Multi-ANN Edge System based on a Custom 3DIC DRAM

Lee B. Baker, Paul Franzon *Fellow, IEEE*

***Abstract*—Machine Learning in the form of Deep Neural Networks (DNN) 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(AN). With popular DNNs, the artificial neural network (ANN) is often formed from tens of layers of ANs with each layer containing many ANs. 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 ANs and within the network, each AN can have hundreds, even thousands of feeder or pre-synaptic ANs.**

There have been implementations that use different number formats from double precision floating to eight bit integers, but in all cases these useful ANNs require significant memory requirements to store the connection weights (parameters) therefore requiring Dynamic Random Access memory (DRAM) to store the AN 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 amout of reuse of the weights connecting ANs and thus they can take advantage of local memory (SRAM).

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, 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 opportuities 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 AN. In additon, 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".

***Index Terms*—machine learning, edge system, DNN, CNN, neural network**

L. B. Baker, and P. Franzon are with the Department of Electrical and Computer Engineering, North Carolina State University, 2410 Campus Shore Dr., Raleigh NC 27606 Tel/Fax: 919-515-5460/5523 Email: lbbaker@ncsu.edu, jasteve4@ncsu.edu, paulf@ncsu.edu

Manuscript received Month Day, 2016; revised Month Day, 2016.

I. Introduction

USEFUL DNNs often require hundreds of thousands of ANs. Within the network, each AN can have hundreds of feeder (pre-synaptic) ANs. With popular DNNs, there are often tens of layers.

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

When these ANNs are required to be solved in fractions of a second, the processing and memory bandwidth becomes prohibitive.

In most cases, Graphics processing Units (GPU) are used to implement large ANNs. In many ANN architectures, such as Convolutional ANNs (CNN), they are quite effective. However, we should not forget they are not optimized purely for NN processing and are restricted by available SRAM and they are power hungry. These limitations limit the effectiveness of GPUs.

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

Another technology that has been considered over the last decade is 3D integrated circuit technology (3DIC). This 3DIC technology stacks multiple die together to form a system-on-chip with potentially disparate technology for each die in the stack. By staying within the die footprint, 3DIC technology promises high connectivity and consequently high bandwidth along with lower power all within a smaller footprint.

As a metric, this work asumes that any useful DNN will employ 100's of thousands of ANs. Although there is a lot of debate regarding number formats for ANNs, this work also assumes single-precision floating point. Assuming an ANN with 250K neurons and an average fanin to each AN of 2000, a system employing 10 ANNs for various disparate functions and an average processing time of 10 ms suggests a average bandwidth of 16 Tbit/s (1).

$$\begin{aligned}\text{Average Bandwidth} &= \sum_{\mathbf{n}=0}^{\bar{\mathbf{N}_\mathbf{n}}} \left(\frac{\bar{\mathbf{N}_\mathbf{a}} \cdot \bar{\mathbf{C}_\mathbf{p}} \cdot \bar{\mathbf{b}_\mathbf{w}}}{\bar{\mathbf{T}_\mathbf{p}}}\right) \\ &= \sum_{\mathbf{n}=0}^{9} \left(\frac{250 \times 10^4 \cdot 2 \times 10^3 \cdot 32}{10 \times 10^{-3}}\right) \\ &= 16\,\text{Tbit}\,\text{s}^{-1} \end{aligned} \quad (1)$$

where $\mathbf{N}_n$ is the number of ANNs
$\mathbf{N}_a$ is the average number of ANs
$\mathbf{C_p}$ is the average number of connections
and $\mathbf{T_p}$ is the processing time

Regardless of the combination of ANNs, this work suggests that these edge systems will require memory bandwidth of the order of 10's of Tbit/s.

This work demonstrates a system that is able implement multiple useful sized DNNs whilst maintaining an average memory bandwidth of >30 Tbit/s. This is made possible by ensuring the system stays within the die stack footprint of a typical 3DIC DRAM . This work removes a reliance on SRAM to achieve high performance thus allowing the proposed design to be utilized in edge applcations when processing multiple disparate ANNs at or near real-time. Although not optimized for specific ANNs, such as CNNs, this work demonstrates the potential for real-time performance at the edge when implementing fully connected DNNs or other similar ANNs such as LSTM.

