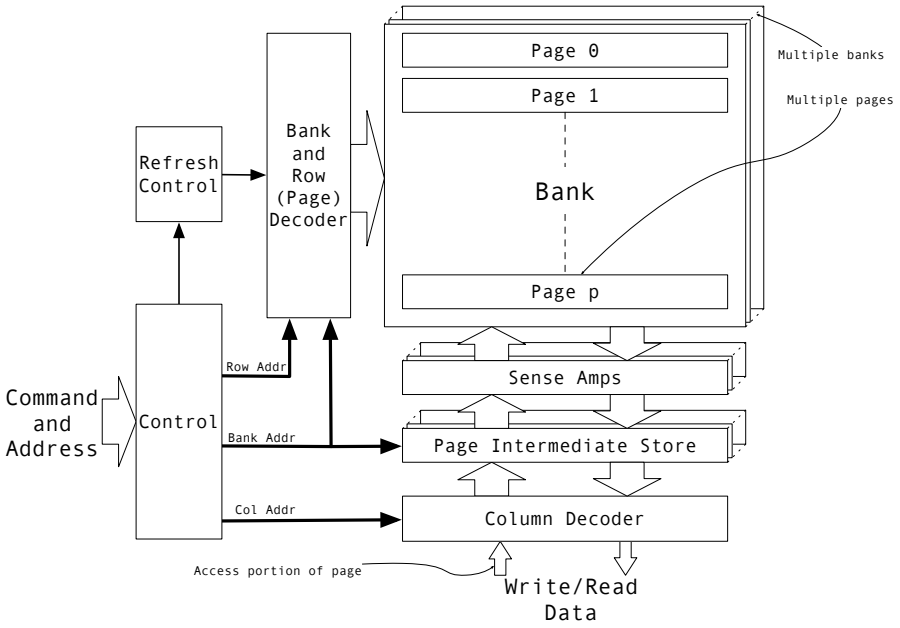


Figure 5.3 Typical DRAM Block Diagram

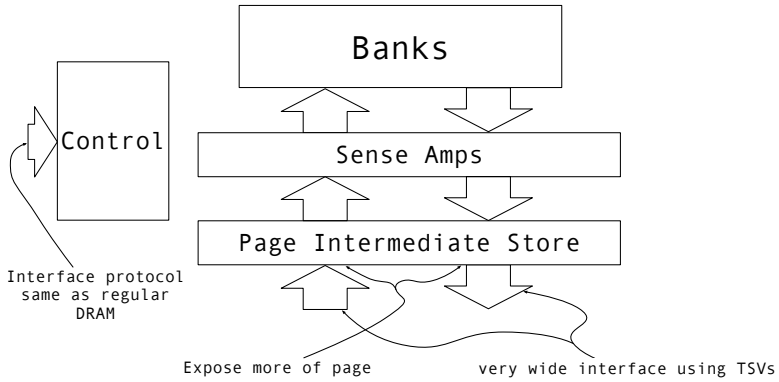


Figure 5.4 Exposing more of the DRAM page

of 2048. Given the DiRAM4 employs a burst of two for read and write cycles, an entire DiRAM4 page of 4096-bits is accessed during each read or write.

5.2.2 Customization Two: Write Mask

When processing an ANN, to compute the activation of an individual AN involves reading the pre-synaptic AN activations 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, 100s or 1000s 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.

This work assumes single precision floating point for ANN weights and activation, so a mask bit will be provided on a word basis or every 32-bits.

CHAPTER

6

STATE OF THE ART

For the most part, large scale ANNs have been implemented using GPUs. In some cases, such as CNNs, these GPUs are quite effective when the ANN parameters can be reused in the GPUs processor local SRAM.

Much of the ASIC and ASIP research has focused on CNNs such as [Che16][Far11], In some cases, implementations have focused on solving specific processing "hot-spots"ANN [Che16]. Almost all ASIC and ASIP solutions employ arrays of PEs each with local processing capability and local memory. For most of these, the size of the ANN supported is limited by the size of the local memory. In some cases the area consumed for local memory can exceed 65% of the processing element die [Kim16a][Che14].

Those that employ external DRAM, such as NnSP[Esm05] and NeuroCube[Kim16a] still load

weights and ANe states to local SRAM prior to processing.

In the case of NnSP[Esm05], the paper discusses caching data to bridge the speed gap between external memory and the PE but does not provide details on how to ensure data locality when reading a DRAM cacheline and how to minimize the impact of DRAM protocol.

NnSP does not provide any detail regarding network size and supported types.

Neuflow[Far11] is limited to CNNs and the external memory is Quad data rate (QDR) SRAM and thus will be limited by the network size.

NeuroCube uses a 3D stack along with HMC 3D-DRAM and data is transferred from the DRAM to the PEs via a NoC. The combination of limited HMC interface bandwidth and the NoC limits the processing performance.

Eyeriss[Che16] focuses on CNNs and specifically on the convolution "hot-spots". It does not support the pooling operations although these can be supported by a local CPU. However, it does not support the memory intensive fully-connected classifier layers. Eyeriss can not be effectively applied to locally-connected type ANNs such as Deepface [Tai14].

The current state-of-the-art does not:

- currently propose streaming data directly from 3D-DRAM through the processing functions to avoid the use of large local memory
- propose data structures to support continuous streaming from DRAM
- propose a 3D-stack supporting more than one processing layer

NeuroCube[Kim16a] is the only solution to explicitly discuss 3D memory but will be limited by its focus on utilizing local SRAM where our focus is on providing silicon for processing and not local storage. Neuflow[Far11] has processing elements that are limited to CNNs. NnSP[Esm05] transfers weights and inputs to local PE memory prior to processing. nn-X[Gok14] limits the number format to integer, and although there is some acceptance that similar results can be obtained compared

to floating point, this will likely limit the applcation space. nn-X is also limited to performing the convolution operation and SRAM size will limit its capability when performing the classifier stage.

Below is a summary of current technology.

6.1 NnSP[Esm05]

Uses external DRAM with a cache. Does not address the main issue of DRAM access limitations and providing data structures. It simply says "use a cache" but doesnt justify locality issues.

Transfers weights and inputs to local PE prior to processing Similar "Streaming" terminlogy but streams from local memory. ANe outputs are kept locally and transferred to other ANe via an NoC.

No detail on processing element but likely limited to "standard" feedforward ANNs.

6.2 NeuroCube[Kim16a]

NeuroCube[Kim16a] transfers data to PE prior to processing which can limit classifier size although most convolution kernels can be supported. The SRAM on the PE is significant (see Fig. **??**). NeuroCube uses HMC 3D-DRAM which has limited bandwidth and this will limit network processing performance. The NeuroCube implements a processing layer but does not address adding additional layers.

3D uses HMC - limited by HMC bandwidth - is serdes a good choice for die-to-die communication (typically used for die to chip/board?) - doesnt really discuss adding additional layers

Transfers data to PE prior to processing

6.3 IMAPCAR [KO11]

IMAPCAR uses external SRAM so will be limited by network size. The architecture of IMAPCAR is optimized for processing video.

In general will not scale to ANNs of interest.

6.4 NeuFlow [Far11]

Neuflow[Far11] has processing elements that are "highly optimized" to CNNs. Neuflow is designed around external QDR memory and does not address the issues associated with supporting large networks.

6.5 nn-X [Gok14]

nn-X[Gok14] limits the number format to integer, and although there is some acceptance that similar results can be obtained, this will likely limit the applcation space. nn-X is also limited to performing the convolution operation and SRAM size will limit its capability when performing the classifier stage. nn-X was implemented in an fpga and does not address SRAM size when scaling to larger networks.

6.6 Diannao [Che14]

Diannao[Che14] acknowledges the need for large amounts of storage which will exceed the capability of local memory. The kernel sizes are limited by SRAM size although it does state kernels can be "broken up". The SRAM use on the PE is significant (see Fig. **??**).

6.7 Eyeriss [Che16]

Eyeris[Che16] focuses on CNNs and specifically the convolution stage. Eyeriss ignores the pooling and classifier stages which although not as compute intensive they are memory intensive. Therefore Eyeriss limits network size and does not implement the network.

6.8 TPU [Jou17]

The TPU utilizes an 256x256 array of 64×10^3 processing elements and demonstrates significant (30X) performance improvements over GPUs and central processing units (CPUs). It does employ DRAM which provides the required storage capacity but employs SRAM as an intermediate store between the DRAM and the PEs. This Google designed solution acknowledges that it is bandwidth limited when implementing the fully connected ANNs. It also states that their experience of implementing ANNs in the Google server farms suggests that these fully connected ANNs represent the bulk of their processing requirements. The simpler CNNs only represent 5% of the servers ANN processing requirements. It should also be stated that this paper also believes that GPU solutions cannot reach the performance targets even though the GPU community might state otherwise.

