Interconnect for commodity FPGA clusters: standardized or customized?

A. Theodore Markettos, Paul J. Fox, Simon W. Moore, Andrew W. Moore
Computer Laboratory, University of Cambridge, UK
{firstname.lastname} @cl.cam.ac.uk

***Abstract*—We demonstrate that a small library of customizable interconnect components permits low-area, high-performance, reliable communication tuned to an application, by analogy with the way designers customize their compute. Whilst soft cores for standard protocols (Ethernet, RapidIO, Infiniband, Interlaken) are a boon for FPGA-to-other-system interconnect, we argue that they are inefficient and unnecessary for FPGA-to-FPGA interconnect. Using the example of BlueLink, our lightweight pluggable interconnect library, we describe how to construct reliable FPGA clusters from hundreds of lower-cost commodity FPGA boards. Utilizing the increasing number of serial links on FPGAs demands efficient use of soft-logic, making domain-optimized custom interconnect attractive for some time to come.**

I. Introduction

FPGA systems are hard to scale. A designer can use the largest FPGA money can buy but this comes with a significant price penalty – as shown in Figure 1. Furthermore, the largest FPGA available may only give a factor of two or four improvement. A large workload may outgrow the FPGA by a factor of a hundred or more.

Additionally a large FPGA does not necessarily increase resources. If a problem is memory-bound, the number of external memory interfaces may remain constant as the FPGA gets larger. Packaging limits constrains the number of I/O pins so more DIMMs cannot be attached. Sooner or later a designer is forced to move to a multi-FPGA system.

In this paper we describe an approach to building FPGA clusters at scale, using commodity parts to minimise costs and engineering, and high-bandwidth serial transceivers for interconnect. We then consider the protocols for interconnect.

A natural assumption would be to use a standard protocol such as Ethernet using standard soft cores. We assess whether such a *standardized* communication system makes sense, or whether it is worth building a *customized* interconnect tailored to the application requirements.

We illustrate the question with BlueLink, a custom FPGA interconnect toolkit we designed for a specific application. We compare this with standard soft IP cores to evaluate the merits of a custom approach.

Furthermore we describe how custom interconnect can make best use of the commodity FPGA platforms available and continue to scale in the future.

II. Building an FPGA Cluster

When building a multi-FPGA system, the obvious approach is to put multiple FPGAs on the same printed circuit board (PCB). After all, FPGAs have hundreds of general-purpose I/O (GPIO) pins which can be used to connect them.

However there are a number of pitfalls to a multi-FPGA PCB. Firstly, designing such a board is a complex task. FPGAs have upwards of 1000 pins to route, many at high speed. The Altera Stratix V PCIe Development board has 16 layers [3]. FPGA power design is also complex – this single-FPGA board has 21 power rails with the highest current being 28 A. This requires complex design and simulation – for professional designers a board takes about one man-year of effort to design. FPGAs are typically found in advanced ball grid array packages – which also makes manufacturing difficult. In addition there is the headache of managing the whole process of parts procurement, production, test and debug.

Secondly, many such boards (especially commercial products) are not regular – each FPGA is not connected to the same peripherals. That means a separate synthesis run for each FPGA in the cluster, which makes it difficult to scale to large numbers of FPGAs. Testing such boards is difficult, a fault may cause the whole board to fail, and repair is complex.

For example Mencer et al. [11] used 64 Spartan 3 FPGAs on a large 8-layer PCB (320×480 mm). With 64-bit connections at 100 MHz between FPGAs they achieved 6.4 Gbps inter-FPGA bandwidth. They had to employ fault tolerance because replacing a faulty device is difficult. Furthermore, the engineering required for such boards makes them niche commercial products with high price tags. The DINI Group quote 'below 0.1 ¢ per [ASIC] gate' for a '130 million ASIC gate' system containing 20 Stratix IV 530 devices – which makes the board cost around US$130,000 [5].

