Implementation of Hierarchical Mesh NoC Architecture based on Eyeriss v2 Accelerator

Karthikeyan Sugumaran
A53269850
ksugumar@eng.ucsd.edu

Imtiaz Ameerudeen
A53271622
iameerud@eng.ucsd.edu

University of California San Diego

***Abstract*— The Eyeriss DNN accelerator introduced a novel dataflow, the Row-Stationary Dataflow, for efficient processing of neural networks with optimal reuse of data. Eyeriss v2 expands on this concept with an architecture that includes a highly flexible on-chip network. Such an architecture supports easier data transfer to different Processing Elements (PEs) and adapts to different amounts of reuse and bandwidth requirements. The work here presents a SystemVerilog implementation of a scaled-down version of the same, simulated and synthesized on Xilinx Vivado.**

Keywords—Eyeriss, DNN, accelerator, dataflow architecture, on-chip network

I. Introduction

Convolutional neural networks have evolved significantly over time and are widely used today. The constantly increasing demand to support computationally intensive tasks such as training and inferring neural networks has led us to build Application-Specific Integrated Circuits instead of relying on general-purpose processors. While faster processing of neural networks is being reasonably addressed by SIMD engines such as GPUs, energy consumption has also become a crucial factor in deciding the viability of any hardware for a specific application.

Eyeriss[1] presents a novel dataflow, called row-stationary(RS), that minimizes energy spent in moving data by exploiting data reuse and maximizing processing element utilization. The authors demonstrate how RS dataflow has increased energy efficiency by at least 1.3x by conducting experiments with the proposed architecture on AlexNet CNN.

Over time, the general trend of neural network development has skewed toward more compact architectures and increased sparsity in filter weights. A follow-up work proposed by the same authors, the Eyeriss v2[2] architecture, presents a highly flexible network-on-chip (NoC), that can adapt to a wide range of bandwidth requirement depending on the kind of layer being processed, thus helping to exploit spatial data reuse to achieve high energy efficiency.

The work here implements a scaled-down version of the Eyeriss v2 architecture that includes 4 grouped clusters, each of which includes separate clusters for global buffers, routers, and processing elements. The routers are used to form a mesh network that helps in spreading out data in different ways according to their configuration. The implementation is in SystemVerilog and the models are synthesized on Xilinx Vivado to estimate frequency and performance.

The rest of the report is organized in the following way. Section II covers the architecture of Eyeriss v2 and details on the scaled-down version that is implemented in this project. Section III explains the implementation of the project in detail and gives information on all the components built as part of the design. Section IV covers the methodology undertaken to complete the project and discusses the results obtained. Finally, section V concludes the report and covers the future scope of this project.

II. Architecture

Eyeriss v2 is a systolic array-based architecture. It consists of an array of processing elements, each equipped with local scratchpads (SPad) for reading and storing data, and a multiply-and-accumulate unit (MAC) for computations. The PE array represents the heart of computation and forms the basis for processing different layers of a neural network. The local scratchpads represent a distributed memory system that reduces overall memory access latencies and helps exploit data reuse.

The architecture also has an additional level of memory hierarchy between the PEs and the off-chip DRAM. This is fulfilled by the Global Buffers (GLB) which are present outside the PEs. These buffers again serve as an internal data storage component to eliminate costly DRAM transfers. The communication between the PEs and GLBs are handled by routers, which are present as a network all over the architecture. Fig. 1 shows how the Eyeriss v2 architecture differs from the original Eyeriss architecture. The difference that stands out is the presence of a 2D mesh network that is present atop the layer of GLBs and PEs.