II. System Description

A. ANN System

The primary objectives of this work was to a) consider systems that are unable to take advantage of memory reuse opportunities and therefore not able to achieve high performance using local SRAM to store ANN parameters or the ANN input, b) acknowedge that DRAM is required for storage of ANN parameters, c) that many edge devices will likely apply many disparate ANNs to perform various system functions, and d) it is assumed that many edge applications will have space and power limitations.

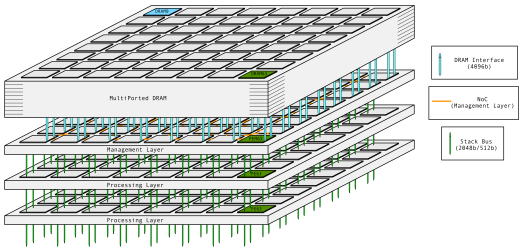


Fig. 1: 3DIC System Stack

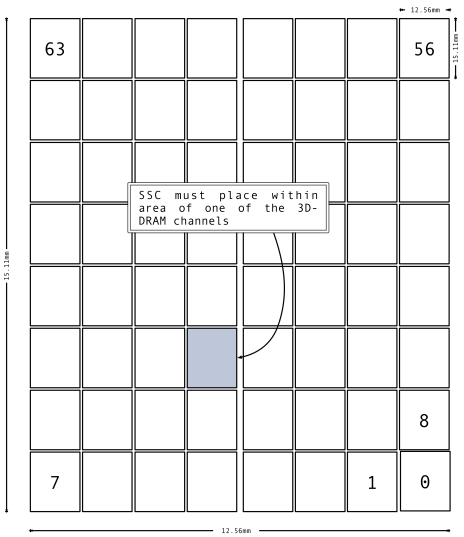


Fig. 2: DRAM Physical Interface Layout showing area for SSC

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

The 3DIC system die stack (figure 1) includes the 3D-DRAM with a system manager below and one or more processing layers below the manager.

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

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

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

This work targeted a baselne system with:

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

The work includes customizing the interface to a 3D-DRAM, researching data structures to describe storage of ANN parameters, designing a memory manager with micro-coded instructions and a processing engine (PE) layer. The system is designed such that a sub-system, known as a sub-system column (SSC) operates on one of these disjoint memories within the 3D-DRAM (see figure 2).

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

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

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

2) Manager Layer: The Manager block is the main controller in the system. The operations required to process an ANN are formed from individual instructions which are decoded by the Manager. These instructions include descriptors to describe memory read operations, processing engine operations and memory write operations. The manager

reads these system instructions from an instruction memory, decodes the instruction and configures the various blocks in the system. The configuration includes:

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

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

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

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

A control and data flow diagram of the stack showing the 64 sub-system columns can be seen in figure 3.

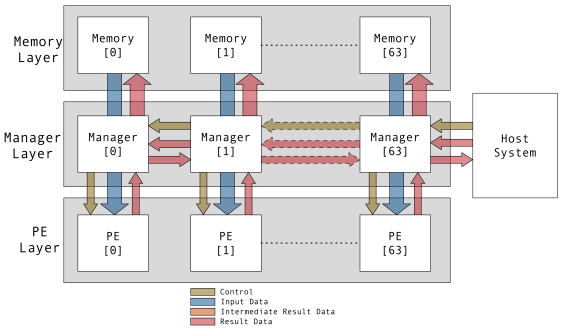


Fig. 3: System Flow Diagram

III. System Operations

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

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

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

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

The managers primary responsibility is:

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

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

The PE has three major blocks:

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

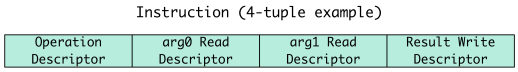


Fig. 4: Instruction 4-tuple

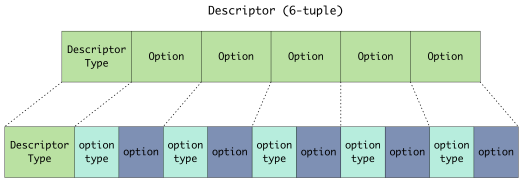


Fig. 5: Descriptor 6-tuple

A. Manager Operations

1) Instructions: The instructions communicate:

- To the Manager
  - ROI Storage descriptor
  - Parameter/Weight Storage Descriptor
    - Broadcast or Vectored
  - Result write storage descriptor
    - include descriptors for all destination managers
- To the PE
  - stOp operation
  - SIMD operation
  - Number of active lanes
  - Operand Vector length

Instructions include information to control the above operations.