CHAPTER

7

SYSTEM OVERVIEW

As mentioned in chapter 1 and chapter 3, this works target application implements mulltiple ANNs each of which have limited opportunities for both weight reuse and batch processing. These requirements require DRAM to be employed for main storage of ANN parameters and local SRAM is of limited use. Under these conditions, the DRAM bandwidth is the system bottleneck.

To meet these requirements, this work proposed employing 3DICs technology along with a customized 3D DRAM and ASIC technology. By physically staying within the 3DIC footprint and taking advantage of high density TSVs this work is able to maintain a significantly higher bandwidth over 2D or 2.5D ASIC or ASIP solutions. The objective was to demonstrate that a pure 3DIC system can implement multiple disparate ANNs within reasonable power and area constraints.

The 3DIC system die stack (figure 7.1) includes the 3D-DRAM with a system manager below and

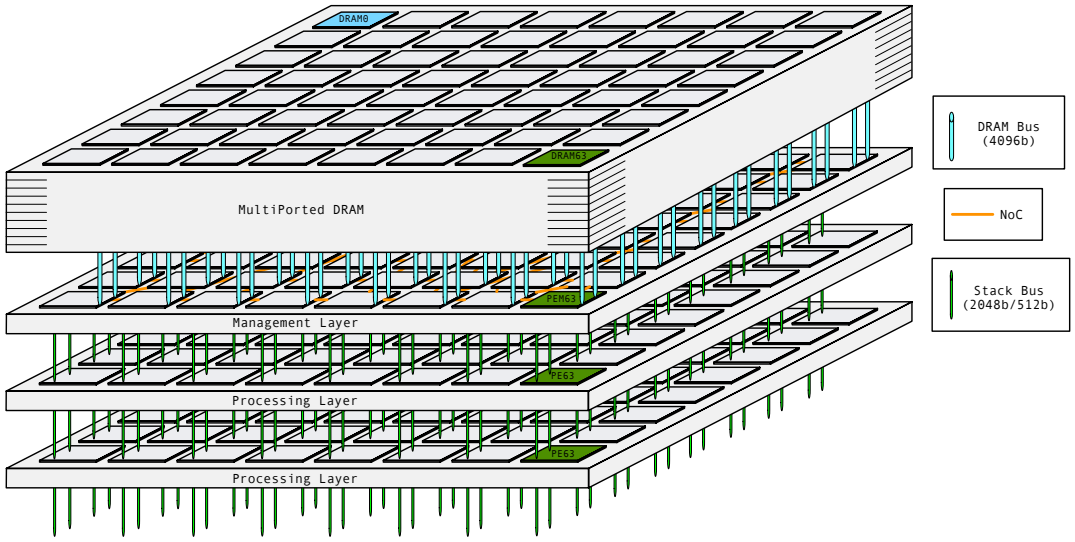


Figure 7.1 3DIC System Stack

one or more processing layers below the manager.

3D-DRAM has recently become available in standards such as 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.

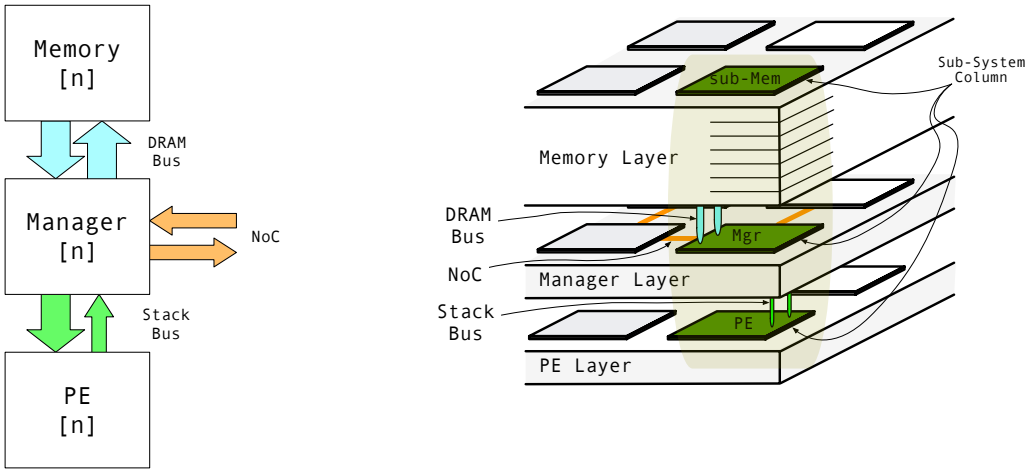
In the case of HBM and DiRAM4 [Teza], the technology can be combined with additional custom layers to provide a system solution.

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

This work targeted a baseline system with:

- Computations requiring Single-precision floating-point format (binary32)
- Tezzaron Dis-Integrated 3D DRAM (DiRAM4) [Teza] for main memory
- 28nm ASIC technology

The work includes customizing the interface to a 3D-DRAM, researching data structures to describe storage of ANN parameters, designing a memory manager with unique micro-coded instructions and



(a) Sub-system column logical block diagram

(b) Sub-system column physical diagram

Figure 7.2 Sub-System Column (SSC)

a PE layer. The targeted 3D-DRAM, the Tezzaron DiRAM4 employs 64 disjoint memories arranged in a physical array.

This works system is designed such that a sub-system, known as a Sub-System Column (SSC) operates on one of the disjoint sub-memories within the DiRAM4 (see figure 7.3). As shown in figure 7.2, the SSC includes the DiRAM4 sub-memory (referred to as the SSC memory), a manager module and a PE module. The DiRAM4 has 64 sub-memories so there are 64 SSCs. The SSC has been designed as a standalone unit and does not have a knowledge of the other SSCs in the system.

An overview of the various blocks and interconnects are given in sections 7.1 through 7.6 with additional detail provided in chapter 9.

7.1 Customized DRAM : Dis-Integrated 3D DRAM (DiRAM4)

The DiRAM4 [Teza] 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 1 Gbit with a total capacity of at least 64 Gbit. Unlike traditional DRAM, the DiRAM4 has two independent channel

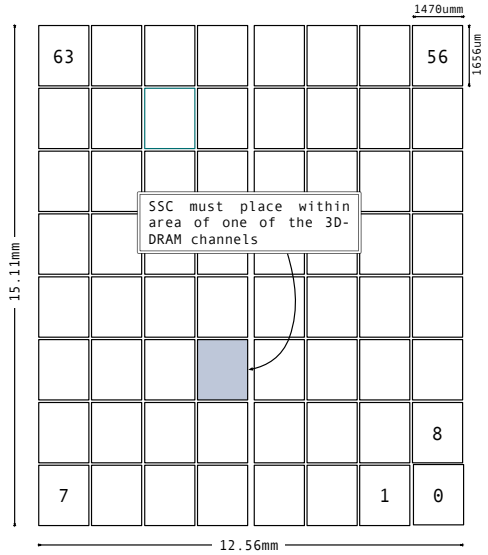


Figure 7.3 DRAM Physical Interface Layout showing area for SSC

which are accessed using DDR signalling on the control signals. Each channel has 32 banks and 4096 pages per bank with 4096 bit/page.

The standard DiRAM4 has a 32-bit read databus and a 32-bit write databus enabling simultaneous read and write. Both read and write databuses employ DDR signaling. The read and write transactions are burst-of-two providing 64bits per read/write. When accessing a DRAM, a read and write are often referred to as a cacheline. The device is designed to operate at 1 GHz although this work targeted a 500 MHz clock frequency.

This work is proposing customizations to the DiRAM4 which are outlined in chapter 5. One of these proposed changes is to widen the read and write databuses to 2048-bits. Using the same burst-of-two means each read and write will access an entire page. A cacheline becomes 4096-bits. Another proposed change is the add mask bits to the write databuse to avoid having to perform read/modify/writes when writing back data much smaller than the new large cacheline.

7.2 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 area and bandwidth benefits provided by TSVs. The high density interconnect provided by TSVs allows the system to take advantage of the very wide DRAM bus provided by the DRAM customization described in section 5.2.1. The wide interconnect between the manager and PE are also implemented using TSVs.

The interconnect between the manager and PE is referred to as the stack bus. The interconnect between the manager and DRAM is referred to as the DRAM bus.

7.2.1 Stack Bus

The stack bus is bidirectional with a 36-bit Out of Band (OOB) configuration bus from the manager to the PE, a 2048-bit "downstream" data bus from the manager to the PE and a 512-bit "upstream" data bus from the PE to the manager.