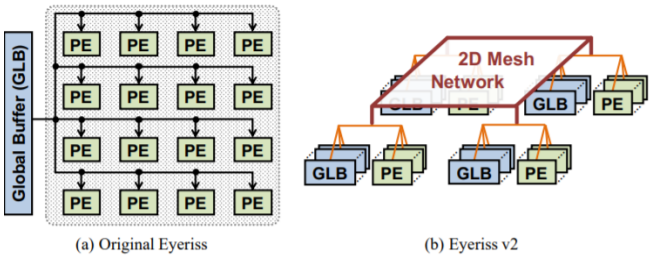

Fig. 1. Comparison of Eyeriss v2 with original Eyeriss

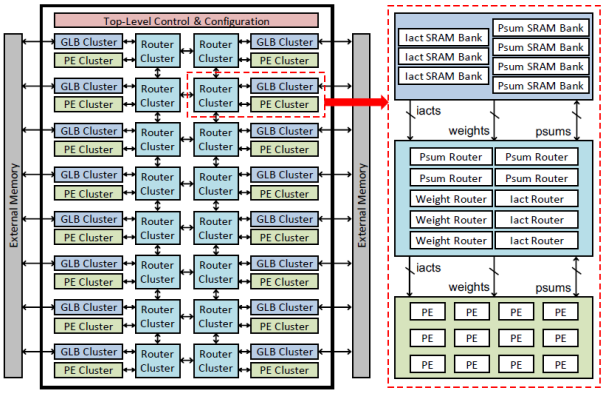

Fig. 2. Eyeriss v2 top-level Architecture

To support parallelism across the architecture and to reduce the overhead of dealing with a huge array of distinct routers for each PE and GLB pair, several PEs, GLBs, and routers are clustered together to form a grouped cluster. Each grouped cluster can be thought of as a smaller but a complete system of what the entire architecture represents. This grouped cluster is replicated to form an 8 x 2 array of clusters, which forms the entire structure of the design.

Finally, a top-level control and configuration unit is present and is used to configure the routers based on the kind of computation that is undertaken while processing a neural network. Fig. 2 shows the overall organization of all the components of the architecture.

Continuing with the architecture of Eyeriss v2, this section will describe each cluster introduced before. Since neural networks mainly deal with 3 different data types, the input activations(iact), the weights, and the partial sums(psum), the clusters are designed in such a way that they cater to the processing and routing of each data type. The original architecture of Eyeriss v2 contains a predetermined number of routers and global buffers in their respective clusters to handle each data type. The GLB cluster contains 3 SRAM banks for iacts and 4 SRAM banks for psums. The router cluster consists of 3 routers for iacts, 4 routers for psums, and in addition to these, it also contains 3 routers for weights. The original design does not contain SRAM banks for weights as the external DRAM is used to directly feed the PEs with the necessary weights for computation and are stored in their local SPads. The rest of this section details the architectural aspects implemented as part of this project.

The main motivation for this project is to explore and implement the hardware design concepts of neural network accelerators in general. Given how complicated it is to achieve a completely functional design of an architecture from its abstraction and verification of the model, our implementation deals with a scaled-down version of the Eyeriss v2 architecture. Our design includes RTL models of each component of the design described above and an array of 4 (2 x 2) grouped clusters that form a small mesh of PE, GLB, and router clusters. Fig. 3 shows all the components we have built and gives a top-level representation of how they are organized.

The dataflow of a neural network accelerator is a crucial aspect in defining its overall working and performance. Our implementation exclusively considers 2D convolution and the clusters built are used to perform this fundamental operation. The data movement through the design hierarchy of this project is as follows.

- The control unit acts as the external memory and is used to initially store and feed the input data required for computation. The iacts and weights are transferred onto their respective GLBs within all the clusters.
- After data is stored in respective GLBs, the control unit triggers the data transfer from them to the PE scratchpads. This transfer happens via the routers present between them.
- Once the data is stored in the PE cluster, each PE reads weights and iacts from its local scratchpad and operates on them to produce partial sums.
- The partial sums are then stored back into the GLB cluster via the routers and are finally read by the controller for verification.

The original architecture states that the routers are configured before dealing with each layer computation. A similar approach has been undertaken in this project, where the control unit acts as the brain for controlling all storage, data transfers, and computations that occur as part of overall processing. The following section describes in detail the implementation of this architecture.

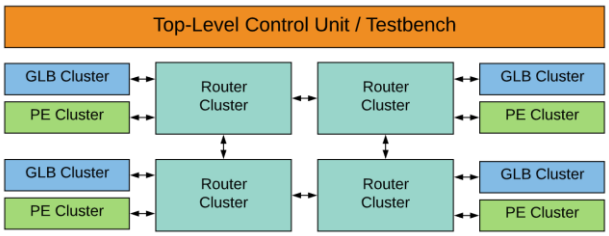

Fig. 3. Eyeriss v2 - Scaled-down Architecture

III. Implementation

This section goes over the device-level details of all the components used for construction and assembling the architecture. The components include the processing element, router, global buffer, and their respective clusters.