Meanwhile, FPGA evaluation cards have become a commodity. These are boards commonly sold to engineers to prototype their design before the final product, but they are increasingly being used as standalone platforms for research and development. The non-recurrent expenditure (NRE) from design and tool costs is amortized across the thousands of units being shipped, reducing the unit cost. If a board fails, it can simply be swapped out for another costing a few hundred or thousand dollars. It therefore makes economic sense to use many commodity cards with some kind of interconnect. If this allows smaller FPGA parts, that ship in greater numbers and have better yield, it will further reduce cost. For example, in Figure 1 the Cyclone V parts are roughly one sixth of the price of the comparable Stratix V.

A. Application Partitioning

When building a cluster, we must consider the applications to run on it before we design the interconnect.

Some applications do not require any communication between FPGAs. These can be described as *loosely coupled*.

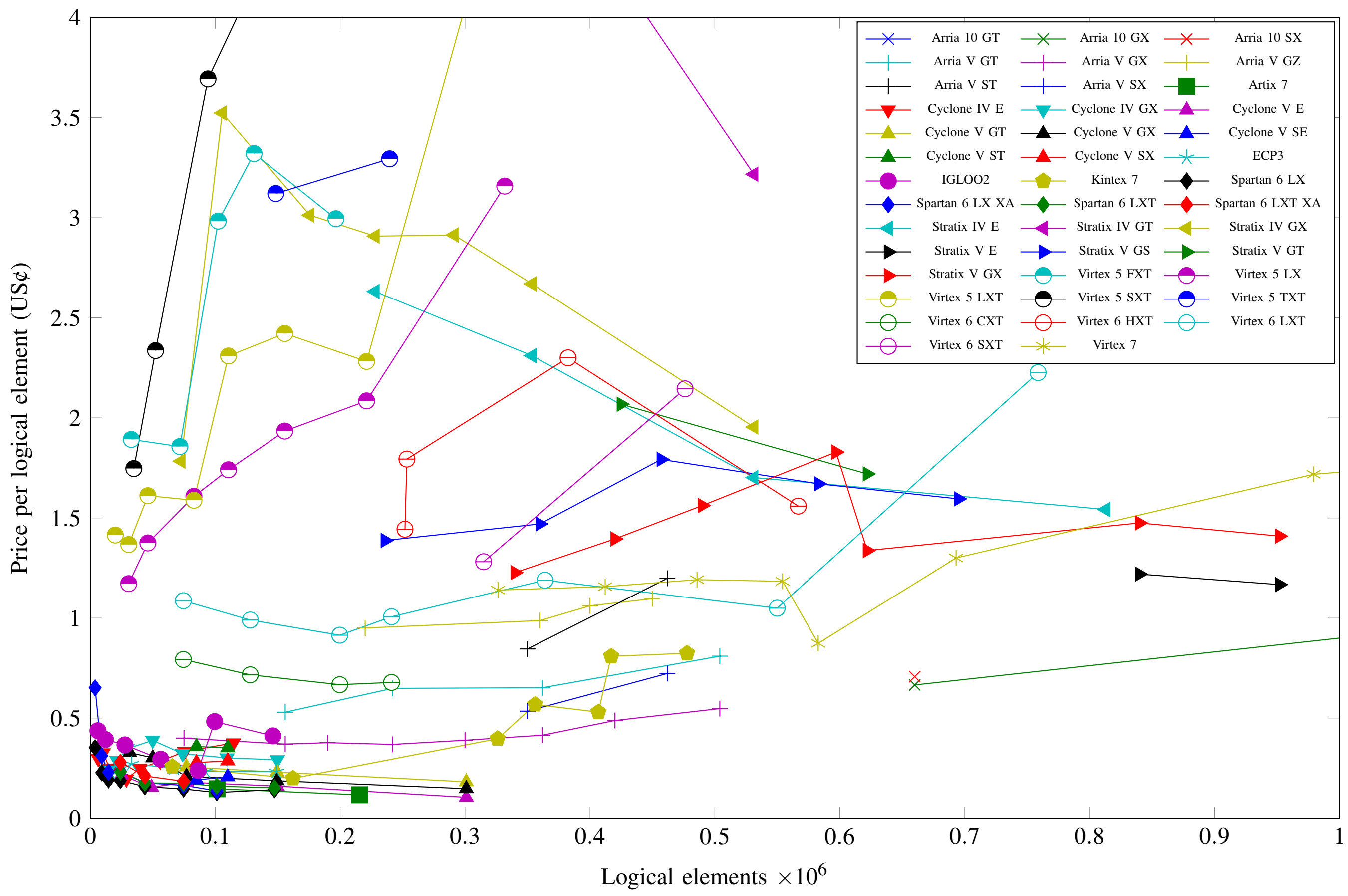

Fig. 1. FPGA pricing trends. Devices cluster in two board categories: smaller budget ranges with a lower cost per logic element, and premium parts which are considerably more expensive for the same number of elements. Within the budget range larger parts are cheaper per element. In some families in the premium range larger devices are disproportionately more expensive. Economics suggests that a cluster should use budget ranges wherever possible or alternatively smaller premium devices. Data: digikey.com [4]; we plot the median of prices for a given model and size, combining options of package and speed grade into a single point.

MapReduce fits this model, while FPGA examples include Bitcoin mining or OpenCL-based accelerators. It only needs a connection to a host PC, and scale is achieved by buying more PCs with FPGAs. Such an FPGA cluster is easy to build.

Other applications are *tightly coupled*. One example is gate-level system-on-chip simulation. This is very latency-dependent: node operating in lock-step require single-cycle interconnect latency. This makes partitioning a hard problem, compounded by many possible fine-grained partitions.

To allow scale, it is better to have a coarser-grained architecture. If the number of possible subdivisions is reduced, partitioning becomes simpler. A higher level of abstraction may also permit relaxation of the latency requirements, though latency can still be a major bottleneck. Less efficient area usage can be compensated as more hardware is easier to add.