The 36-bit OOB bus is designed to send configuration packets to configure the StOp and Single-Instruction Multiple-Data (SIMD) blocks inside the PE. The packet primarily contains the StOp function to be performed on the downstream data and the operation the SIMD is to perform on the result from the StOp. It also includes the size of the downstream data stream and an operation idenfication tag.

The 2048-bit downstream stack bus is designed to carry 32 lanes of data with each lane containing two operand buses. In the baseline system, the two operand buses are designed to carry streams of binary32 numbers. This allows the downstream stack bus simultaneously stream 32 execution lanes each with two 32-bit argument streams. Typically these streams are the weights and pre-synaptic ANe states to calculate the states of up to 32 ANes. The streams can also be configured to calculate the state of a single ANe where the weights and pre-synaptic ANe states for the single ANe are sent in

parallel across the downstream stack bus and a reduction operation can be performed in the PE by the StOp and SIMD.

The 512-bit upstream bus is primarily designed to send the results of a downstream operation. Typically this is the states of up to 32 ANes. The upstream bus is packetized with the result data contained in the data portion of the upstream packet. Additional information includes the operation identification tag provided by the OOB command packet and the number of data words. The upstream bus is not as wide as the downstream bus because the ratio of downstream operands to result data is a function of the pre-synaptic fanin. Based on the average fanin from the baseline ANN shown in table 2.12a, a 512-bit bus exceeds the required average bandwidth.

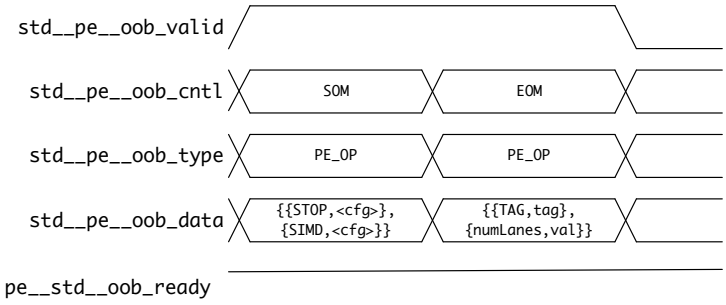
The stack bus OOB, downstream and upstream signalling can be seen in figure 7.4.

7.2.1.1 Common Bus Signalling

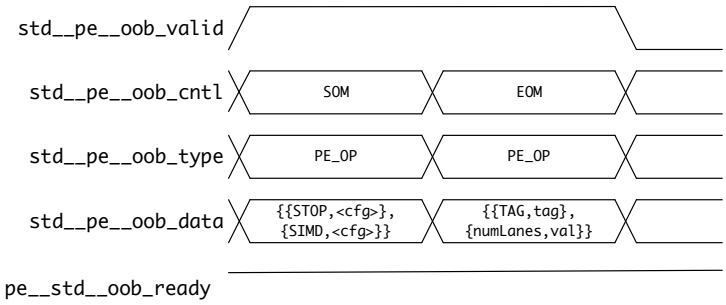
With almost all interfaces in this system, additional control signals are used to a) validate, b) delineate and c) flow control messaging between blocks. This "common" bus protocol signal group includes three signals, VALID, CNTL and READY. The single VALID signal is high when the bus contains valid information. The two bit CNTL signal is used to delineate by encoding start of message (SOM), middle of message (MOM) and end of message (EOM). Because of the asynchronous nature of the system, most interfaces employ small first-in first-out queues (FIFOs). When these FIFOs are almost full, the source is flow controlled by the READY signal. The point at which the FIFO indicates the source to stop sending is based on the latency of the particular interface. The common signalling protocol will be seen in all waveform diagrams as seen in figure 7.4.

7.2.2 DRAM Bus

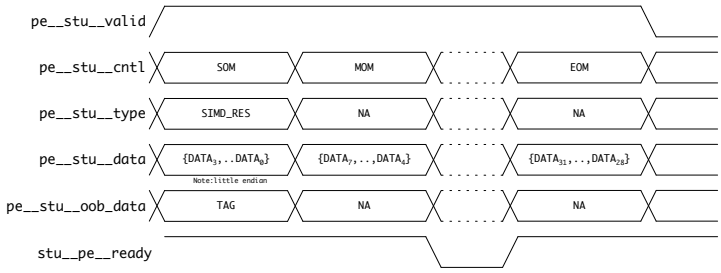
The interface to the DiRAM4 is similar to classic DRAM interfaces with control signals including bank address and multiplexed page/cache address bus. There are separate 2048-bit DDR read and write databuses (see section 5.2.1). In total there are 4180 signals in the DRAM interface. Other than the



(a) Stack downstream OOB Bus



(b) Stack downstream Data Bus



(c) Stack upstream Bus

Figure 7.4 Stack Bus signalling

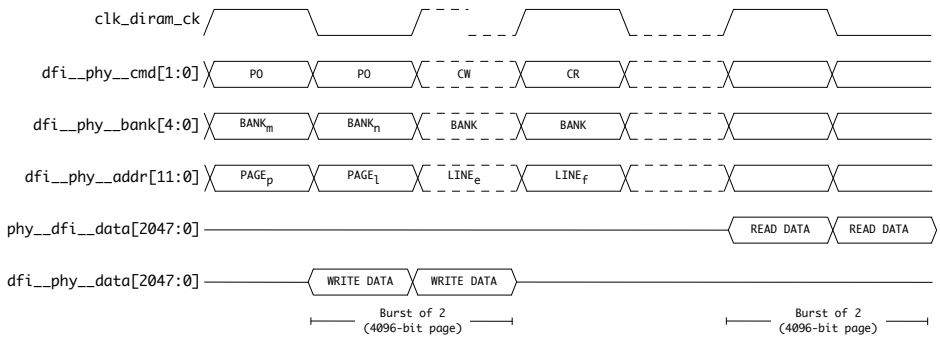


Figure 7.5 Read and Write request to DiRAM4 [Teza]

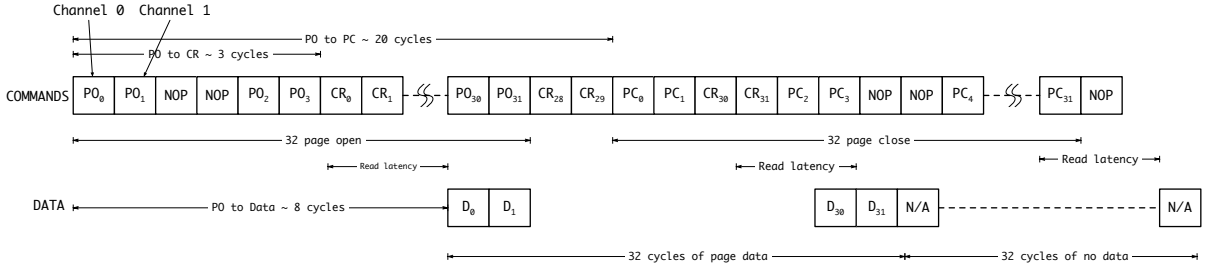
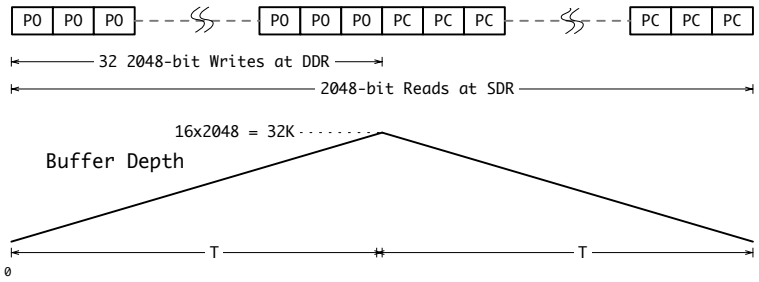


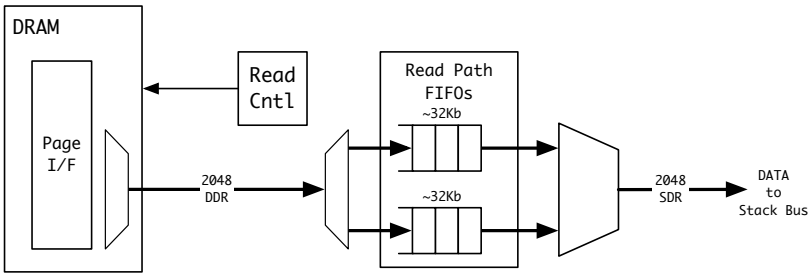
Figure 7.6 Worst case PO/PC sequence

wide databuses, the interface protocol is as described in [Teza]. A timing diagram showing a read and write to the DiRAM4 are shown in figure 7.5.