A. PE Architecture

The processing elements in a systolic array form the basis of computation. The PE implemented in our design is very similar to the specifications used in Eyeriss as well as other papers that deal with systolic arrays. Fig. 4 shows the design of a single PE in our project. Each PE contains a MAC unit that accepts three inputs, a weight, an input activation, and a partial sum. The weights and activations are read from the local scratchpad(SPad) of size 1 kilobyte. The SPad is divided into two for storing weights and iacts separately. Since the SPad size is parameterized, it supports changing the size depending on how much data it a PE might need to hold Also, a simple control logic is built, within the PE, as a state machine to receive control signals from the top and automatically read and compute partial sums. This eases the control of an array of PEs from the top as they undergo continuous computations.

The PE cluster is essentially an array of processing elements and is important for mapping parallel computations. The PE cluster of our design is also parameterized, and the number of PEs can be decided based on the sizes of inputs and parallelism we wish to exploit from them. In our implementation, we have verified a 3x3 and a 5x5 PE array for performing convolutions with 3x3 and 5x5 kernels respectively. A more sophisticated and flexible PE cluster that can handle convolutions with a wide range of input and kernel sizes would require extensive amounts of verification and a complex mapping logic, and thus has not been implemented as part of this project.

The Row-Stationary(RS) dataflow of the Eyeriss architecture has been incorporated in each PE cluster. The original Eyeriss paper considers this dataflow to be a combination of the Input Stationary, Output Stationary, and No-Local Reuse dataflows and states that the RS dataflow has a position to optimally reuse all datatypes. Fig. 5 shows how the data movement happens during computation. The concept is to breakdown the inputs into 1D convolution primitives and perform row-wise computations. Each primitive is mapped to one PE for processing and the computation of each row in a convolution is

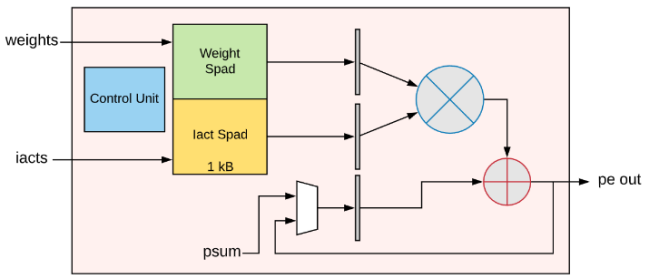

Fig. 4. PE Architecture

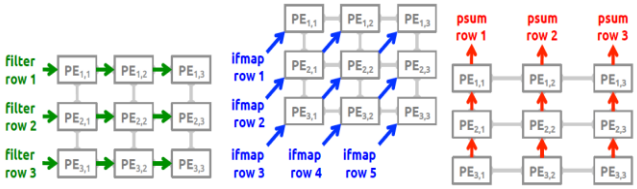

Fig. 5. Row-Stationary Dataflow

handled by its corresponding PE. The partial sums, after computation, are accumulated vertically and this essentially represents rows of the final convolution output.

B. Router Architecture

The routers are an essential part of any network-on-chip (NoC) architecture. NoCs implemented in commercial chips, especially System-on-Chips (SoCs) are mainly used for reliable communication between the various subsystems that form the entire design. While this is highly secure and flexible with regards to on-chip communication, it is built with extremely complex logic to deal with contention, bandwidth, and packet prioritization. Incorporating such a circuit inside an ASIC aimed at fast computations might lead to immense latency overheads. These are characteristics of a packet-switching network and this kind is widely used in network chips and complex SoCs.

The router design proposed in Eyeriss v2 employs circuit-switched routing, which is used for establishing a path for a stream of data to travel and reach its destination. This implementation is much simpler than a packet-switching router as it does not deal with data flooding and buffering of incoming information. The routers are statically configured by the controller according to the layer being computed. Fig. 6 shows the structure of a single router designed for this project.