Additionally, a tiled approach can reduce FPGA synthesis times. If the FPGA bitfile is the same for each node, it only needs to be synthesized once. Different behaviour for each node can then be configured with different software, datasets and runtime configuration.

B. Physical Interconnect

We wish to build a cluster at scale, using hundreds of FPGAs on multiple boards. If connections between FPGAs are required, how should such a cluster be interconnected?

The simplest approach would be to connect the GPIO pins. These can be driven either single-ended or with low-voltage differential signalling (LVDS). However, the frequency they can be driven is limited (to about 1 GHz in LVDS mode). Long parallel links are subject to signal integrity (constrains cable geometry for a good quality signal) and skew (signals arrive at different times). This means such cables are typically short (centimetres) and must employ careful (expensive) construction. This limits the size of cluster that can be built. Kono et al. [8] achieved a data rate of 4 Gbps per link using the HSMC connectors on a Terasic DE4 board using expensive proprietary ribbon cabling. With only two ports per board their cluster topology was forced to be a ring.

FPGAs now incorporate increasing numbers of high-speed serial transceivers. A device can have up to 96 transceivers each capable of up to 56 Gbps (though 14 Gbps is a more realistic maximum for lower cost parts). Many commodity I/O standards have shifted from parallel to serial interconnect (such as USB, SAS, SATA, PCI Express, etc). This means there are now cheap passive multi-gigabit serial cables on the market. Active repeater and optical cables are also available for longer distances. Such cables can be used as physical-layer bit-pipes, without using the intended protocol along them. All that is required is point-to-point cabling between whatever connectors the board manufacturer provided.

Therefore we suggest that a cluster can be built at scale with the following properties:

- Commodity FPGA boards, to reduce cost and development time.
- Serial interconnect using FPGA transceivers.
- Low cost commodity passive copper cabling between boards. If necessary, optical cabling for longer distances.
- Multiple-hop routing, so that a fully-connected network is not required.

