Neuromorphic Array Communications Controller to Support Large-Scale Neural Networks

Aaron R. Young, Mark E. Dean, James S. Plank, Garrett S. Rose
Department of Electrical Engineering and Computer Science
University of Tennessee
Knoxville, Tennessee, USA, 37996
Email: ayoung48, markdean, jplank, garose@vols.utk.edu

Catherine D. Schuman
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831-6085
Email: schumancd@ornl.gov

***Abstract*—Neuromorphic computing is one promising post-Moore's law era technology. In order to develop and use neuromorphic systems, traditional von Neumann-based computers must be able to communicate with neuromorphic hardware to support functionality such as monitoring the state of the network, optimizing the array to better perform the task, and input/output data processing. In this paper, we describe our use of a separate neuromorphic array communications controller to support high-throughput, low-latency communication between a traditional computer and our implementations of neuromorphic systems. The goal of the communications controller is to provide enough performance to facilitate the desired interaction between the systems and to enable scaling of the neuromorphic systems to larger sizes.**

I. Introduction

As the limits of conventional computation are reached, new architectures that break away from the traditional von Neumann architecture will need to be researched, developed, and deployed. One promising class of post-von Neumann architectures are the brain-inspired, neuromorphic architectures. Spiking neural networks, one type of neuromorphic architecture, are event-based networks with an inherent notion of time [1]. The neurons in this network create a fire event when their charge exceeds a threshold. This fire event results in a spike with a weight value. The connected neurons receive the spike and increase their charge by the weight. The input, output, and internal communication are all done via spikes.

Many of these spiking neuromorphic architectures are implemented in a separate physical device, either as a custom designed VLSI chip [2] or within an FPGA [3]. In order to develop these neuromorphic systems, traditional von Neumann-based computers must be able to communicate with the new neuromorphic arrays over a fast and flexible communication channel. This communication setup should be fast enough to allow the neuromorphic processor to operate in real-time as a coprocessor to the von Neumann computer, thereby allowing real-time data processing and control applications to run on the system. The communication setup also needs to allow for larger array sizes and use of multiple chips, enabling the setup of larger neuromorphic arrays. Creation of a communication setup that meets the growing needs of larger neuromorphic hardware is a challenge.

We propose to solve the challenge of interfacing traditional computers with neuromorphic systems through the use of a separate neuromorphic array communications controller implemented on a separate FPGA board. This separate communications board acts as a fast intermediary between the computer and the neuromorphic array, providing both a high-speed, low-latency channel between the devices and the capability to scale the neuromorphic chips to larger sizes, as well as to multiple neuromorphic chips.

This paper discusses the considerations and design of our neuromorphic array communications controller, which has been designed for high performance, scalability, and flexibility. We go into fine detail on the various decisions that we made in designing and implementing the controller, and we measure the impact of these decisions experimentally. The decisions and accompanying details are intended to be useful to those who design neuromorphic hardware systems. In particular, most hardware design focuses on the neuromorphic elements themselves. In this paper, we wish to demonstrate that the communications between the hardware, and the host who uses the hardware, is an important component that needs careful consideration and can impact performance of the neuromorphic system.

II. Related Work

Many research groups are working on neuromorphic hardware. Although the neuromorphic components are different, each group must develop ways to connect their neuromorphic elements together and also to off-chip devices. The human brain has billions of neurons, each with thousands of connections,

Notice: This material is based on research sponsored by the Air Force Research Laboratory under agreement number FA8750-16-1-0065. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government. This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan). Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U.S. Department of Energy.

so as researchers work to scale up their designs, they will be faced with additional communication challenges.

Researchers at Stanford University have created a mixed-analog-digital system called Neurogrid [4]. Neurogrid uses deadlock-free multicast tree packet routers to transmit data between its 16 Neurocores [5]. Each Neurocore has its own full-custom asynchronous VLSI implementation of this router. Neurogrid communicates off-board to a computer running the software stack via USB 2.0 with the use of a Cypress EZ-USB FX2LP (FX2) [6].

Manchester University, in the United Kingdom, is working on a large neuromorphic system called SpiNNaker [7]. SpiNNaker simulates neurons and synapses with digital chip multiprocessors (CMPs). Each processor has its own communications controller which communicates with an on-chip router to send neural spike signals. The packet-switched router forms links between each on-chip processing core and to the routers of the neighboring chips. A conventional computer is connected by Ethernet to one or more SpiNNaker CMPs [8]. This computer is used to specify the neuromorphic model, trigger the simulations, and retrieve the results.

The Human Brain Project has developed waferscale neuromorphic hardware through the FACETS and BrainScaleS projects [9]. A special communication infrastructure had to be developed to support communication needs of the waferscale neuromorphic system [10], [11]. Intra-wafer communication is handled by a packet-based network which connects the wafer to surrounding wafers and host PCs. The main component of the packet-based network is an application-specific integrated circuit called a digital network chip (DNC). The DNC employs synchronous high-speed serial packet communication to transmit time-stamped spike events. The packet network is set up hierarchically with high input count analog neural networks connected to DNCs, which are connected to a custom FPGA board, which then connects to the host PC.