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

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

An instruction will typically have four descriptors:

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

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

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

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

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

These option elements are a two-tuple with option and associated value. In figure 5 we see the format of a 6-tuple descriptor.

2) Accessing of Pre-synaptic AN states and connection weights: As was discussed previously, the ANN input and configuration is stored in main DRAM memory. A part of the research is determining how to store the ANN input and parameters in such a way to effectively make use of main DRAM bandwidth. To provide parameters for the up to 32 execution lanes within the PE, we store the AN parameters in consecutive address locations. With one read to the DRAM, we access 128 words. This provides four weights for each of the 32 ANs being processed. These weights are sent to each

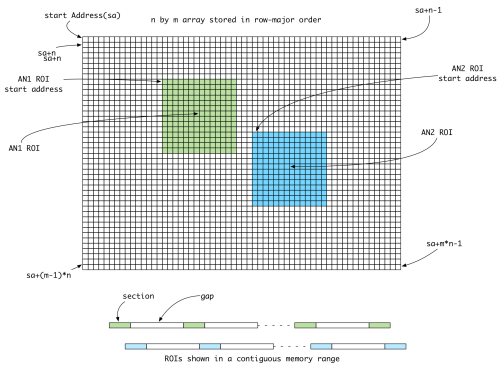


Fig. 6: ROI Storage

lane of the PE over four cycles. We will discuss memory efficiency later, but by taking advantage of the multiple DRAM banks along with pre-fetching and buffering, we are able to achieve relatively high efficiency of the available maximum bandwidth.

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

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

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

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

In figure 6, an input to a ANN layer in the form of a 2-D array along with the ROI of two ANs.

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

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

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

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

This storage descriptor contains, amongst other things the start address of the ROI and a pointer 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 7 shows the structure of the storage descriptor. The SOD, MOD and EOD are used to delineate each descriptor in memory and stand for start-of-descriptor, middle-of-descriptor and end-of-descriptor.

3) 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

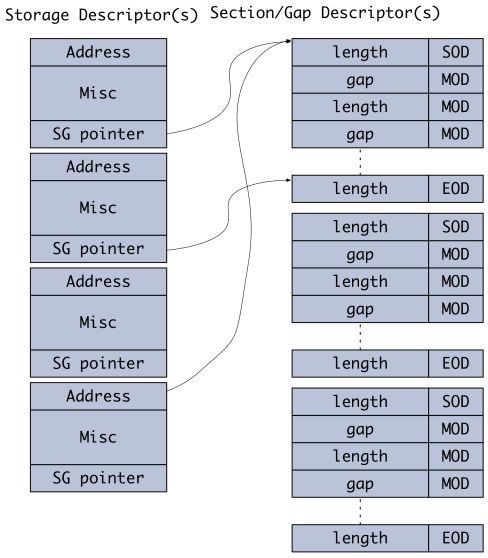


Fig. 7: Storage Descriptor

that's not always true.

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

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

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

B. PE Operations

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.

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

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

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

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

IV. Suggested DRAM Customizations

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

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

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

Figure 8 shows a block diagram of a typical DRAM.

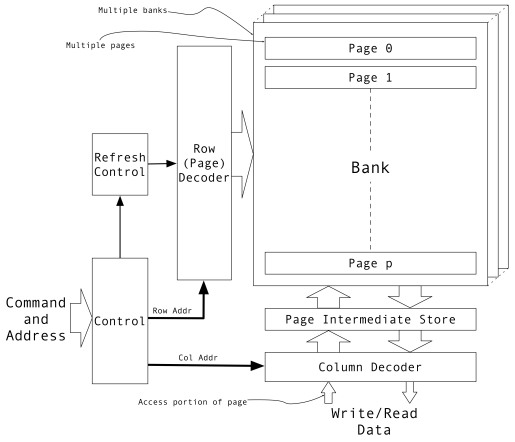


Fig. 8: Typical DRAM Block Diagram

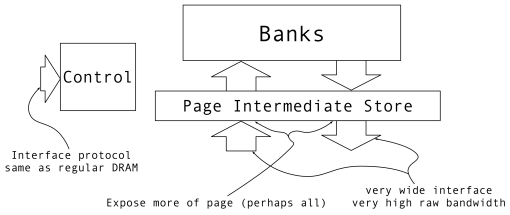


Fig. 9: Exposing more of the DRAM page

A. *Expose more of the Page*

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

Without the limitations of having to transfer data beyond the chip stack, this work suggests exposing a larger portion of the page over a very wide bus. By staying within the 3D footprint, this bus can be implemented using fine pitch through-silicon-vias. (see figure 9).