The bandwidth of the DRAM bus is designed to ensure data can be constantly maintained to the stack downstream bus. Because an entire cacheline is accessed for each read request, there is an extreme case when back-to-back requests result in up to 32 DRAM page opens commands. It is possible that this sequence page opens could be followed by page closes resulting in a period when no useful data is being read from the DiRAM4. This case is shown in figure 7.6. To accomodate this, the DRAM bus has twice the raw bandwidth of the downstream stack bus. Therefore under this worst case scenario the higher bandwidth data from the DiRAM4 must be buffered, as shown in figure 7.7a. In this case, per channel 32 kbit FIFOs must be placed in the read path as shown in figure 7.7. These FIFOs are instantiated in the managers memory read controller(s) as described in section 9.1.3.



(a) Worst case read path FIFO depth



(b) Read path buffering diagram

Figure 7.7 DRAM Read Path Buffering

7.3 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 (for more detail see section 8.1.1) are sub-divided into descriptors to describe memory read operations, processing engine operations, memory write operations and general system operations for synchronization. The manager reads these system instructions from an instruction memory, decodes the descriptors and configures the various blocks in the system. The configuration includes:

- initiating operand reads from DRAM
- preparing the processing engine (PE) to operate on the downstream stack bus operand streams
- prepare the result processing engine to take the resulting ANe activations from the PE via the

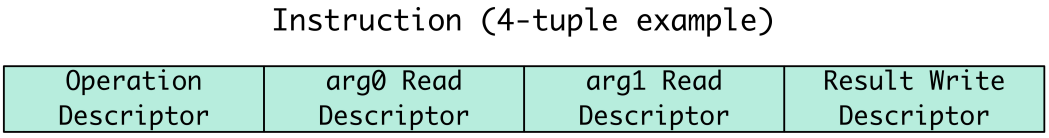


Figure 7.8 Instruction 4-tuple

upstream stack bus and write those results back to the DRAM

- replicate the resulting ANe activations to neighbor managers over the NoC for processing of other ANN layers

The most common instruction is to perform ANe state calculation on a group of ANes. This instruction contains four descriptors (see figure 7.8), an operation descriptor, two memory read descriptors and a memory write descriptor.

The operation descriptor describes the operations the PE will perform, which typically includes a multiply-accumulate (MAC) to be performed by the StOp followed by a Rectified Linear Unit (ReLu) function to be performed by the SIMD. The manager extracts the operation information, embeds the information in an OOB packet and sends the packet to the PE over the downstream OOB bus. The two memort read descriptors are used to define the memory locations of the downstream stack data buses. There is one memory read descriptor for each of the two operand streams in the execution lanes, one typically defines where the pre-synaptic weights are stored and one for where the pre-synaptic ANe states are stored. The architecture is designed to for an instruction to compute the state of multiple ANes or for an individual ANes state to be computed. If a group of ANes are computed, the pre-synaptic connection weights for the ANes are stored interleaved and when read from DRAM are directed to one of the streams of each of the execution lanes. If a single ANe is computed, the pre-synaptic connection weights for the ANes are stored linearly and when read from DRAM are directed to one of the streams of each of the execution lanes. The pre-synaptic ANe states are stored in row-major order and when read are broadcast to the other stream of each execution lane.

7.4 Processing Engine Layer

The PE contains two main processing modules, the StOp and the SIMD block. Both the StOp block and the SIMD have 32 execution lanes. The execution lanes inside the StOp contain functions required to perform ANe related operations. The SIMD will be based on the device described in [Sch17].

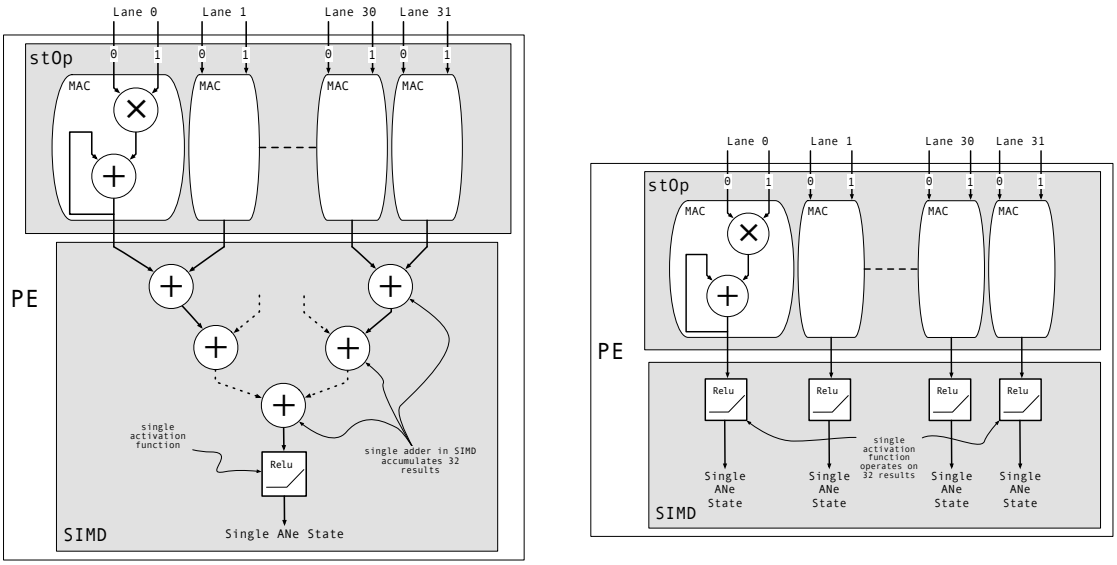
The PE is configured by the manager to perform operations on two operand data streams from the manager. The StOp is able to operate on this data directly from the manager at the full bandwidth rate of the stack bus and does not have to be stored in local SRAM prior to processing. There is a small FIFO to provide buffering to allow asynchronous configuration of the StOp block and the source of the streaming data in the manager. The FIFO also allows the two argument streams in each of the execution lanes to wander with respect to each other and with respect to the other lanes.

The architecture is expandable to allow various functions to be provided in the StOp. The current baseline implementation includes the MAC operation. There is a MACs per execution lane allowing up to 32 simultaneous pre-synaptic ANe $weight \cdot state$ computations on the two operands from the two streams in each execution lane. These computations can be for a group of up to 32 ANes or for a single ANe as shown in figure 7.9.

If it is for a group of ANes, the SIMD only has to perform the activation function on the 32 results from the StOp. If a single ANe is being processed, the SIMD must accumulate the result from each execution lanes StOp before applying the activation function. The activation function currently implemented is the ReLu. The ANe states can be sent back to the manager over the stack upstream bus or retained for further processing such as pooling or softmax calculations.

7.5 Inter-Manager Communication

During configuration and/or computations, it may be required to replicate data to other SSCs. During ANe computations, an SSC only reads ANN weights and states from its local DRAM and not DRAM



(a) Single ANe calculation

(b) Group ANe calculation

Figure 7.9 PE ANe calculation

of other SSCs. In some cases, such as fully connected layers and locally-connected layers with ROI overlap, the ANe computation in an SSC for layer n may include ANe states from layer $n-1$ which were computed in another SSC. In this case, when ANes are being computed for layer $n-1$, the operation instruction contains information as to where the result should be written back for computing ANes in layer n. If a particular layer $n-1$ ANe is required by another SSC when computing a layer n ANe, the result is sent to all dependent SSC via the NoC. The instruction contains one or more storage descriptors (see section 8.1.2.1) that identify the destination of the operation result. If the result must be replicated, there are multiple storage descriptors. When the result is returned by the PE, the manager examines the storage descriptors and determines which SSC the result should be sent. The manager creates an NoC header which includes information on all the destination SSCs. This is encoded as a 64-bit field although the NoC header supports a multicast group. The multiple storage descriptors and the result data are placved in the data portion of the NoC packet. As the packet traverses the NoC, it is replicated to outgoing ports based on the destination bit field. When the

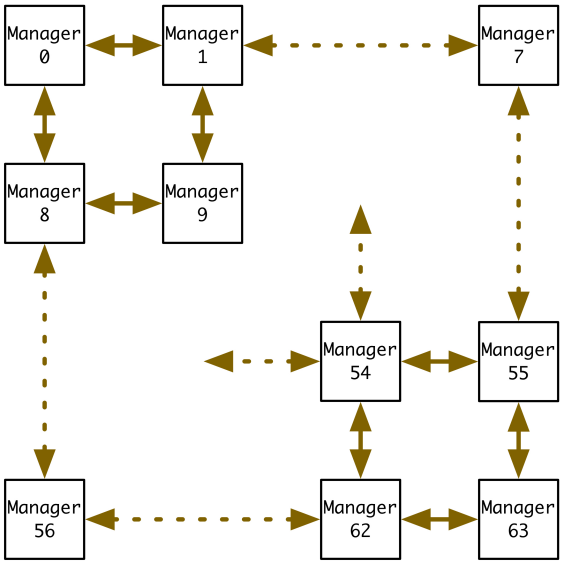


Figure 7.10 NoC manager connectivity

destination SSC receives the packet, it extracts the storage descriptor and writes the data to its local DRAM. The NoC packet format can be seen in figure 7.11. This inter-manager communication is provided by an NoC with all managers connected in a mesh as shown in figure 7.10.