IBM has also developed a neuromorphic platform called TrueNorth as part of the DARPA SyNAPSE project [12]–[14]. TrueNorth is composed of a scalable network of neurosynaptic cores. These neurosynaptic cores are connected together through an on-chip communications network to form large neuromorphic arrays. Short range connections are implemented with an intra-core crossbar memory and long range connections are implemented through an inter-core spike-based message-passing network.

The current methods of handling host-to-neuromorphic array communications are not without their problems. Slow connections like USB 2.0 and Ethernet have limited bandwidth, which in turn limits the number of simultaneous commands and events which can be transferred. Additionally, the current designs are not well suited for processing information in real-time and for guaranteeing accurate arrival of time-sensitive data. New designs need to have sufficient communication capacity to achieve in situ processing of information. New features are also desired, such as real-time monitoring, online optimization, and live reconfiguration of the array. The TENNLab research group is working to overcome these challenges and research new features through their unified application framework and development environment, which allows cutting-edge research on multiple neuromorphic models simultaneously [15]. The neuromorphic array communications controller discussed in this paper allows the host to connect to hardware implementations of the neuromorphic models.

III. High-level communication considerations

Communication between a traditional computer and a neuromorphic array is needed to perform a multitude of operations. Often the neuromorphic hardware acts as a co-processor and is configured and controlled by a computer which will hereinafter be referred to as the host system. The types of information which needs to be sent between the host and the neuromorphic array include the array configuration, input/output firing commands, and array monitoring data. These host machine functions drive the communications requirements and deserve further discussion.

A. Monitoring

A crucial function for both developing and debugging neuromorphic hardware is the capability to monitor the operation of the neuromorphic array. Live monitoring of the network can be leveraged to provide valuable feedback used to understand the properties of the network that allow the network to learn to run the application. The live monitoring also allows analysis of network activity, leading to comparisons between different neuromorphic models and the analysis of power utilization between different networks and applications.

Monitoring can also be used to detect unexpected behavior of the network, which could either be caused by a bug in the logic of the system or an attack on the system. For high security jobs, monitoring can be used to detect a security or safety vulnerability in the network.

Real-time monitoring will cause an additional strain on communication as the diagnostic data has to be transmitted in addition to the normal information. The level of monitoring will also have a direct impact on the communication load. The information collected per element in the network will have an exponential impact on the communication needed.

B. Optimization

The host can also be used to drive real-time learning and optimization of the neuromorphic network. The host can deploy multiple mutated networks along with the current best network. If one of the new networks starts to perform better than the existing best network, the host can switch to using the new best network. This allows for a robust form of online learning to take place while the application is running, enabling the network to adapt or be optimized to solve the application while the network is active. Using the host to perform online optimization would increase the communication demand; additional bandwidth is needed to load and switch between multiple networks.