Since the grouped clusters are arranged in a 2x2 array as shown in Fig. 2, the router clusters handle 4 communication directions, 3 to communicate with other routers and 1 to communicate with a PE and a GLB cluster. To model such an organization, each router is designed with 4 interfaces, each representing a direction of data transfer. Within each router is a combinational logic that mimics a circuit-switching crossbar to establish output data path depending on the input direction and the control signal from the control unit. The combinational logic is mainly a composition of multiplexers and select lines that choose the output path based on the configured direction.

The router cluster in our project contains 3 routers to deal with iacts, weights, and psums. The router clusters on the west side of the array have the PE and GLB cluster interfaced on their west direction and similarly, the router clusters on the east side of the array are interfaced with other clusters on their east direction. The rest of the directions in both cases are interfaced with their neighboring routers. A more detailed description of the dataflow using the router will be covered in the next section.

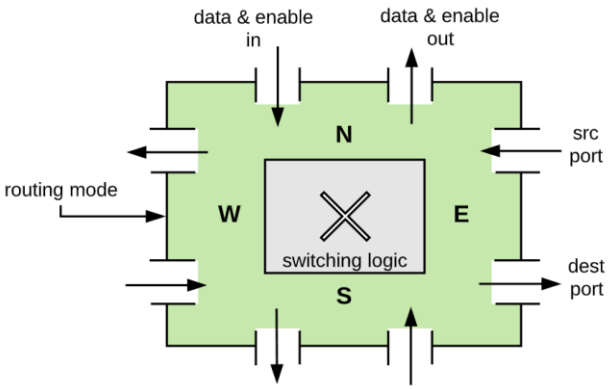

Fig. 6. Router Architecture

C. GLB Architecture

The global buffers are very similar to the local scratchpads in design but are larger in size(2kB) to allow storage of more weights and iacts for data staging and routing. A GLB cluster consists of three global buffers, each used for exclusively storing a single datatype. The GLB cluster acts as the point of entry for any data that enters the design. The control unit, that also acts as off-chip DRAM, directly controls reading and writing of the global buffers when bringing data on-chip. The router cluster can also communicate with the buffers when data must be transferred onto the PEs.

IV. Methodology and Results

This section covers all the steps taken in verifying the RTL models and the experiments done using the implementation defined in the previous section. It also provides the simulation and synthesis results obtained. The following section is subdivided into three phases of execution that were tried as part of this project. The three phases differ in the kind of implementation and experiments done to verify the architecture. They also roughly represent the design stage that the project was in during that phase.

A. Phase I

The main objective during this design phase was to get the PE cluster working and perform convolution operations. The architecture at this phase dealt with a single grouped cluster (a PE cluster, a router cluster, and a GLB cluster) for easier verification of the convolution output. The router used in this design was a placeholder for the original router and did not have interconnection capabilities. It possesses a control logic to read data from the GLB cluster and pass it on to the PE cluster and finally get the partial sums/outputs back into the GLB cluster. Fig. 7 shows the data movement this implementation.

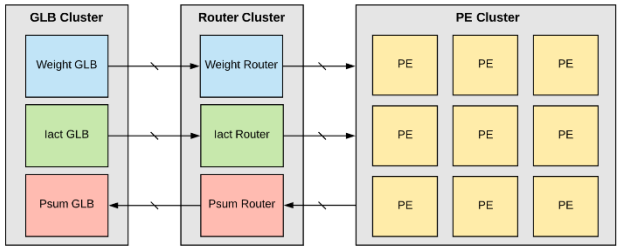

Fig. 7. Data Movement – Phase I

Although this was a naïve approach of routing the data and did not provide enough control at the top to choose the direction of routing, it made it easier to confine the verification scope just on the PE cluster. The data movement, as described in section II, included staging the data into the local scratchpads and performing RS dataflow-based computations to produce final convolution output.

Fig. 8 shows the simulation output obtained from Xilinx Vivado while verifying this model. It includes the convolution output of a 3x3 kernel over a 5x5 iact channel. The simulation includes every data movement step from transferring input data (iacts and weights) into the iact and weight GLBs, until the storage of partial sums for that convolution into the psum GLB. A rough estimation of the latency has been calculated based on the frequency and cycles obtained from synthesis and simulation with Vivado. The entire simulation takes 120 cycles including minimal delay overhead from the testbench used to control the design. Synthesis results from Vivado show that a frequency of 250MHz can be achieved for this design without any timing violations. Combining these results, we get a latency of 480ns for performing a single convolution operation starting from data staging.