When computing an ANN across multiple processing sub-systems, often ANe 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.

Each manager has an integrated NoC module that has four ports. The managers in the middle of the array use all four ports to connect to adjacent managers. The managers in the corners of the array connect to two adjacent managers. The managers at the edges connect to three adjacent managers. The host is connected to one (or more) of the managers at one edge of the array.

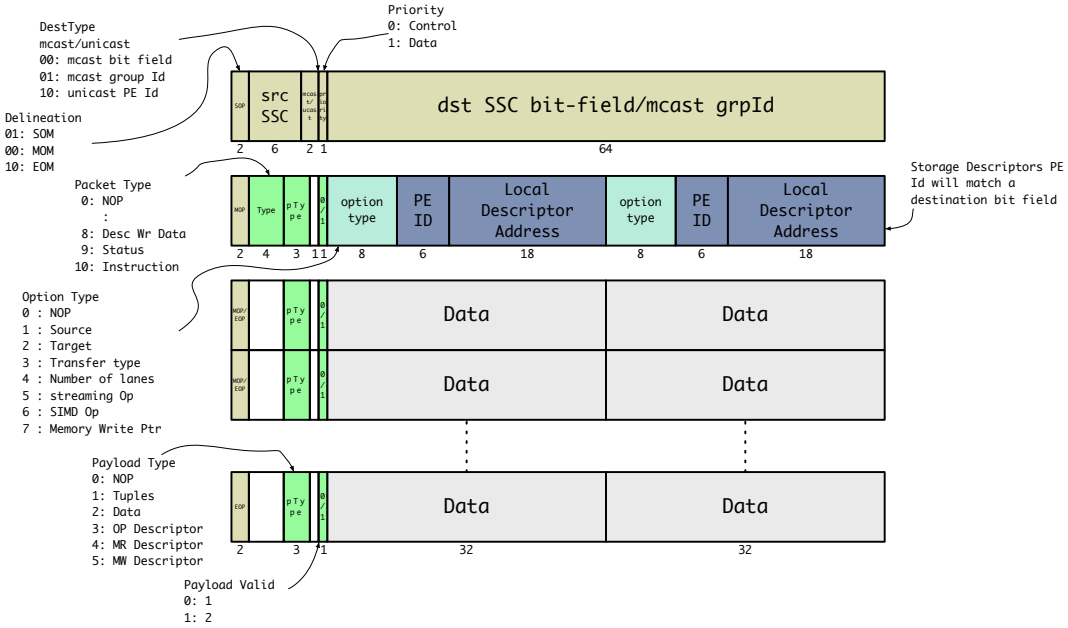


Figure 7.11 NoC packet format

7.6 Summary

A control and data flow diagram of the stack showing the 64 sub-system columns can be seen in figure 7.12.

7.6.1 Configuration, Command and Data Flow

**** WIP ****

The host is responsible for loading each managers instruction memory over the NoC and a main controller module in the manager. The host sends a command to start the manager decoding a defined number of instructions at a specific address in the instruction memory.

The manager decodes instructions and sends control commands to various modules in the manager and the PE.

One module, the OOB downstream channel sends configuration packets to the PE over an

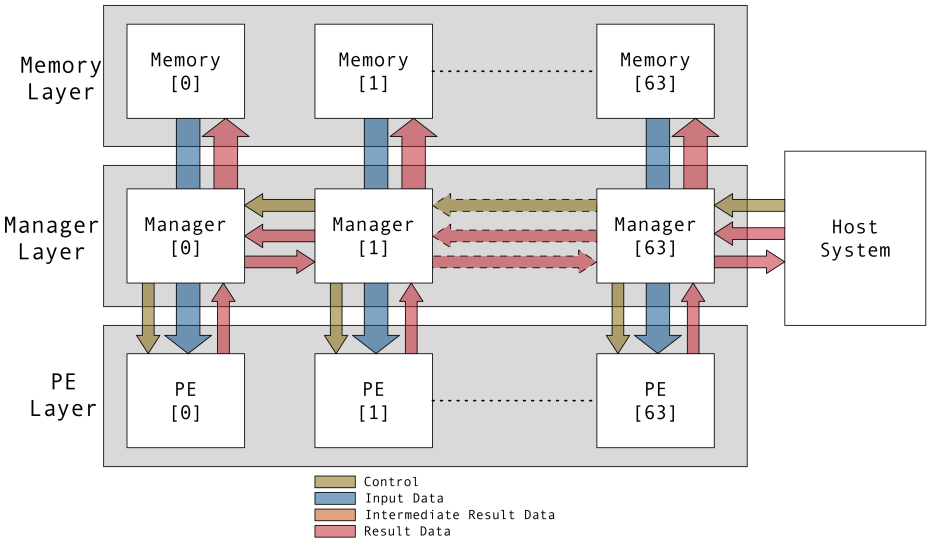


Figure 7.12 System Flow Diagram

OOB channel embedded in the stack bus. This configuration packet includes a) a pointer to a pre-programmed operation to be performed by the streaming operation module on the data streams to the PE and b) a pointer to a pre-programmed portion of the SIMD instruction memory which defines the operations to be performed by the SIMD on the result of the streaming operation block. in the PEcommwhich is a part of the is instructed to send SIMD and StOp commands over an out-of-band channel in the stack bus to the PE. The managers at the edge of the array The host is connected to one (or more) of the managers at one edge of the array. EachAll communication between SSCs is over an NoC. Some causes of communication between SSCs are host configuration, synchronization messages between SSCs, ANe state replication etc..

CHAPTER

8

SYSTEM OPERATIONS

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

- Synchronization to other managers
- Inform the Manager and PE which operations are to be performed
- Inform the manager where to access the states of the pre-synaptic neurons
- Inform the manager where to access the weights of the connections from the pre-synaptic ANs
- How to provide the pre-synaptic neuron weights and states to the processing engine execution lanes

- Inform the manager where to store the resulting AN state back to memory

This work has developed an instruction architecture to describe the above operations. In the baseline system, the manager is not responsible for performing specific algorithm computational 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:

- System Control
- Instruction decode
- Internal Configuration messages
- Operand read
- Result write
- Inter-manager communication

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 before returning data to the manager
- DMA/local memory controller
  - transfer configuration data to PE controller or to store StOp results to a small local SRAM which can be used for access by SIMD or by the StOp function

8.1 Manager Operations

8.1.1 Instructions

The instructions include information to control the operations below:

- To the Manager
  - Operation descriptor
  - Parameter/Weight Storage Descriptor(s)

    reads for the two argument streams to the PE

    read data can be broadcast or vectored to the PE
  - Result write storage descriptor

    include descriptors for all destination managers
  - Synchronization to other managers
- To the PE
  - StOp operation
  - SIMD operation
  - Number of active lanes
  - Operand Vector length

Instructions contain sub-instructions called descriptors. These descriptors contain, amongst other things the information to control the various operations associated with the processing of a group of ANes. The size of this group of ANes is related to the number of execution lanes, which for this work is 32. 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.

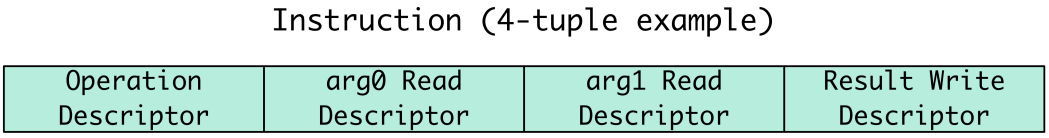


Figure 8.1 Instruction (4-tuple example)

To perform ANe activation calculations, an instruction will typically have four descriptors which will include:

1. Operation descriptor
2. Memory read descriptor for operand stream 0
3. Memory read descriptor for operand stream 1
4. Result descriptor write

The instruction is an n-tuple where the tuple elements are descriptors. The number of descriptors can vary based on the operation being performed. In figure 8.1 we see the format of a 4-tuple instruction which is used to perform an activation calculation for a group of ANes.