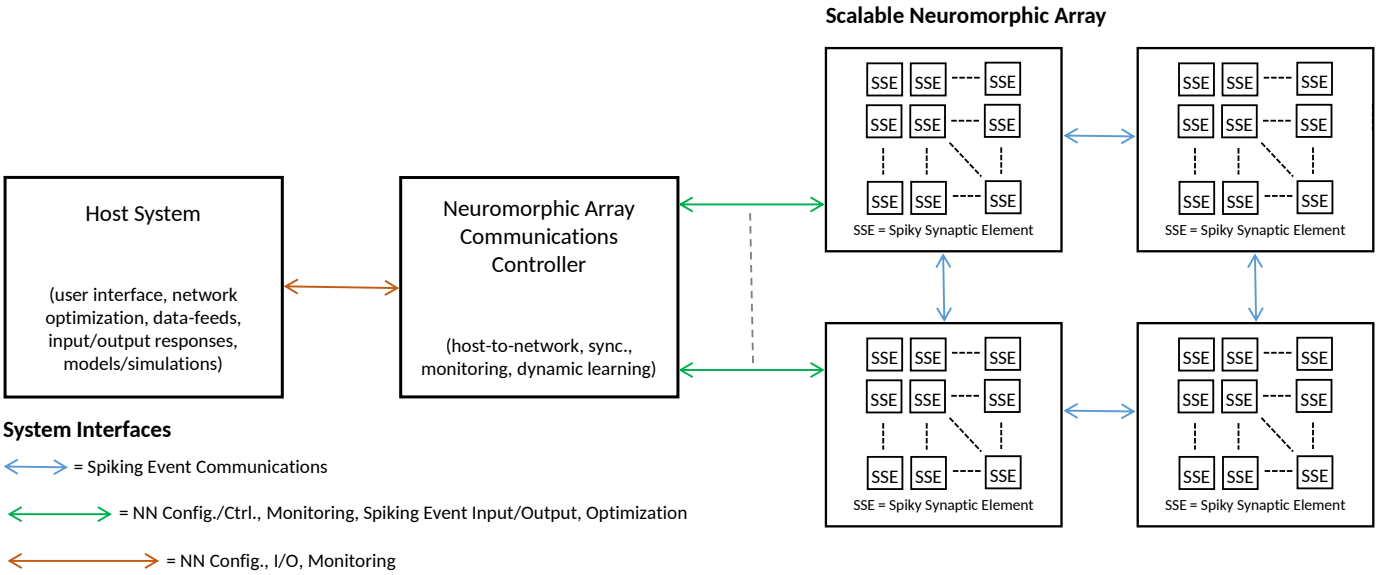

Fig. 1. Scalable Neuromorphic System Infrastructure

C. Host to array communications

The host will often need to send operational commands to the network. These commands include network loading and initial starting and stopping of the operation of the network. The host will also often need to send data between the neuromorphic network for processing. The most promising use case is sending real-time streaming data to the neuromorphic processor for real-time processing. Streaming real-time data will drive throughput and latency requirements needed to ensure that information can pass seamlessly between the two processors.

D. Scale to external interfaces

The host machine is also needed to support scaling to a multitude of external interfaces. Many neuromorphic applications will require processing of either the input or the output of the neuromorphic spikes. The host machine can provide this functionality and allow the neuromorphic processor to scale to any external interface.

IV. Use of an Intermediate Communication Board

While weighing the possible methods of interfacing the host machine and neuromorphic array, the method which made the most sense is to develop a custom neuromorphic communications board to facilitate the communication. Using a separate communications board has many advantages over a direct connection. For starters, a separate board provides a great increase in flexibility. With a communications board, separate methods of communication can be used between the host and the communications board and between the communications board and the neuromorphic array. This allows for a wider range of hosts and communications boards to be supported. In addition, the interface on one side of the communication can be changed without having an impact on the other communication interface.

A separate communications board also relieves the burden of supporting a complex protocol from the board implementing the neuromorphic array. Often a complex protocol, requiring a substantial amount of resources, is needed to communicate with a host machine. With a separate communication board, the burden of implementing the complex protocol is handled by the communications board, and a simpler protocol, with less resource requirements, can be used to connect to the neuromorphic array. This will free up resources on the neuromorphic chip to be used for the implementation of the neuromorphic hardware.

A dedicated communication board also enables increased scaling of the neuromorphic array. A single connection from the host to the communication board is needed; however, multiple neuromorphic processing boards can be connected to the same communications board. This allows the neuromorphic arrays to scale to multiple boards without modifying the host connection. Figure 1 shows a diagram of the connection of the host machine to the neuromorphic array with a communication board being used as an intermediary between the two systems. Use of a separate communications board will allow host-to-array communication to have additional flexibility and greater scaling potential.

V. Exploration and Selection of Communication Protocols

In order to support the key features listed in Section III, the communication protocol selection is crucial. The selected protocols need to have sufficient bandwidth and low latency to support the real-time, high-bandwidth transfers that will be needed.

A. Host to Communication Board

The connection between the host and the communication board is limited to protocols that are available both to an

FPGA and to a PC. The main options are USB, PCIe, Ethernet, fiber optic, Serial ATA (SATA), and UART. UART is quickly eliminated by being the slowest by far with a maximum speed of 115200 b/s. We have previously used USB with the help of a Cypress USB 3.0 peripheral controller. It proved to be difficult to use and offered insufficient performance [16], [17]. Gigabit Ethernet (GbE) over twisted pair cables has a maximum speed of 1000 Mb/s or 125 MB/s and is widely available on most PCs. However, at this speed it is slower than USB 3.0.

Fiber optics can be used to run 40Gb Ethernet (40GbE) with a maximum speed of 5 GB/s. A major disadvantage is special computer hardware is required since commercial PCs do not come with fiber optic ports.

SATA has a maximum speed of 16 Gb/s or 2 GB/s, which makes it a compelling option; however, SATA connectors are not commonly found on FPGA boards without an adapter and interface protocol intellectual property (IP) would have to be licensed or custom-designed in order to use it.

Peripheral Component Interconnect Express (PCIe) is a high speed serial computer bus standard, which is used by virtually all modern computers. PCIe is well-suited to connect peripheral devices and is used to connect other co-processing boards such as GPUs. PCIe is a complex protocol with many low-level details that must be implemented correctly to create a successful design. Luckily, there are existing solutions that make getting started with Host to FPGA communication over PCIe easy. Xillybus is one such solution which proved sufficient for our initial communication board design [18].

PCIe is the fastest option followed by 40GbE. If Xillybus is used to aid in PCIe communication, the maximum bandwidth is 800 MB/s, 1700 MB/s, and 3500 MB/s for Xillybus revisions A, B, and XL respectively [19]. Should Xillybus ever prove insufficient, a custom PCIe driver and FPGA IP core can be designed. Since PCIe is a common PC interface available on all desktop computers, fast and easy to implement with the help of Xillybus, it was chosen as the interface between the host PC and the communication board.

