

# 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 amount 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 NN 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 process-

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.

ing 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 portions of the ANN in SRAM. A further assumption is that the target edge devices do not include opportunities to perform batch processing.

This work uses the DRAM as the primary processing storage and employs minimal or no SRAM for the processing of the AN. 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. 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

## 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 NNs. In many NN architectures, such as Convolutional NNs (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 will limit the effectiveness of GPUs regardless of what we might hear from the GPU community ("declare an interest").

Much of the NN application specific (ASIC/ASIP) research has focused on taking advantage of the performance and ease of use of Static Random Access Memory or SRAM. These implementations can be shown to be effective with specific NN architectures (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.

This work demonstrates a system that is able implement multiple useful sized DNNs whilst staying within the die stack footprint of a typical 3DIC DRAM. This work also removes a reliance on SRAM to achieve high performance thus allowing the proposed design to be utilized in edge applications 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 when implementing fully connected DNNs or other such ANNs such as LSTM, at the edge.

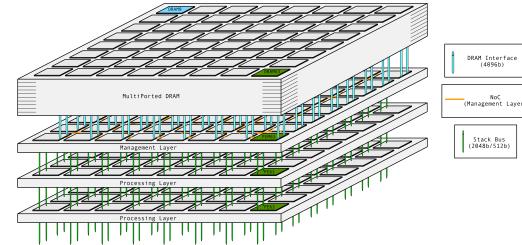


Fig. 1: 3DIC System Stack

## 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) acknowledge that DRAM is required for storage of ANN parameters and 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.

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 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 co-exist within the same 3D footprint?

This work targeted a baseline 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 targeted 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

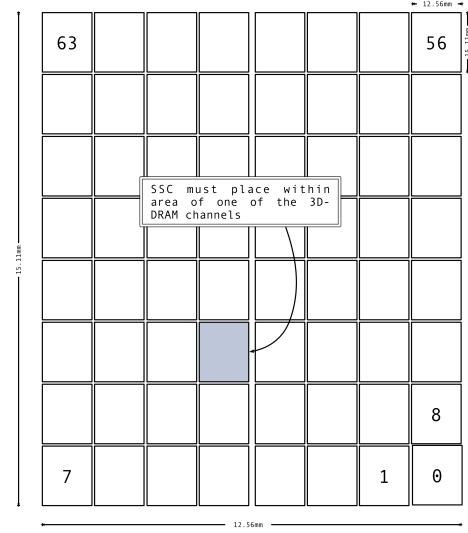


Fig. 2: DRAM Physical Interface Layout showing area for SSC

- 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 operation on the operand data streamed from the manager. In our 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

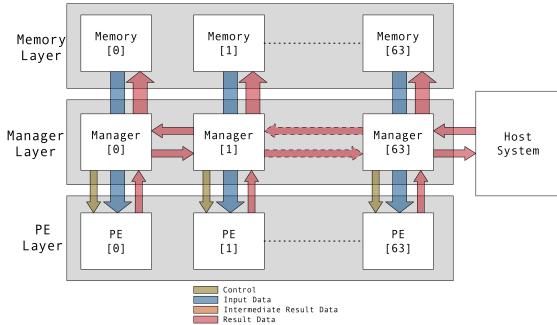


Fig. 3: System Flow Diagram

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 subsystems, often neuron activation data must be shared between these SSCs. In our case, 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.

### III. SYSTEM OPERATIONS

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

- Inform the Manager and PE which operation is to be performed
- Tell the manager to access the states of the feeder neurons
- Tell the manager to access the weights of the connections from the feeder ANs
- Provide the feeder neuron weights and states to the processing engine
- Tell the manager (or PE) where to store the resulting AN state back to memory

This work has researched an instruction architecture to describe these operations so they can be interpreted by the manager.

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

The managers primary responsibility is:

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

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

The PE has three major blocks:

- streaming operation function
- SIMD
- DMA/local memory controller

#### A. Manager Operations

1) *Instructions:* Lets first review what our instruction communicates:

- 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

Our instructions have to include information to control the above operations.