Within a descriptor, various options are described, such as storage descriptor pointer, number of operands etc.. The descriptor also employs an n-tuple format where the first tuple element always describes the descriptor operation followed by an m-tuple whose elements contain the options required by the operation. The option elements within a descriptor are a two-tuple with option and associated value. In figure 8.2 we see the format of a 6-tuple descriptor.

8.1.2 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.

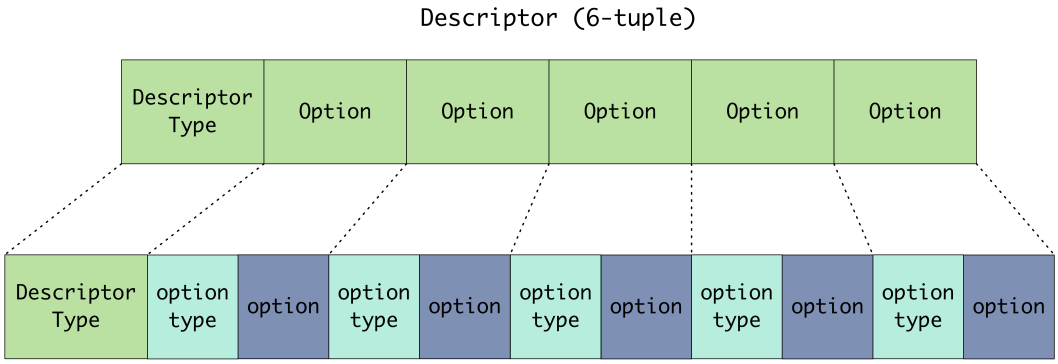


Figure 8.2 Descriptor 6-tuple

To provide parameters for the up to 32 execution lanes within the PE, the ANe parameters were stored 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 ANe parameters (weights) are stored in contiguous memory locations, providing the input state to a particular ANe presents us with an interesting problem.

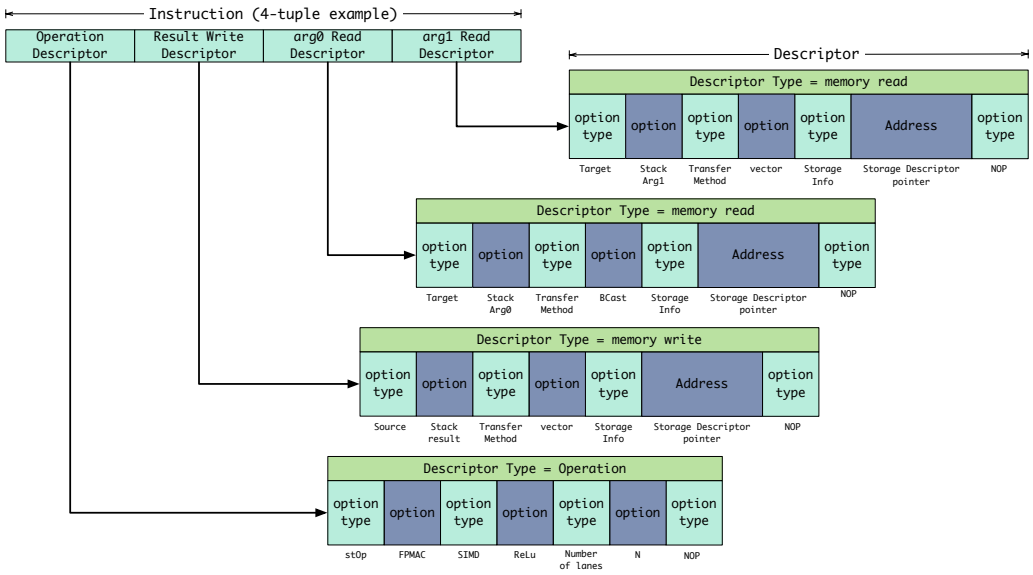
Most often DNNs are represented by layers of ANs whose pre-synaptic neurons are from the previous layer. These previous layers represent the input to a given layer. The first layers input is the actual input to the ANN.

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.

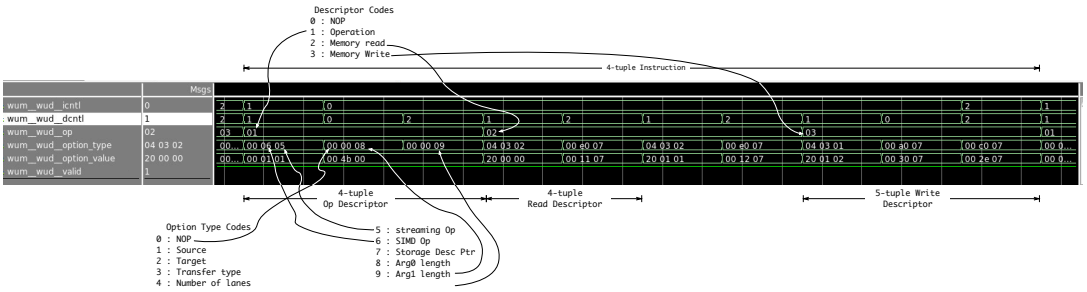
Any given AN operates on a region of interest (ROI) within the input array.

In figure 8.4, 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



(a) Instruction and Descriptors



(b) Instruction memory waveform

Figure 8.3 Instruction Details

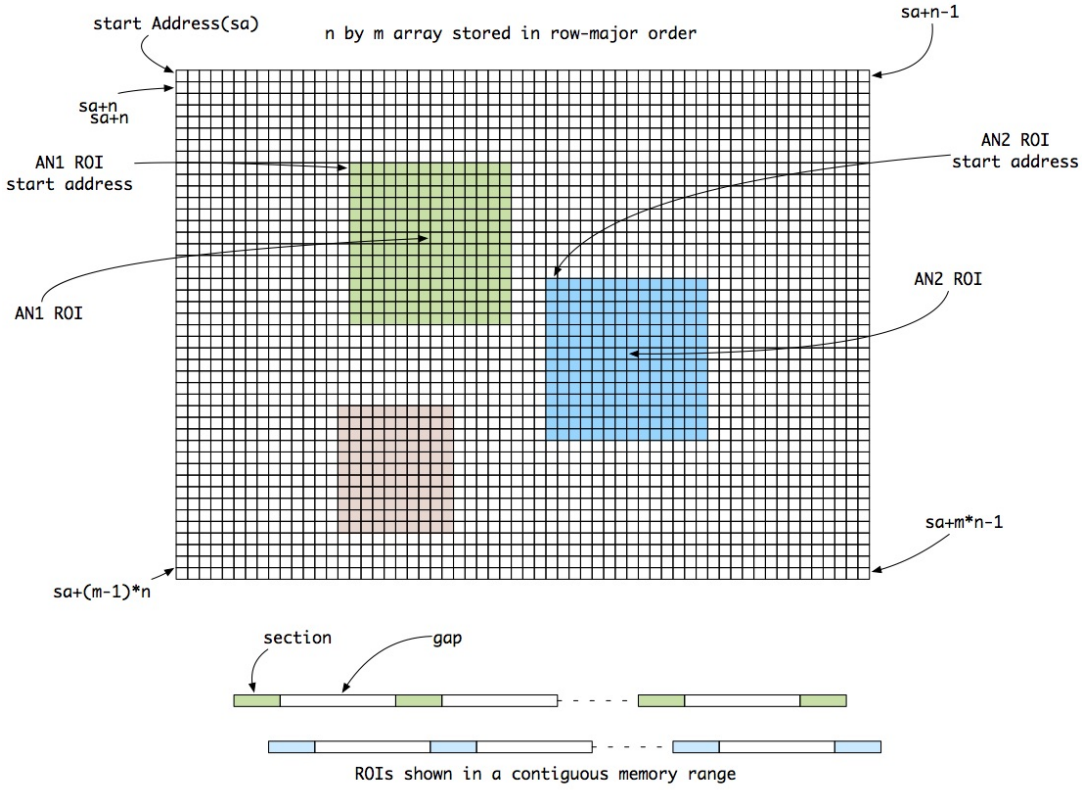


Figure 8.4 ROI Storage

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 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.

8.1.2.1 Storage Descriptor

This storage descriptor contains, amongst other things the start address of the ROI and a pointer 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 8.5 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.

8.1.3 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 thats 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.