B. Communications Board to Neuromorphic Array

Even more options are available when choosing an interface between the communication board and the neuromorphic array. In order to have a complete communication setup, decisions have to be made about the physical connection, the encoding of the data, the link level protocol, and the transport level protocol. To get the best performance and use fewer pins, a high-speed, asynchronous bus is used. Gigabit transceivers are used to transmit the data across the bus.

Xilinx provides a LogiCORE IP called Aurora, which is an open link-layer protocol that uses the high-speed serial transceivers on Xilinx FPGAs. The Aurora core is lightweight, scalable, and provides many configuration options to the user. The Aurora core can take full advantage of the high-speed transceivers and can use up to 16 transceivers for a channel, which results in a throughput that ranges from 480 Mb/s to over 84.48 Gb/s. Aurora was selected as the link-layer protocol because of its availability, flexibility, cost, and speed. For VLSI implementations of neuromorphic arrays, an Aurora compatible network interface can be designed using the open Aurora protocol specification.

Aurora is able to use either 8B/10B or 64B/66B line encoding. The 8B/10B encoding is widely used with many serial technologies, such as Ethernet and PCIe. The 64B/66B encoding is used for 10 Gigabit Ethernet and has less encoding overhead than 8B/10B [20]. A disadvantage to 64B/66B is a lower ratio of sync bits to payload bits, which can result in the possibility of a slight DC bias, longer alignment times, and more complex encoders and decoders. Because of the downsides of 64B/66B encoding, 8B/10B encoding was chosen for the first implementation. If the overhead of 8B/10B proves to be too great, the encoding can be changed to 64B/66B at a later time.

Since Aurora is designed as a link-level protocol, it does not have any provision for guaranteed delivery. Aurora will try to maintain an open channel and deliver packets on a best effort basis. Aurora does have the capability to perform CRC checking on frames of data that are transmitted, but Aurora does not have any built-in recovery mechanism for incorrectly transmitted data. There are existing commercial transport level protocols available which include delivery guarantees; however they do have their downsides. Besides the licensing agreements, these solutions included many more features than are needed for a lightweight, high-speed, and low error chip-to-chip protocol. The extra overhead for the unused features of the more complex protocols would result in reduced performance without an added benefit. To overcome this limitation, a simple lightweight Go-Back-N retransmission protocol was developed and added on top of Aurora to ensure that packets sent over the Aurora channel are received correctly. This addition of the lightweight retransmission protocol proved to have minimal overhead due to the hardware implementation and the lightweight nature of the protocol.

Deciding to use Aurora as the link-layer protocol is only part of a board-to-board communication solution; a physical connector still needs to be chosen. Most Xilinx FPGA boards route the high-speed transceivers to either a special purpose connector, like PCIe or SFP, or to a general purpose connector, like an FMC connector. Since the FMC connector is commonly found on most FPGAs, and since it has the highest number of high-speed signals, it was the logical choice. Some FPGAs are designed to stack and can be directly connected together. Other FPGAs need an intermediary. FMC is designed primarily to connect FPGAs to a daughter card and is not designed to be able to connect two FPGAs together unless one of the FPGAs is designed to stack. Because of this, the 8-port FMC to SMA daughter card and SMA cables were chosen to connect FPGA boards that are not designed to stack. Using SMA cables as a general connection is logical since they can handle the high-speed differential signals and provide maximum flexibility with each transceiver being wired up independently. SMA connectors are also easily added to custom boards with VLSI chips. The main downside is the large number of cables that will have to be connected—2 cables per lane per direction, resulting in 4

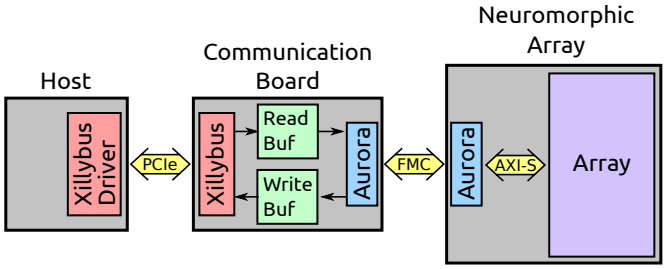

Fig. 2. Diagram of the Communications Board

cables needed to connect one duplex Aurora lane. However, this solution is still the best available, resulting in the fastest speeds and the most flexibility.

VI. New Communication Board Design

After evaluating various possible communication solutions, a new communications board was designed using the best options. Figure 2 shows a high-level block diagram of the new communication setup. The communication board sits in the middle and facilitates communication between the host PC and the neuromorphic array. The communication board connects to the host over PCIe using Xillybus. The Xillybus driver has to be installed on the host to interface with the communications board. The Aurora protocol is used to transfer data from the communication board and the neuromorphic array. A Go-Back-N automatic repeat request (ARQ) transport layer guarantees packet delivery. The transceivers used by Aurora are connected via an FMC connector. Data sent to the communication board is stored in buffers until the destination is ready to receive it. The buffers are asynchronous FIFOs and provide synchronization between the clock regions used by Xillybus and Aurora. On the neuromorphic array chip, an AXI4-Stream bus is used to connect Aurora to the neuromorphic array.