III. Custom Communication?

The question that remains is: what protocol should be used on this interconnect? Should it follow a standard, or is it worth designing your own? An FPGA designer may be comfortable with the idea of *custom compute*, where their compute is optimized for the workload. This usually works much better than simply using a standard CPU soft-core on their FPGA. A natural extension of this would be *custom communication*, where the communication is similarly optimized. Is it worth optimizing your communication, or is standard IP sufficient?

We shall consider a number of application examples, and a communication system that was designed for them. We will then compare with existing standards to identify the merits and pitfalls of each system.

IV. Application Case Studies

The compute and communication requirements of a FPGA cluster may be different from other clusters such as datacenters or PC-based scientific compute. The following examples describe two applications that are suited to FPGA clusters and their communication requirements.

A. Memory interconnect

Consider a massive multiprocessor system using shared memory. A number of CPU cores (such as NIOS-II/Microblaze or custom processors) are located on each FPGA. Each FPGA board contains up to 8 GB of DRAM. When a CPU core needs to access memory stored on another board, it must request a cache line from the other board. Each cache line might be 256 bits which is set by the width of the interface to the memory controller. Thus a memory read consists of sending a 64-bit address and receiving a 256-bit response, or writing a 64-bit address and 256-bit value. Superscalar CPU architecture can mask a limited amount of memory latency, up to a few tens of cycles. A lost or further delayed memory transaction will cause a CPU to give an incorrect result or stall.

B. Neural computing

The human brain has approximately 10^{11} neurons with 10^{14} synaptic connections. Each neuron fires at about 10 Hz. In some neuron models, neuron updates can be represented by a simple differential equation, but there are approximately 10^{15} synaptic messages per second. To achieve real-time operation the network must compute the state of every neuron, accounting for its 10^{3} incoming messages, every millisecond.

The need for timely delivery of large numbers of small, low-latency messages rules out classical CPUs, which do not have enough compute, and GPUs, which do not have enough communication, but is a good target for FPGAs.

Using the Izhikevich neuron model, the state of each synaptic message can be represented in 48 bits [12]. Critical neuron parameters fill the FPGA BRAM, so space for packet buffers (for both message coalescing and retransmits) is very limited. With 128K neurons per FPGA, each FPGA generates 1.28M 48-bit synaptic messages per millisecond with a real-time deadline of arriving by the end of the next millisecond.

The worst-case throughput is therefore 1.28 billion messages per second from each FPGA. Due to spatial locality, some of these messages are for neurons that reside on the same FPGA and so can be stored in off-FPGA DRAM – the exact proportion depends on the neural network being simulated. The throughput requirements are therefore some percentage of this figure.

V. Communication System Requirements

In these application examples payload sizes are small (48 to 256 bits) and the application is latency critical. Furthermore, the application does not have inbuilt support for retransmission: if a cache line request is dropped the CPU will simply stall, while a neural message being dropped will cause an inaccurate computation.

When building our FPGA cluster, these applications led us to list the following requirements:

1) Small message sizes: the interconnect must be able to efficiently deal with messages between 32 and 256 bits.

2) Low latency: cluster applications are often more constrained by latency than bandwidth.

3) Reliable: with thousands of links each running at gigabits per second, errors are inevitable and could cause crashes or invalidate results.

4) Hardware-only: the interconnect must support reliable packet delivery in hardware, without leaving reliability to software layers (as in TCP/IP).

5) Lightweight: the interconnect must take minimal FPGA area. This leaves more space for compute and permits smaller cheaper FPGAs.

6) Ubiquitous: the interconnect must maximize use of the FPGA's transceiver resources. More links and higher link rates means more bandwidth and fewer hops across the system.

7) Interoperable: the interconnect should be able to connect FPGAs of different types to build a heterogenous cluster.