B. Phase II

Once the PE cluster has been verified for computation, the next step was to make a more scalable router design with direction control. The objective of this phase was to model the router depicted in Fig. 6 and interface it with the PE and the GLB clusters. As explained in the previous section, this router has the capability to interface with other routers and clusters and hence gives a sense of scalability to the architecture. Once a single grouped cluster was formed with this router, the clusters were replicated to form a mesh network, like the one illustrated in Fig. 3. Since a single grouped cluster can handle a single convolution operation at a time, four such clusters can perform four separate convolution operations, each representing the computation of partial sums resulting from different channels of the input activations.

Synthesis of this design gives a frequency of 285.71MHz and simulation, which was performed with similar conditions as phase I, resulted in 112 cycles. Combining this data, we can roughly estimate the latency of a single convolution operation, done by each PE cluster, will take a total of 392ns.

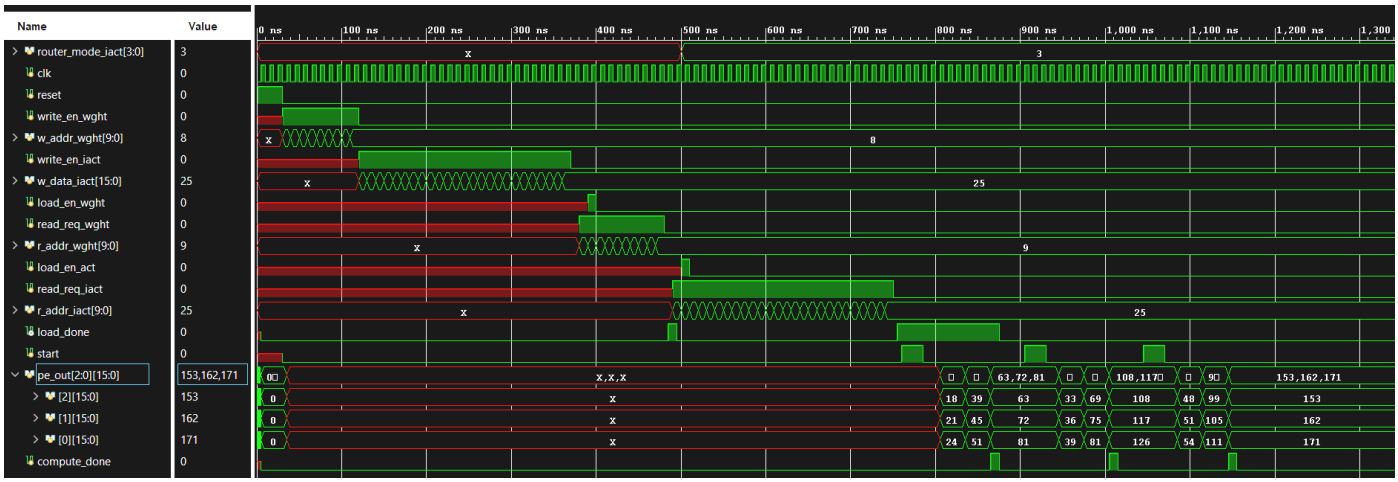

Fig. 8. Simulation Output – Convolution with a single grouped cluster

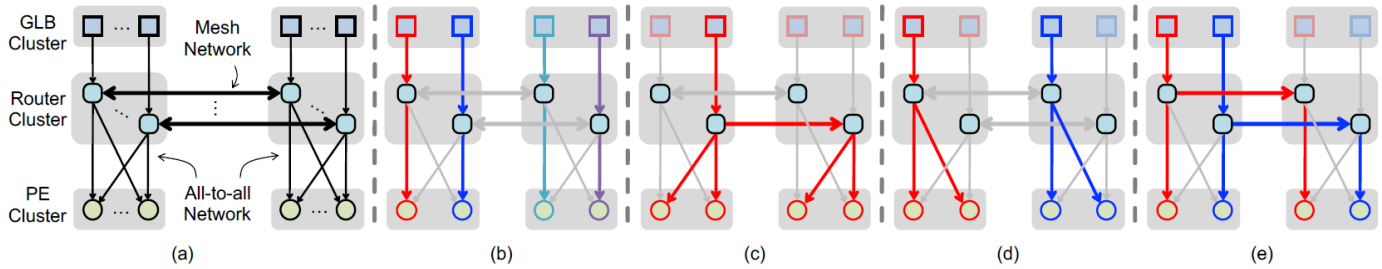

Fig. 9. Router configurations – a) all-to-all network, b) High-Bandwidth mode, c) High-Reuse mode, d) & e) Multicast mode