The complete communication system has been implemented and the performance of the system has been evaluated. The next section looks at the performance benchmarks of the system.

VII. Results

In order to verify the performance of the communication board versus the previously used Cypress FX3 USB based communication, multiple benchmarking tests were conducted. Each test setup is designed to measure either the complete communications path or an individual component of the communications path. Each test implemented a communications loop back. The host sent messages and measured how long the same messages took to be received. The packet size for the messages was chosen to be 64 bytes long. The test setups used to measure performance are as follows.

FX3: Measures the performance of the prior Cypress FX3 communication setup. This is the only test setup not using PCIe.

PCIe: Measures the performance of Xillybus PCIe component of the communication board design.

PCIe 64: Measures the increase in performance of the Xillybus PCIe component when 64-bit buses are used internally instead of 32-bit buses. The larger 64-bit bus forces the smallest transfer size to 64-bits but provides additional performance.

PCIe with FX3 Emulator (PCIe GPIF): Designed to measure the performance of the General Programming Interface (GPIF) used with the FX3 implementation.

Aurora x1: Aurora with one lane of communication.

Aurora x2: Aurora with two lanes of communication. The number of lanes equal the number of high speed transceivers used for the connection.

Aurora x1 Ack: One lane Aurora with a Stop-and-Wait ARQ. One packet is sent and acknowledge before the next packet is sent. This shows the need for a more complex acknowledgement protocol to guarantee packet delivery.

Aurora x1 Window: One lane Aurora with a Go-Back-N ARQ. Go-Back-N has sufficient performance when the error rate for the data path is low.

All the tests were conducted from a host system consisting of an Asus P10S-M micro ATX motherboard, an Intel Xeon e3-1275 processor, and 32 GB of DDR4-3333 memory. The computer is running Ubuntu 16.04.2 LTS. Because of a buffer flushing problem with the Xillybus driver packaged with Ubuntu 16.04, the newest Xillybus driver needs to be downloaded and installed. Xillybus Revision B was used as it performs better than Revision A and is a drop-in replacement. The FX3 test setup used a Cypress FX3 board (FX3) and a HiTech Global HTG-777 with a Xilinx Virtex7 X690T (690T). The remaining tests all used a Virtex7 Xilinx VC707 evaluation board (VC707) for the communication board. The PCIe GPIF test setup and the Aurora test setups all communicated with a 690T that acted as the neuromorphic array board. The Aurora communication logic uses less than 1% of the FPGA's resources, allowing the vast majority of the FPGA's resources to be used for the neuromorphic array.

The two main metrics measured are round trip latency and round trip throughput. There are two main variables when performing the benchmarks. The first is the size of the buffer used when making a call to the transfer and receive functions. A larger buffer means that more data can be transferred before the user program has to be involved. This variable is called the transfer size.

The other main variable is the total amount of data that is transferred. This total amount is transferred one transfer size at a time until the total amount is reached. The user program needs to be reentered to make the next transfer call when the total size is bigger than the transfer size. Thus, the user program will have to be entered $\frac{\text{total transfer size}}{\text{transfer size}}$ times. The benchmark makes the assumption that the total transfer size is a multiple of the transfer size.

A. Latency Benchmarks

The first set of benchmarks are aimed at measuring the latency of one round trip transfer of a 64-byte packet. This means that the total transfer size and the transfer size were

both kept to 64 bytes. In order to obtain clean measurements, the computer was taken off the network and run without a graphical user interface. In addition, the benchmark program would send 1000 round trip packets before sending a packet that is measured. The program would then average 1000 of the measured packets together to get the mean and standard deviation for the data point. The latency benchmark was run for all the test setups and the results of the benchmark can be found in Figure 3. The FX3 test setup had by far the highest round trip latency, with a latency value of $80.38\,\mu s$. All the other test setups have a round trip latency of around $6\,\mu s$. Aurora x1 Window has higher round trip latency than the other PCIe based implementations, which is caused by the additional buffers used to store the send window. The measurements obtained had a low variance, with the standard deviation from the FX3 benchmark being $2\,\mu s$ and from the PCIe-based benchmarks being $0.2\,\mu s$. The low variance in part indicates that there were no measurement artifacts in the data collected. Additionally, the measured values are as expected. The documented FX3 firmware processing time for each DMA buffer is about $40\,\mu s$. Since a round trip transfer has to be processed twice by the DMA engine, a total round trip time of $80\,\mu s$ seems very reasonable [21]. The much lower values of the PCIe benchmarks is also logical. Since Xillybus allows for explicit flushing of the DMA buffers, the latency of the round trip packet is much lower.