B. *DRAM Write Mask*

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

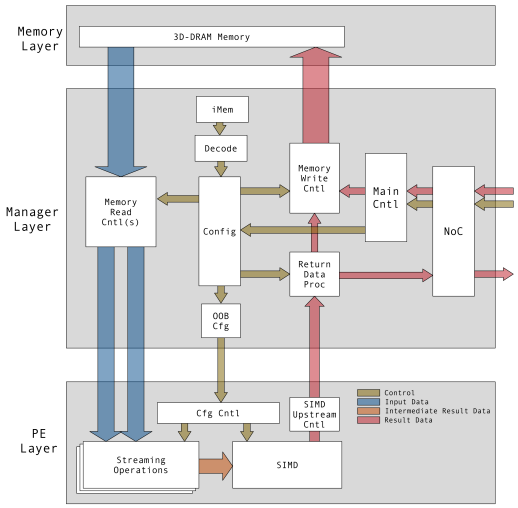


Fig. 10: Sub-System Column (SSC) Detailed Flow Diagram

V. Detailed System Description

A detailed flow diagram and block diagram of the sub-system column can be seen in figures 10 and 11 respectively.

A. *Manager*

1) Operation Decode: In figure 11, 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 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.

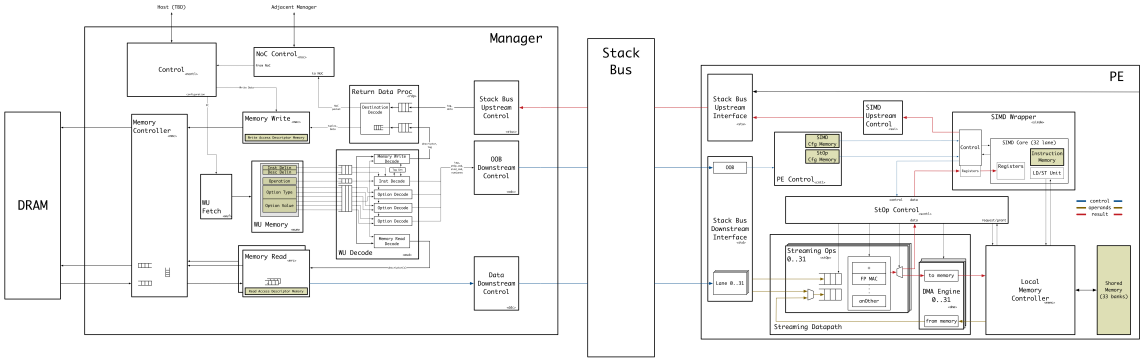


Fig. 11: Sub-System Column (SSC) Detailed Block Diagram

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.

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.

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

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

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

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

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

into the DRAM write protocol.

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

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

B. Processing Engine

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.

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.

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.

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.

VI. Results

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

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

implementations using standard DRAM. This work targeted the Tezzaron DiRAM4 3D-DRAM.

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

The target technology node was 28nm. This was chosen because its the technology node employed for some recent GPUs and other ASICs such as [1]. The design was synthesized using an available 65nm technology node and then scaled to 28nm to demonstrate fitting within the 3DIC footprint.

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

As mention previously (1), to process multiple useful sized ANNs requires a sustained bandwidth to the PE of the order of ten's of Tbit/s.

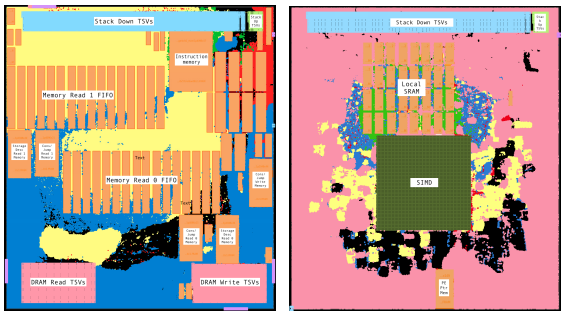
The Manager and PE have been place and routed as shown in figure 12. Although the design is yet to close timing, the parasitics were extracted from these layouts and simulated against a group of operations. The simulation generated an activity file which was then used by the Synopsys® Primetime-PX™ power analysis tool to obtain power and bandwidth estimates. Those estimates were used to estimate power dissipation for operating frequencies of 500 MHz and 700 MHz which are shown in table I.