VI. BlueLink: a Custom Interconnect Toolkit

To address the communication requirements of the applications on our cluster we created the BlueLink interconnect toolkit. An overview of the interconnect is shown in Figure 2. It has five major layers, written in Bluespec SystemVerilog:

1) Serial Transceiver: A hard IP block provided by an FPGA manufacturer. It is assumed that it implements 8b10b coding and can be configured to send and receive 32-bit words with a 4-bit k symbol indicator. BlueLink makes no assumptions about the properties of a transceiver beyond its

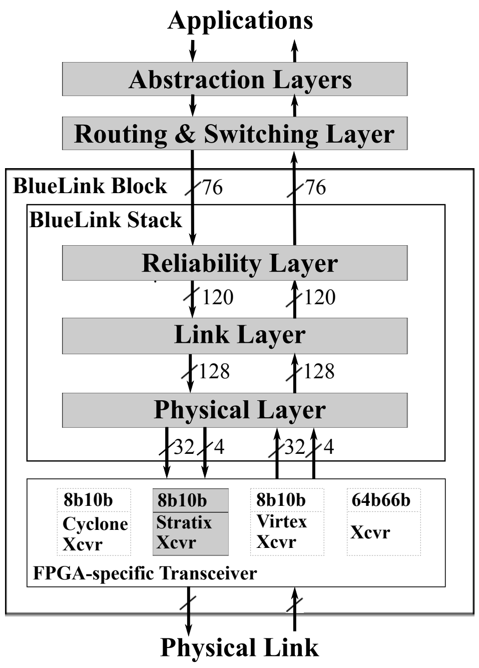

Fig. 2. Architecture of BlueLink interconnect

ability to successfully send and receive these 8b10b symbols. Alternatively another coding scheme such as 64b66b could be used. We have versions for Altera Stratix IV and Stratix V – it should be straightforward to wrap FPGA transceivers from other manufacturers.

BlueLink can use whatever transceivers are available on the FPGA board, over any physical medium. Currently SATA, PCIe, SMA, SFP+ copper and SFP+ optical cabling has been tested.

2) Physical: Transforms a FIFO-like stream of words from the Link layer into a continuous stream of words for the serial transceiver. Idle symbols and any alignment symbols required by the serial transceiver are inserted and removed as needed.

3) Link: Serializes 128-bit flits into 32-bit words on transmit and aligns these words back into flits on receive. Also performs clock crossing between the main FPGA clock domain and the transmit and receive clock domains of each transceiver.

4) Reliability: Implements reliable transmission with ordering and back pressure.

5) Routing and switching: Uses a hop-by-hop routing scheme to direct packets to a given FPGA and port.

6) Application: Provides primitives for applications.

The Reliability and Application layers are described in further detail below.

The unit of reliable data transmission is a *flit* with a 64-bit payload and 12-bit addressing field. This is expanded to 120 bits by the reliability layer by addition of a 32-bit CRC, a sequence number and an acknowledgement field. The physical layer adds a further 8-bit header, so that 128-bit flits are sent and received by the FPGA transceivers, often split into 4×32-bit words.

A. Reliability layer

The reliability layer is the first layer in the stack which is more than transforming and aligning symbols. It is tailored to meet the requirements identified in Section V.

It implements a reliable communication channel with FIFO semantics, providing a similar service to the TCP layer of an Internet stack. However it is customized for small packet transmissions and for low FPGA area. This means it must be economical with both header fields and memory buffers.

Reliability is implemented using a CRC and sequence number in each flit, which are validated by this layer in the receiving reliable transceiver. A 32-bit CRC is used because the probability of false-negative is high in a large cluster with billions of flits per second. An acknowledgement number may also be appended to a flit to acknowledge correct receipt of a flits with that sequence number. If the receiver receives a flit which either fails the CRC or that is out of sequence, it does not send an acknowledgement. If there are pending acknowledgements to be sent but no input flits, an empty flit is sent for each acknowledgement.