Looking more closely at the various PCIe-based benchmarks, they all appear as expected. Taking into account the standard deviation values, the measurements for PCIe are all roughly the same. However, the mean is higher for PCIe with the FX3 emulator and the four Aurora tests. One would expect the mean for these to be higher since they communicate to the 690T and back whereas the PCIe test does not. The theoretical latency for both the GPIF and Aurora can be calculated. The GPIF latency is calculated as shown in (1).

$$\text{GPIF latency} = \text{clock freq.} \times (\text{data cycles} + \text{overhead}) \quad (1)$$

Assuming the overhead is around 10 cycles, then the round trip transfer time is $0.4\,\mu s$, as calculated in (2).

$$\text{round trip time} = 100\,\text{MHz} \times ((16 \times 2) + 10) = 0.4\,\mu s \quad (2)$$

The theoretical increase of $0.4\,\mu s$ is larger than the observed increase of $0.03\,\mu s$, but taking into account that the transfer can start while the PCIe transfer is still in progress, the observed increase seems reasonable.

A theoretical calculation for Aurora can similarly be made. The Aurora latency can be calculated as shown in (3), which results in a theoretical latency of $0.1638\,\mu s$.

$$\frac{\text{bits to transfer}}{\text{transfer rate}} = \frac{64 \times 16}{6.25\,\text{Gbps}} = 0.1638\,\mu s \quad (3)$$

The theoretical increase of $0.16\,\mu s$ is about the same as the observed increase of $\approx 0.1\,\mu s$. Again, the Aurora transfer can start while the PCIe transfer is still taking place, which explains why the observed value is less than the theoretical value.

B. Throughput Benchmarks

The second set of benchmarks are aimed at measuring the maximum throughput of each design. Figure 4 shows the throughput measured for each test design when the total transfer size is held constant and the transfer size is varied. The FX3 setup has the lowest throughput. It starts off at about $1\,\text{MB/s}$ and increases linearly to $108\,\text{MB/s}$. This increase in throughput is a result of making better use of the USB 3.0's bursting capabilities. In order to maximize the FX3's performance, a large burst length and buffer size is required. The upper bound of the FX3's performance is caused by the implementation of the GPIF interface [21]. The GPIF's maximum throughput is shown by the PCIe with FX3 emulator line. According to "Optimizing USB 3.0 Throughput with EZ-USB" [21], the maximum throughput of the FX3 is $450\,\text{MB/s}$. This means the FX3's performance is limited by the implementation of the GPIF interface logic. The maximum theoretical throughput of the GPIF interface is $32\,\text{bits} \times 100\,\text{MHz} = 400\,\text{MB/s}$ for both directions. Both directions share the $400\,\text{MB/s}$, so each direction only gets $200\,\text{MB/s}$. By adding in communication overhead, the measured GPIF throughput of $117\,\text{MB/s}$ in the PCIe FX3 emulator test seems reasonable.

The maximum throughput of the PCIe test setup is $896\,\text{MB/s}$. Since this maximum is much greater than the Aurora or PCIe with FX3 emulator tests, it can be inferred that PCIe was not the bottleneck in the other tests. The PCIe test shows the upper throughput limit with the PCIe implementation used in the communication board. If more bandwidth is needed, then 64-bit streams can be used. The maximum throughput of PCIe 64 is $1511\,\text{MB/s}$. In order to reach the maximum throughput, a transfer size of 1K and 2K is needed for PCIe and PCIe 64, respectively. All the implementations have the same rate of change in the beginning region before the maximum is achieved. This means that the Xillybus PCIe bus transfer is the limiting factor and that the limit is the same for 64-bit as it is for 32-bit.

One lane of Aurora achieves a maximum throughput of $345\,\text{MB/s}$. Moving from one lane to two lanes roughly doubles the maximum throughput to $669\,\text{MB/s}$. Aurora should maintain this trend as more lanes are added until its throughput starts to match the limits of the PCIe implementation. There is an interesting artifact in the data as the PCIe transfer size starts to exceed 32768 bytes. The throughput starts to drop and the variance in the data greatly increases. This can be caused by exceeding the size of the buffers on the communication board or from exceeding the size of the host DMA buffers. PCIe 64 shows a similar dip in performance, but the change happens with a larger transfer size.

The addition of a Go-Back-N ARQ only added a slight decrease in bandwidth with a maximum throughput of $328\,\text{MB/s}$ for one-lane of Aurora. This is only a 5.5% decrease in throughput caused by the overhead of adding packet numbers and sending acknowledgments.

From this graph, many helpful conclusions can be made. First, the new communication board has room to scale. If

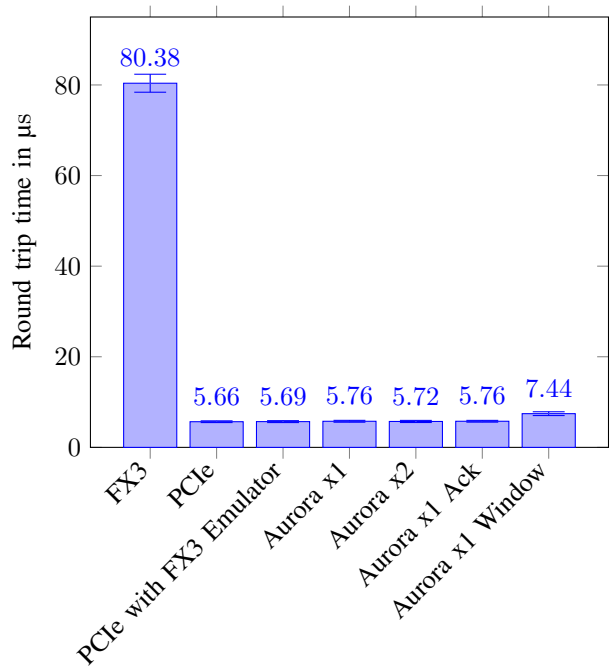

Fig. 3. Round trip time comparison.