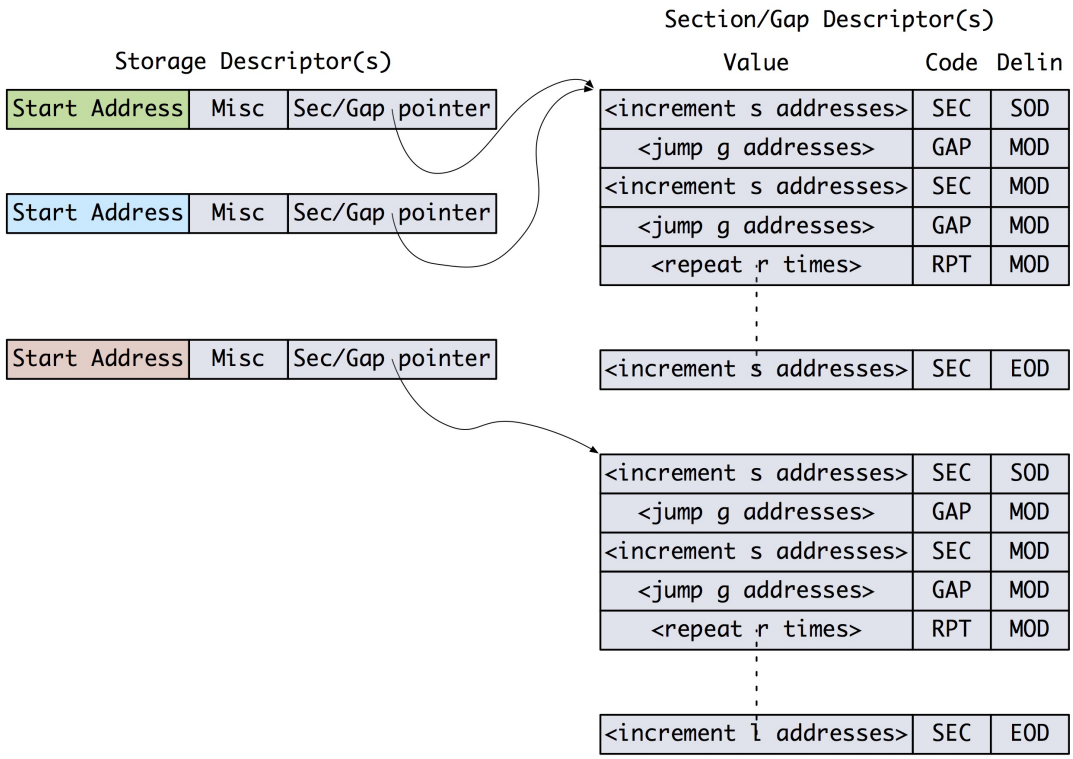


Figure 8.5 Storage Descriptor

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.

8.2 PE Operations

8.2.1 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.

8.2.2 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.

8.2.3 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 (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

9

DETAILED SYSTEM DESCRIPTION

A detailed flow diagram and block diagram of the sub-system column can be seen in figures 9.1 and 9.2 respectively.

9.1 Manager

9.1.1 Operation Decode

In figure 9.2, 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

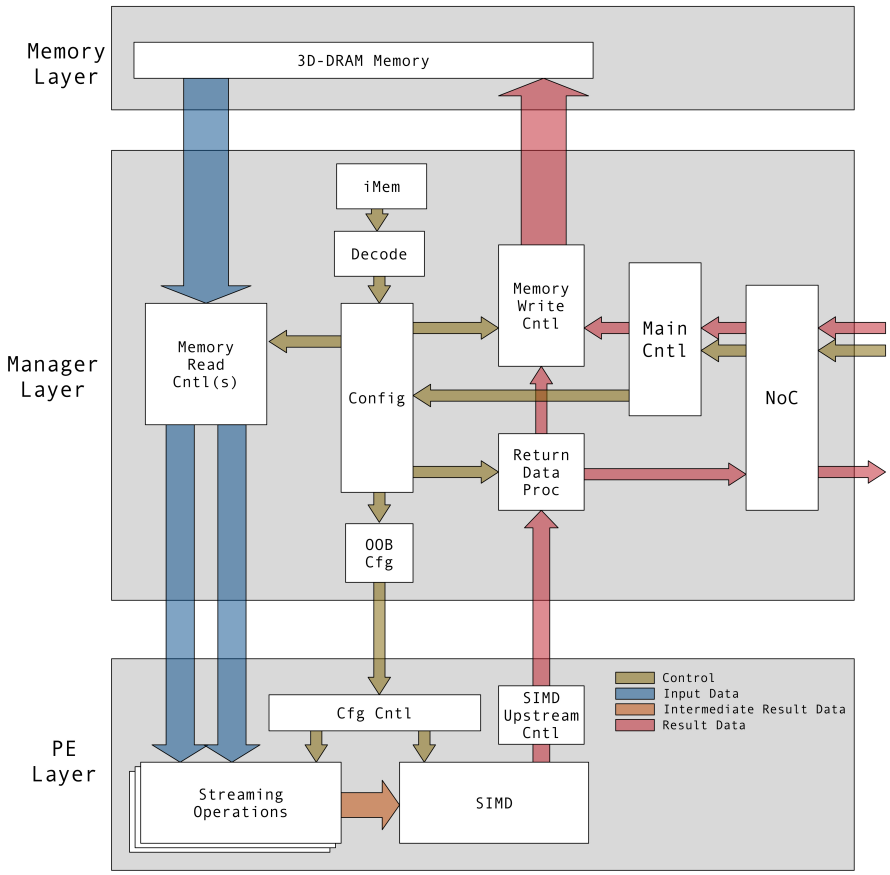


Figure 9.1 Sub-System Column (SSC) Flow Diagram

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.

9.1.2 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.

9.1.3 Memory Read Controller

The ANe states and connection weights can be interleaved or linearly. When the data is stored linearlya) stored for a group of ANes in an interleaved manner such that when memory is read, weight data and/or ANe states are spread across all the execution lanes for individual ANe state caclutation or b) linearly in which case the.

9.1.4 Result Data Processor

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.

9.1.5 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.

9.2 Processing Engine

9.2.1 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.

9.2.2 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.

9.2.3 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.

9.2.4 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.

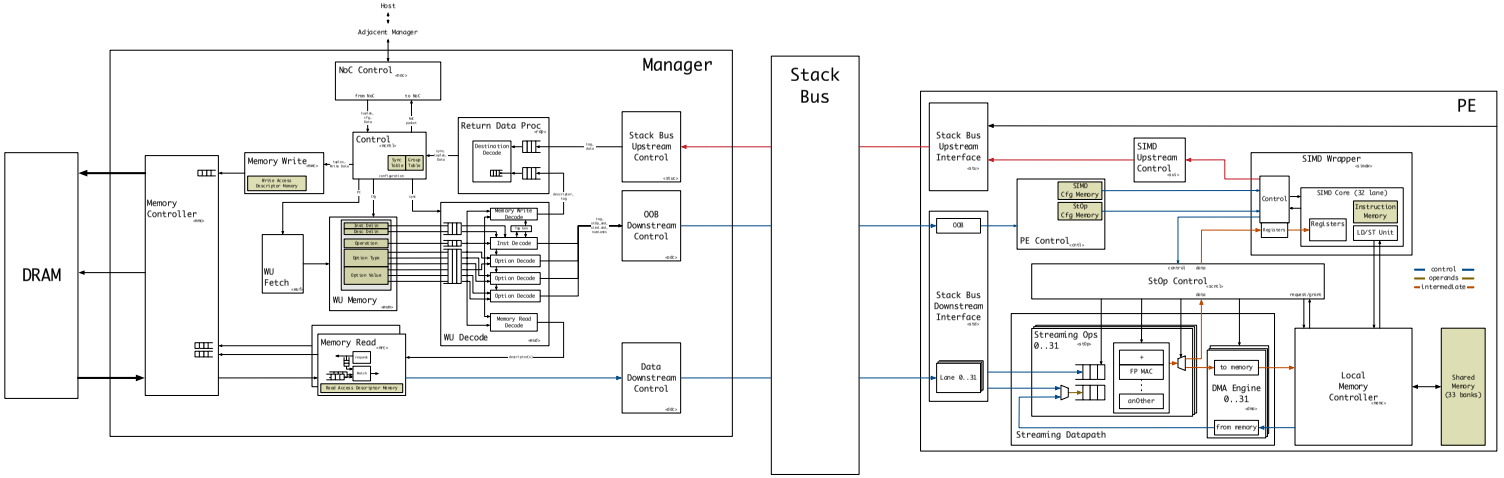


Figure 9.2 Sub-System Column (SSC) Block Diagram

CHAPTER

10

RESULTS

The objectives of this work was to design a system able to accelerate ANNs in customer facing systems implemented at the edge. Given that these systems cannot effectively utilize SRAM, the main objective was to demonstrate a system that can operate efficiently using 3D-DRAM.