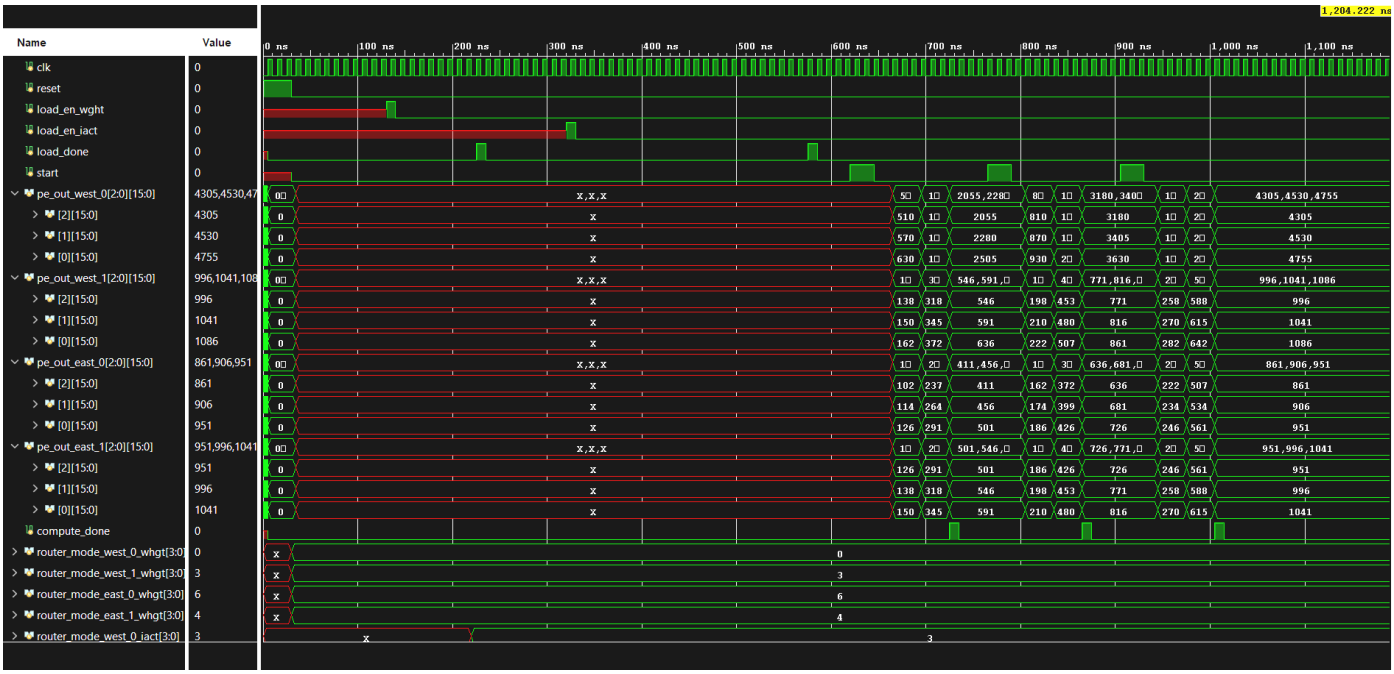

Fig. 10. Simulation Output – Convolution with 4 PE clusters and router network configuration based on phase III

The difference in latency can be attributed to the different router design compared to phase I. The router in phase II is essentially a switch that establishes data path according to the given configuration and does not have a sophisticated control of its own. This helps in achieving a higher frequency for the overall design. Also, this makes the GLB cluster indirectly connected to the PE cluster with a combinational path between them and thus consumes lesser cycles to perform a convolution operation.

C. Phase III

The objective of this phase is to explore the capabilities of the mesh NoC architecture. From phase II, routers can now be controlled to communicate with other router and clusters. This section is based on the router organization of the Eyeriss v2 paper and uses multiple routers in each router cluster to explore the data reuse opportunities present.