TABLE I: Simulation-based estimates

Tech	Freq	Usable DRAM Bandwidth	Expected Power	FLOPS
28	500 MHz	$\sim$ 46 Tbit/s	53W	2.0 TFLOP/s
28	700 MHz	$\sim$ 64 Tbit/s	73W	2.8 TFLOP/s

VII. Conclusions

There have been many attempts to accelerate ANNs. Many have shown excellent performance mainly when implementing CNNs. The improvement mostly comes from the ability to hold weights



(a) Manager (b) PE

Fig. 12: Manager and PE Die layouts

and/or AN activations in local SRAM and perform multiple operations whilst that data is held in local SRAM. The ability to hold data in local SRAM is often due to pooling of batch requests or storing of CNN kernel weights or input Regions-of-Interest. When considering applications where this local storage cannot be used effectively, all these implementation suffer large degradation in performance.

This work considers edge applications where a system is processing single requests into a disparate set of ANNs. The assumpiton is that local SRAM is no longer effective and performance is based on DRAM bandwidth. This work considers 3DIC technology and the performance benefits it provides. In addition, a customized 3D-DRAM is proposed. The customized 3D-DRAM combined with a design based on custom instructions and operation descriptors allows the system to achieve high levels of memory bandwidth efficiency.

There is no doubt CNN accelerators combined with batch processing achieve a performance that is difficult to better, but applying these systems to this works target application exposes the DRAM bandwidth limitations.

This work demonstrates a 3D-DRAM system that at the surface has a relatively low FLOPs, but the target applcation is memory bound and under these circumstances this work is potentially 10-100X better that existing ASIC/ASIPs or GPUs.

VIII. Further Work

This work does not ut a high level of importance to the processing engine. The choice of using single-precision floating point format was in some part convenient. In fact, half-precision might have been a better choice especially as there is a level of acceptance that lower precision is acceptable in ANN inference. Therefore, further work should include :

- Manager changes to support half-precision format
  - supports 16-bit FP would require additional muxing logic when directing words in the wide DRAM bus to execution lanes, but the bulk of the design should remain relatively intact.
- PE Single-Precision
  - modifying the PE to support 16-bit FP would be trivial
- PE pipeling
  - Adding layers of processing to the PE would effectively increase the operations per second
    - using a reduction array would effectively double the FLOPs
    - looking at storing weights and inputs would be required and the manager would likely be customized to the PE, but the belief is the complexity should be similar
- Systolic PE
  - systolic arrays have shown efficacy when used in server applications such as TPU, but its not clear how effective they would be in an edge application
    - consider how to effective feed a systolic array
    - consider types of systolic array

References

[1] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," *arXiv preprint arXiv:1704.04760*, 2017.

[2] Y. Liu, W. Luk, and D. Friedman, "A compact low-power 3d i/o in 45nm cmos," in *2012 IEEE International Solid-State Circuits Conference*. IEEE, 2012, pp. 142–144.

[3] R. Patti, "2.5 d and 3d integration technology update," *Additional Papers and Presentations*, vol. 2014, no. DPC, pp. 1–35, 2014.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[5] ——, "Imagenet classification with deep convolutional neural networks," http://image-net.org/challenges/LSVRC/2012/supervision.pdf, accessed: 2016-08-30.

[6] S. Galal and M. Horowitz, "Energy-efficient floating-point unit design," *IEEE Transactions on Computers*, vol. 60, no. 7, pp. 913–922, 2011.

[7] J. Schabel, J. B. Park, R. W. Davis, and P. D. Franzon, *Predictive energy-Per-Op scaling for exploring the design space*, ECE Dept., North Carolina State University, Box 7911, Raleigh, Box 7911, Raleigh, NC, 27695-7911, 2014.

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

[9] *DiRAM4-64Cxx Cached Memory Subsystem*, Tezzaron Semiconductor, July 2015, rev. 0.04.