We partition the instruction into sub-instruction we call descriptors. These descriptors contain the information to control the various operations associated with the processing of a group of ANs.

For want of a better word, we might consider this a variable length instruction word or VLIW.

Instruction (4-tuple example)			
Operation Descriptor	arg0 Read Descriptor	arg1 Read Descriptor	Result Write Descriptor

Fig. 4: Instruction 4-tuple

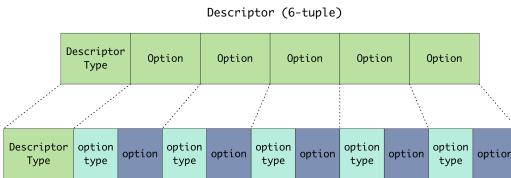


Fig. 5: Descriptor 6-tuple

Remember, the group size is related to the number of execution lanes which for our baseline system is 32. So a group can be anywhere from 1-32. It should be said that unless we consistently have group sizes approaching 32 the system performance will be poor.

So lets look at the format of our instruction.

We typically have four descriptors:

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

Note: We will refer to an operand stream as an argument.

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

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

Now earlier we explained that 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 NN.

Now the input can be represented in the form of a 2-D array of AN states. For the sake of generality, We will even consider the input array elements as AN states.

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

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

Now we mentioned 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. If we consider the input array stored in row-major order, we can see 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

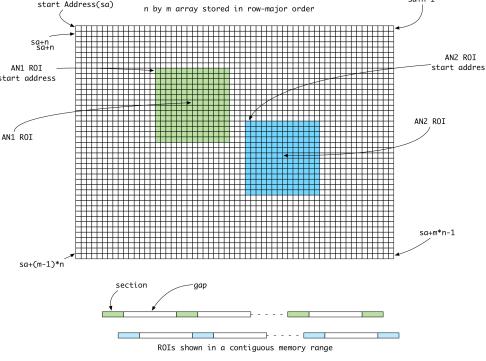


Fig. 6: ROI Storage

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.

We solve this read efficiency problem by again taking advantage of the DRAM banks and pages.

But how do you describe these ROI storage locations?

Now although disparate groups of ANs may have a different start addresses for their ROI, we see commonality in the ROI section lengths and gaps. So for each AN group, we store that group's ROI starting address, but we point to a common set of section length/gaps. We term this structure storage descriptors.

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 our 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.

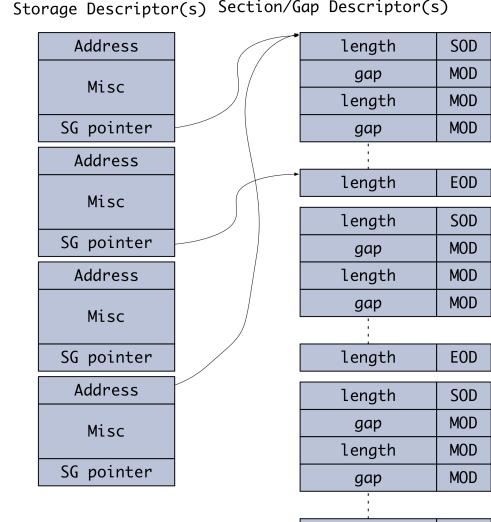


Fig. 7: Storage Descriptor

*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 we described earlier.

A significant difference we take advantage of is that for any given operation, we are writing far less than we are reading. For example, the ROI and parameters are usually vectors that will typically exceed 100 elements and in many cases much higher. When we finish the operation, we are writing, in almost all cases one word per lane. Now that sounds like writing back has a very small impact on performance but that's not entirely true.

When we write, we are 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 we write 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. We handle this with a network-on-chip (NoC) which we

will discuss later.

So how do we communicate result storage. Well, we use the same storage descriptor mechanism mentioned previously to describe where result data needs to be stored. However, the added complication is because the result may have to be written to other managers, we need to provide the storage descriptors for 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 we will focus on the AN multiply-accumulate followed by a ReLu activation function, we have built 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, we send two pointers 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.