The architecture defined in the paper enforces an all-to-all network, which means every router in a router cluster can communicate with every PE of its corresponding PE cluster. A similar implementation has been done with our router design. The top-level design for this exploration consists of two router clusters, having two routers each. This experiment limits only to two clusters due to the presence of a potentially large number of interfaces which would complicate verification.

The possible router configurations which enable different kinds of data reuse opportunities have been listed below and shown in Fig. 9:

- High Bandwidth mode – each router works independently of other routers(unicast). Each router can bring in distinct data from the global buffers and pass them on each PE of the PE cluster. This configuration can be used when one kind of data is extremely reused, while the other kind needs to be spread out with distinct values. A possible application is one end of the fully connected layers.
- High Reuse mode – a single router acts as a master source of data and passes them on to all neighboring routers and eventually route them to all PEs within a cluster. This mode is commonly used to broadcast data to every other cluster and an application of this is the depth-wise convolution layer, which needs one kind of data to be broadcasted while the other kind is unicasted.
- Multicast – This is a tradeoff between unicast and broadcast, where a router communicates with neighboring routers to unicast data to both PE clusters or independently broadcast to corresponding PE clusters. This mode is works well for convolution layers.

These use cases were simulated and verified with two router clusters using the routers from phase II to showcase their capabilities. The simulations included only the router clusters, which were controlled for different configurations from the top. Finally, a combination of the concepts established in phase II and phase III was simulated with an integrated design consisting of 4 PE clusters connected by a weight router network and an iact router network. This simulation is shown in Fig. 10, where the waveforms of the 4 PE clusters (west_0, west_1, east_0, east_1) show the 4 convolution operations done in parallel. Initially, the routers are configured to broadcast weights across all PE clusters from a single global buffer. After transferring weights, the routers are reconfigured to unicast the iacts into each PE cluster their closest router and global buffer. The PE clusters then simultaneously compute 4 convolution operations, corresponding to 4 input channels, with a single kernel. The vice versa is also possible, where multiple output channels are computed with different kernels but the same input channel, when the activations are broadcasted, and weights are unicasted. This simulation depicts an instance where on-chip network can be useful in delivering the right data to several destinations efficiently.

V. Conclusion and Future Work

This project was an effort to design and analyze a systolic array architecture for deep learning. The choice of Eyeriss v1 & v2 accelerators as base references was mainly due to their concept of dataflow at a PE-level and dataflow at a Hierarchical Mesh-level. The usage of a network-on-chip like architecture to facilitate efficient data transfer could give rise to a highly flexible accelerator that exploits data reuse and eliminates staging overhead. A SystemVerilog, RTL model of such an architecture was implemented in this project using Xilinx Vivado. The simulation and synthesis of the models resulted in rough frequency and latency numbers for the architecture.

Since this project is also an effort for exploration of the concepts proposed in the Eyeriss architecture, there is a lot of scope for future work. The first step moving forward should be to fully integrate and verify all the concepts described in each phase of the implementation. This would give a complete working system that captures all the concepts of the entire architecture. Secondly, a more sophisticated and customizable processing element cluster could be designed to support a variety of convolution operations with different sizes of kernels and input activations. Also, more logic can be added to routers to smartly route data without extensive control from the top and use a 2-way or 4-way handshaking mechanism for proper communication between hardware components.

Acknowledgment

We thank Prof. Hadi Esmaeilzadeh of University of California San Diego for motivating and guiding us throughout this project. We also thank Fatemehsadat Mireshghalla, the instructional assistant for the course, Accelerator Design for Deep Learning, and Soroush Ghodrati for helping us during the initial stages of this project. Finally, we thank UCSD for giving us the opportunity to work on this project as a part of this course and our classmates for providing insights and conducting discussions on an assortment of contemporary literature in this research space.

REFERENCES

[1] Chen, Yu-Hsin, Joel Emer, and Vivienne Sze. "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks." *ACM SIGARCH Computer Architecture News*. Vol. 44. No. 3. IEEE Press, 2016.

[2] Chen, Yu-Hsin, et al. "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices." *arXiv preprint arXiv:1807.07928* (2018).