The system decodes instructions, sends configuration to various functions, pre-fetchs and pipelines data. This parallelism allows the system to constantly stream data whilst results from previous operations are being operated on.

To demonstrate such a system, this work targeted 3DIC technology including 3D-DRAM. This work proposes that if a system can be purely in 3DIC, the system can take advantage of the benefits of 3DIC which includes reduced energy, area and high bandwidth. In addition, this work proposes that given the system is 3DIC, then a customized DRAM would provide a significant bandwidth boost over

typical implementations using standard DRAM.

The target technology node was 28nm because its the technology node employed for some recent GPUs and other ASICs such as [Jou17]. As a 28nm was not available to this team, the design was synthesized using an available 65nm technology node and then scaled to 28nm.

The primary control and datapaths of the system have been simulated in a system verilog environment. Initial synthesis timing closure at a frequency of 500 MHz is complete.

Initial place and route for the Manager and PE are shown in figure 10.1. The area contribution of each block within the Manager and PE can be seen in table 10.1.

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 testcases were based on layers similar to CONV2 and FC-7 from [Kri12] and represent a pre-synaptic fanin of 225 and 4000 respectively. Additional testcases were employed representing pre-synaptic fanins of 294, 300, 500 and 1000. Both locally connected (CONV) and fully connected (FC) type fanins were tested. The results showing sustained average bandwidth can be seen in table 10.3.

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 DRAM energy dissipation calculated from [Teza]. 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. The estimated overall power along with per block contribution are shown in table 10.2.

As bus efficiency is the main metric, table 10.3 shows sustained average bandwidth over the fanin testcases.

Table 10.1 Area Contribution

Block Name	Instances	Percentage Contribution
Memory Controller	1	15.0 %
NoC	1	7.1 %
Read Control	2	53.1 %
Write Control	1	7.4 %
Instruction Proc	1	1.6 %
Return Data Proc	1	1.6 %
Misc	1	14.2 %

(a) Manager

Block Name	Instances	Percentage Contribution
Operation Decode	1	3.4 %
Return Data Control	1	1.5 %
SIMD Control	1	8.1 %
SIMD	1	19.3 %
Streaming Operations	32	43.3 %
Streaming Op Control	1	2.1 %
Local Memory + Control[a]	1	17.7 %
Misc	1	4.6 %

(b) PE

[a] A small amount of scratchpad memory was provided between stOps and SIMD but in practice could be much smaller. It is not used in any of the fanin tests.

Table 10.2 Power Estimates

Technology Node	Clock Frequency	Total Expected Power	Testcase
28nm	500 MHz	64W	CONV-294
28nm	700 MHz	88W	CONV-294

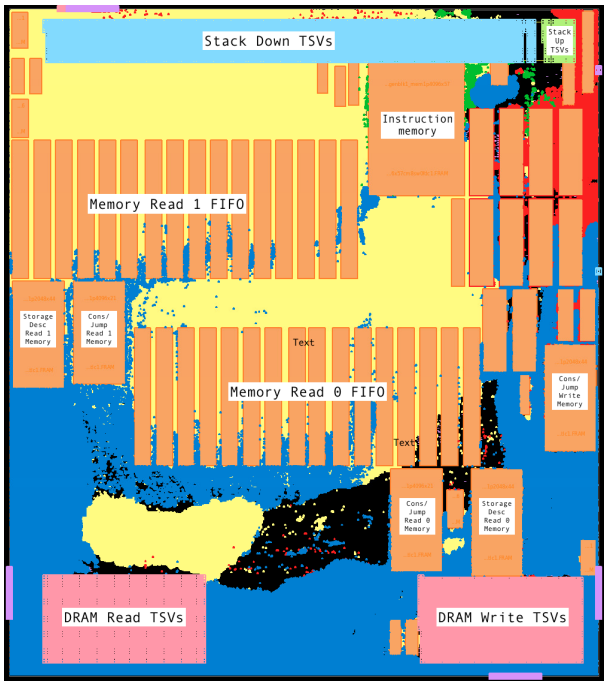
(a) Power Dissipation

Block Name	Percentage Contribution
Manager	66.6 %
PE	28.0 %
DRAM	2.5 %
DRAM TSVs	1.8 %
Stack Bus TSVs	1.2 %

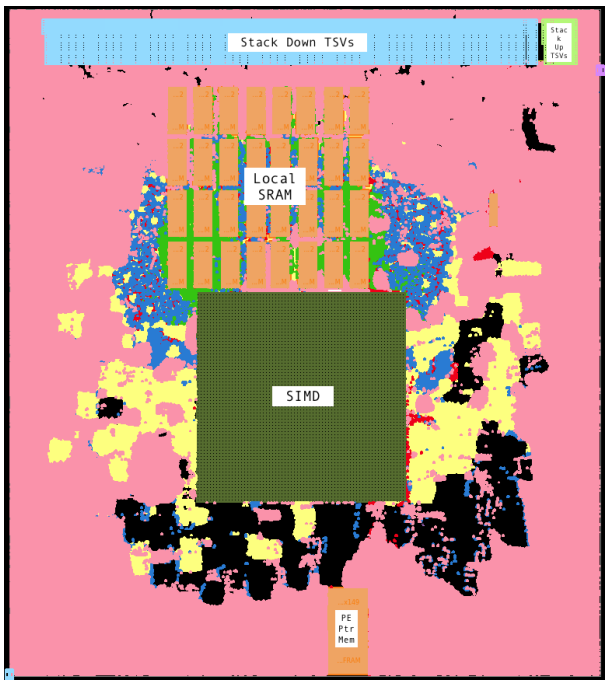
(b) Power Contribution

Table 10.3 Fanin Bandwidth Tests

	Average Bandwidth At Frequency	
Test	500 MHz	700 MHz
CONV2 [Kri12]	∼ 22 Tbit/s	∼ 30 Tbit/s
CONV-294	∼ 23 Tbit/s	∼ 31 Tbit/s
CONV-300	∼ 25 Tbit/s	∼ 34 Tbit/s
CONV-500	∼ 26 Tbit/s	∼ 38 Tbit/s
CONV-1000	∼ 30 Tbit/s	∼ 41 Tbit/s
FC-350	∼ 26 Tbit/s	∼ 36 Tbit/s
FC-500	∼ 27 Tbit/s	∼ 38 Tbit/s
FC-1000	∼ 30 Tbit/s	∼ 42 Tbit/s
FC-7 [Kri12]	∼ 31 Tbit/s	∼ 43 Tbit/s



(a) Manager



(b) PE

Figure 10.1 Manager and PE Die layouts

CHAPTER

11

CONCLUSIONS AND FUTURE WORK

11.1 Conclusion

11.2 Future Work

BIBLIOGRAPHY

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

[BGO17] Bamberg, L. & Garcia-Ortiz, A. "High-Level Energy Estimation for Submicrometric TSV Arrays". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**.10 (2017), pp. 2856–2866.

[CH06] Carnevale, N. & Hines, M. *The NEURON Book*. Cambridge University Press, 2006.

[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.

[Teza] *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.

[Tezb] *Evolving 2.5D and 3D Integration*. Tezzaron Semiconductor.

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

[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.

[ITR15] ITRS. *International Technology Roadmap for Semiconductors 2.0, Interconnect*. 2015.

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

[Jac07] Jacob, B. et al. *Memory Systems: Cache, DRAM, Disk*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

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

[Kim16a] Kim, D. et al. "Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory". *Proceedings of ISCA*. Vol. 43. 2016.

[Kim16b] Kim, S. W. et al. “Ultra-Fine Pitch 3D Integration Using Face-to-Face Hybrid Wafer Bonding Combined with a Via-Middle Through-Silicon-Via Process”. *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*. 2016, pp. 1179–1185.

[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).

[Nvi] Nvidia®. *NVidia Tesla P100 Datasheet*. `http://images.nvidia.com/content/tesla/pdf/nvidia-tesla-p100-datasheet.pdf`. Accessed: 2017-12-29.

[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.

[Sch17] Schabel, J. C. *Design of an Application-Specific Instruction Set Processor for the Sparse Neural Network Design Space*. ECE Dept., North Carolina State University, Box 7911, Raleigh. Box 7911, Raleigh, NC, 27695-7911, 2017.

[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.

APPENDICES