Our 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, we also need to provide the number of lanes being processed for any given operation. In addition, we also send the length of the vector of operands being sent from the manager to the PE.

## IV. SUGGESTED DRAM CUSTOMIZATIONS

The memory bus is essentially a customized version of a typical DRAM interface.

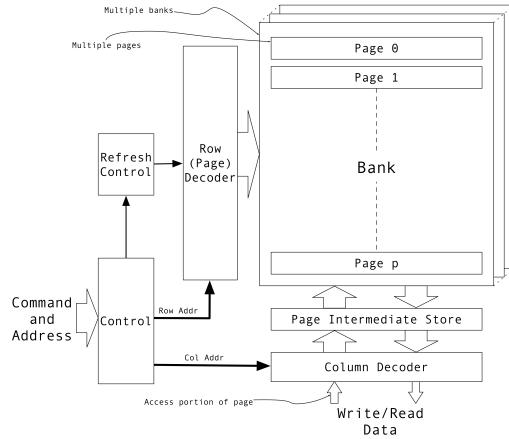


Fig. 8: Typical DRAM Block Diagram

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 is a block diagram of a typical DRAM.

### A. Expose more of the Page

We achieve 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, we expose a larger portion of the page and by staying within the 3D footprint we can transfer data over a very wide through-silicon-via interface (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

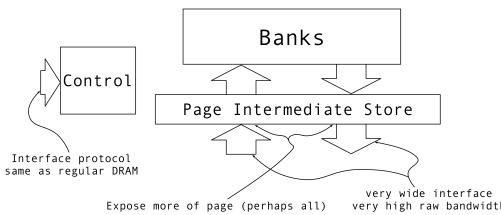


Fig. 9: Exposing more of the DRAM page

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, we often need to write a portion of the page back to memory. To avoid a read/modify/write, we propose adding a write data mask to the DRAM write path.

## V. DETAILED SYSTEM DESCRIPTION

A detailed block diagram of the sub-system column can be seen in figure 10.

### A. Manager

1) *Operation Decode*: In figure 10, 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 our 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.

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

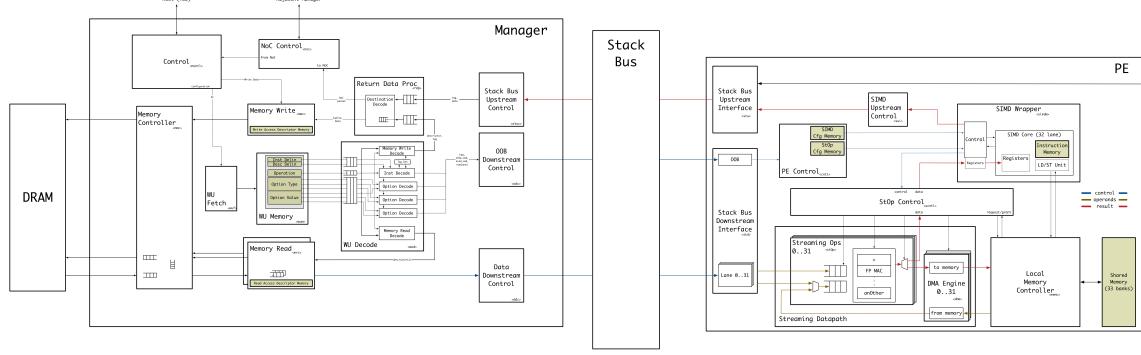


Fig. 10: Sub-System Column (SSC) Block Diagram

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 our baseline system is the Rectified Linear function.

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

To take advantage of the benefits of 3DIC technology, the area of the SSC needs to fit within the area of a single sub-memory in the 3D-DRAM. The synthesized and place and routed design needed to fit within the area shown in figure 2. The target technology node was 28nm. This was chosen because its the technology node employed for some recent GPUs and other ASICs such as [?]. The design was synthesized using a 65nm technology node and then scaled to 28nm.

## VII. CONCLUSIONS

**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.

**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.