[10] Q. Qiu, Q. Wu, M. Bishop, R. E. Pino, and R. W. Linderman, "A parallel neuromorphic text recognition system and its implementation on a heterogeneous high-performance computing cluster," *IEEE Transactions on Computers*, vol. 62, no. 5, pp. 886–899, 2013.

[11] C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Move evaluation in go using deep convolutional neural networks," *arXiv preprint arXiv:1412.6564*, 2014.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[13] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[14] W. E. Lillo, D. C. Miller, S. Hui, and S. H. Zak, "Synthesis of brain-state-in-a-box (bsb) based associative memories," *IEEE transactions on Neural Networks*, vol. 5, no. 5, pp. 730–737, 1994.

[15] H. Esmaeilzadeh, F. Farzan, N. Shahidi, S. Fakhraie, C. Lucas, and M. Tehranipoor, "Nnsp: embedded neural networks stream processor," in *48th Midwest Symposium on Circuits and Systems, 2005.* IEEE, 2005, pp. 223–226.

[16] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3d memory," in *Proceedings of ISCA*, vol. 43, 2016.

[17] S. Kyo and S. Okazaki, "Imapcar: A 100 gops in-vehicle vision processor based on 128 ring connected four-way vliw processing elements," *Journal of Signal Processing Systems*, vol. 62, no. 1, pp. 5–16, 2011.

[18] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable

dataflow processor for vision," in *Cvpr 2011 Workshops*. IEEE, 2011, pp. 109–116.

[19] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 682–687.

[20] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49, no. 4. ACM, 2014, pp. 269–284.

[21] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2016, pp. 262–263.

[22] P. D. Franzon, L. Baker, S. Dey, W. Li, and J. Schabel, *Hardware Acceleration of Sparse Cognitive Algorithms*, North Carolina State University, March 2016, aFRL.

[23] Techpowerup, "Nvidia tesla k20c," https://www.techpowerup.com/gpudb/564/tesla-k20c, accessed: 2016-09-08.

[24] W. Tech, "Nvidia : Pascal is 10x faster than maxwell," http://wccftech.com/nvidia-pascal-gpu-gtc-2015/, accessed: 2016-09-08.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 2016/09/21 1989. [Online]. Available: http://dx.doi.org/10.1162/neco.1989.1.4.541

[27] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[28] Y.-F. Tsai, F. Wang, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Design space exploration for 3-d cache," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 4, pp. 444–455, 2008.

[29] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[30] G. V. der Plas, P. Limaye, I. Loi, A. Mercha, H. Oprins, C. Torregiani, S. Thijs, D. Linten, M. Stucchi, G. Katti, D. Velenis, V. Cherman, B. Vandevelde, V. Simons, I. D. Wolf, R. Labie, D. Perry, S. Bronckers, N. Minas, M. Cupac, W. Ruythooren, J. V. Olmen, A. Phommahaxay, M. de Potter de ten Broeck, A. Opdebeeck, M. Rakowski, B. D. Wachter, M. Dehan, M. Nelis, R. Agarwal, A. Pullini, F. Angiolini, L. Benini, W. Dehaene, Y. Travaly, E. Beyne, and P. Marchal, "Design issues and considerations for low-cost 3-d tsv ic technology," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 293–307, Jan 2011.

Lee B. Baker received a B.S. degree in Electrical Engineering from Brighton Polytechnic, UK, an M.S. degree in Electrical Engineering from Villanova University, USA and an M.B.A. degree from North Carolina State University, USA in 1983, 1994 and 2009 respectively. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department at North Carolina State University. His current research interests include acceleration of artificial neural networks.

Paul Franzon is currently a Distinguished Alumni Professor of Electrical and Computer Engineering at North Carolina State University. He earned his Ph.D. from the University of Adelaide, Adelaide, Australia in 1988. He has also worked at AT&T Bell Laboratories, DSTO Australia, Australia Telecom and three companies he cofounded, Communica, LightSpin Technologies and Polymer Braille Inc. His current interests center on the technology and design of complex microsystems incorporating VLSI, MEMS, advanced packaging and nano-electronics. He has lead several major efforts and published over 200 papers in these areas. In 1993 he received an NSF Young Investigators Award, in 2001 was selected to join the NCSU Academy of Outstanding Teachers, in 2003, selected as a Distinguished Alumni Professor, and received the Alcoa Research Award in 2005. He served with the Australian Army Reserve for 13 years as an Infantry Soldier and Officer. He is a Fellow of the IEEE.