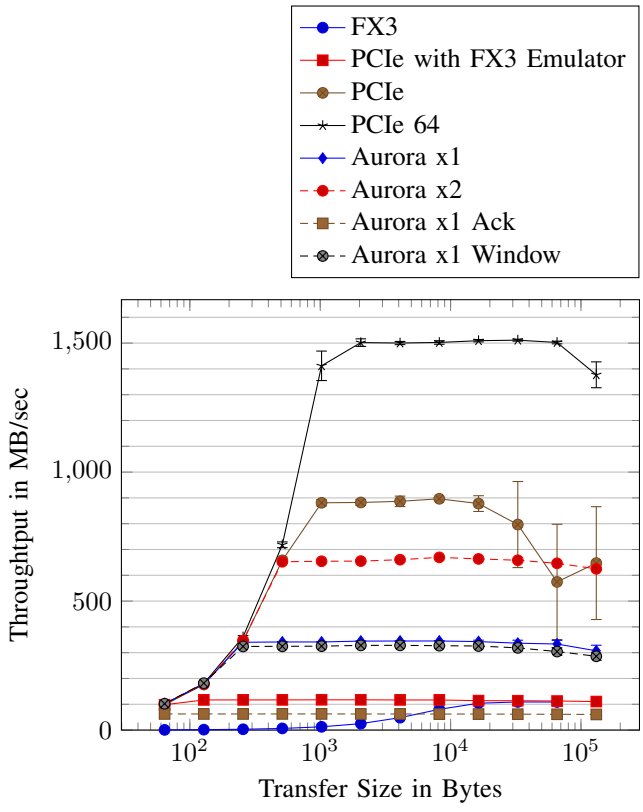

Fig. 4. Throughput experiment, varying the amount of data transmitted per function call.

the single lane Aurora limit is reached, then two or more lanes can be used. If the PCIe limit is reached, 64-bit PCIe can be used. Once the 64-bit PCIe limit is reached, Xillybus Revision XL, which offers a maximum throughput of 3500 MB/s with a 128-bit internal data width, can be used [19]. The new communication setup can scale far beyond the previous communication limits of the FX3 and GPIF interface. Second, the maximum throughput is only reached when large blocks are transferred at a time, with the sweet spot seeming to be 1 KB of data.

Additional tests, which varied the total transfer size and kept the transfer size fixed to a single 64-byte packet, showed that performance is highly dependent on the buffer size used to call the transfer function and not on the total amount of data being transferred. This means that the best performance is achieved by buffering multiple packets together and making large transfer calls to the Xillybus driver.

VIII. Future Work

Now that a new communication board has been designed and tested to show high communication performance with room to scale, the communication board can be extended to become a feature-rich neuromorphic array communications controller and a hyper-scale interconnect allowing multiple neuromorphic arrays to be connected together in a scalable manner. The first level of scaling is to connect multiple neuromorphic boards together with local connections. The communications board will have to be extended to handle data between the host and each of the neuromorphic boards. Additional synchronization logic will also have to be added to ensure that temporal sensitive inputs occur at the same time across each board.

Once local scaling between neuromorphic boards with a single communication board has reached its limit, the scaling can be continued with regional scaling comprised of multiple communication boards each with a local group of neuromorphic boards. At this level of scaling, the communication boards will have to support communication between themselves along with any needed synchronization between the communication boards. With help from the communication boards, the neuromorphic arrays can be scaled linearly in multiple dimensions. Direct links for spiking communication will be used between devices in the local regions, whereas the communication boards will transfer spikes between regions. These spiking communications across high-speed interconnects will allow linear scaling to hyper-scale neural networks (on order of billions of synaptic elements). An additional level of scaling can be achieved by adding multiple hosts, each with multiple communication boards and each communication board connected to multiple neuromorphic boards. The hosts would then be connected via Ethernet or some other communication channel to form a neuromorphic supercomputer.

In addition to forming a communications plane for hyper-scale neural networks, the functionality of the neuromorphic array communications controller can be extended to perform or assist in the tasks performed by the host computer. These tasks include configuring the neuromorphic boards to act as one

large array or as multiple smaller independent arrays, as well as using multiple arrays to perform continuous optimization of the network via dynamic learning and optimization using genetic algorithms.

IX. Conclusion

A new communications system for neuromorphic arrays using a separate communications board was designed and its performance measured. The new communications system was shown to out-perform the prior FX3-based communications setup and can be used to scale-up to communicate with multiple neuromorphic boards simultaneously. PCIe is used to connect the host machine to the communication board and Aurora is used to connect the communication board to the neuromorphic boards. Both have been benchmarked and shown to have much higher throughput and lower round trip latency than the FX3 communication setup. The new communication board offers more flexibility, both in terms of the ease in which the communication packet structure can be modified and in terms of how the boards can be connected. The communications board has sufficient performance to facilitate the desired interaction between traditional von Neumann computers and new neuromorphic systems. It also has sufficient room to scale up to be used as a high-speed communications interconnect for a hyper-scale neuromorphic array.

References

[1] C. D. Schuman, "Neuroscience-inspired dynamic architectures," PhD thesis, University of Tennessee, May 2015.

[2] M. E. Dean and C. Daffron, "A VLSI design for neuromorphic computing," in *IEEE Annual Symposium on VLSI (ISVLSI)*, IEEE, Jul. 2016. DOI: 10.1109/ISVLSI.2016.81.

[3] M. E. Dean, J. Chan, C. Daffron, A. Disney, J. Reynolds, G. S. Rose, J. S. Plank, J. Birdwell, and C. D. Schuman, "An application development platform for neuromorphic computing," in *International Joint Conference on Neural Networks*, Vancouver, Jul. 2016.

[4] Stanford University. (2006). Neurogrid, [Online]. Available: https://web.stanford.edu/group/brainsinsilicon/challenge.html.

[5] P. Merolla, J. Arthur, R. Alvarez, J.-M. Bussat, and K. Boahen, "A multicast tree router for multichip neuromorphic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 820–833, 2014.

[6] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J.-M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, 2014.

[7] T. Sharp, F. Galluppi, A. Rast, and S. Furber, "Power-efficient simulation of detailed cortical microcircuits on spinnaker," *Journal of neuroscience methods*, vol. 210, no. 1, pp. 110–118, 2012.

[8] S. Furber and A. Brown, "Biologically-inspired massively-parallel architectures-computing beyond a million processors," in *Application of Concurrency to System Design, 2009. ACSD'09. Ninth International Conference On*, IEEE, Jul. 2009, pp. 3–12. DOI: 10.1109/ACSD.2009.17. [Online]. Available: https://eprints.soton.ac.uk/270985/1/PID871138.pdf.

[9] J. Schemmel, D. Briiderle, A. Griibl, M. Hock, K. Meier, and S. Millner, "A wafer-scale neuromorphic hardware system for large-scale neural modeling," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 1947–1950. DOI: 10.1109/ISCAS.2010.5536970.

[10] S. Scholze, S. Schiefer, J. Partzsch, S. Hartmann, C. Mayr, S. Höppner, H. Eisenreich, S. Henker, B. Vogginger, and R. Schüffny, "Vlsi implementation of a 2.8 gevent/s packet-based aer interface with routing and event sorting functionality," *Frontiers in Neuroscience*, vol. 5, p. 117, 2011, ISSN: 1662-453X. DOI: 10.3389/fnins.2011.00117. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fnins.2011.00117.

[11] S. Scholze, H. Eisenreich, S. Höppner, G. Ellguth, S. Henker, M. Ander, S. Hänzsche, J. Partzsch, C. Mayr, and R. Schüffny, "A 32gbit/s communication soc for a waferscale neuromorphic system," *Integration, the VLSI Journal*, vol. 45, no. 1, pp. 61–75, 2012.

[12] A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman, *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, 2013, pp. 1–10.

[13] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, IEEE Computer Society Press, 2012, p. 54.

[14] A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza, *et al.*, "Cognitive computing programming paradigm: A corelet language for composing networks of neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, IEEE, 2013, pp. 1–10.

[15] J. S. Plank, G. S. Rose, M. E. Dean, C. D. Schuman, and N. C. Cady, "A unified hardware/software co-design framework for neuromorphic computing devices and applications," in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, Nov. 2017.

[16] J. Chan, "Implementation of a neuromorphic development platform with DANNA," Master's thesis, University of Tennessee, 2015.

[17] A. Young, "Scalable high-speed communications for neuromorphic systems," Master's thesis, University of Tennessee, 2017.

[18] Xillybus Ltd., *Product brief*, Feb. 16, 2017. [Online]. Available: http://xillybus.com/downloads/xillybus_product_brief.pdf.

[19] ——, (2017). Revision b/xl user notes, [Online]. Available: http://xillybus.com/doc/revision-b-xl (visited on 06/12/2017).

[20] A. Athavale and C. Christensen, *High-Speed Serial I/O Made Simple*. Apr. 2005. [Online]. Available: https://www.xilinx.com/publications/archives/books/serialio.pdf (visited on 06/07/2017).

[21] M. Desai and K. Sivaramakrishnan, *Optimizing usb 3.0 throughput with ez-usb fx3*, AN86947, version C, May 9, 2017.