Reliability is window-based, with transmitted flits that have not yet been acknowledged being stored in a replay buffer. If a flit is not acknowledged after a timeout (because the receiver detected an error or because an acknowledgement was lost), the flit at the head of the replay buffer is sent continuously until it is acknowledged, followed by every other flit in the buffer until the whole window of flits has been acknowledged. New flits are then accepted from the input. With 4 bit sequence and acknowledgement numbers the replay buffer only needs to hold 8×64 bit flits to store a whole retransmission window, a major contributor to the reduced FPGA area requirements compared to other protocols that have longer flits/packets and larger windows.

Backpressure is achieved by sending acknowledgements with a flag to indicate that no more flits can be accepted. This prevents any further flits being transmitted, and so leads to the reliable transceiver's input FIFO becoming full.

B. Application abstractions

The Reliability layer provides an Avalon Streaming interface to its clients. On top of this we have implemented a number of application abstractions which match different communication paradigms and levels in the design hierarchy, for ease of programming.

1) Bluespec FIFO: Bluespec is a dataflow hardware description language. Hardware modules are often connected using a FIFO abstraction rather than Verilog wires. This enables them to be easily decoupled at the same time as reducing to simple logic structures. BlueLink provides a Bluespec FIFO type that can be used to join two modules on different FPGAs. The only difference is 10-20 extra cycles of latency compared with an on-chip FIFO.

2) Packet abstractions: BlueLink is also usable as packet-based interconnect from software on custom processors. Hardware provides access to flit send and receive buffers. Traditional polling or interrupt mechanisms may be used to inform a target machine of packet delivery.

3) Blocking reads and writes: A lower-latency alternative to polling or interrupts is for a read or write to the flit buffer to block the application until it is performed successfully. This has lower overhead than polling as it is not necessary to spin in a loop until an operation can be performed. There is, however, a deadlock risk. Additionally it is possible to indicate a target FPGA and port by using part of the address of a write, which allows a flit to be sent in a single clock cycle.

A simple demonstration of this has been achieved by having a NIOS-II CPU executing code from DRAM located on another board. When the link cable is unplugged, the CPU pauses. When the cable is re-attached, the links resynchronize and the CPU continues.

4) Remote DMA: A higher-level abstraction maps a wide region of memory addresses on each FPGA to a hardware module that performs remote DMA. Any read or write is translated to a read or write to a region of memory (or a memory-mapped peripheral) on a remote FPGA. A series of packets is sent to the hardware module on the remote FPGA, which performs the operation and returns the result as if it were a local operation.

Burst reads and writes are supported, enabling block transfers. Since it is not possible or desirable for an application to be aware of the details of the remote FPGA's memory map, such as the word size of a given memory device, bursts are translated into an appropriate sequence of operations at the remote device, including using byte enables with writes if a request does not align with word boundaries.

5) Software pipes: We also have an abstraction layer that emulates Linux pipe semantics. An application can be tested on a PC using Linux pipes between processes, then ported to the cluster and run unchanged.

Using the different abstractions provides a variety of primitives for partition. For example, the FIFO abstraction allows a hardware dataflow architecture to be split across FPGA boundaries, while the remote DMA abstraction means partitions can be viewed as nodes in a cluster-wide shared memory architecture.

VII. IP FOR STANDARDIZED PROTOCOLS

IP cores for standard communication protocols are commercially available from a number of vendors.

A natural assumption of a designer building a FPGA cluster would be to use a popular protocol such as Ethernet. Ethernet is today a switched serial interconnect with data rates up to 100 Gbps. Interface IP, switches and cabling are commodity items. It is well understood, and is a convenient way to connect an FPGA cluster to a host PC. Some FPGA clusters such as the image retrieval accelerator in [10] are loosely coupled with no inter-FPGA communications. In this case Ethernet to a host PC may be a good fit for their application.

There are other protocols for which FPGA IP is available: Serial RapidIO, Infiniband, Interlaken, Fibre Channel, PCI Express and many more. We compare characteristics of a selection of standard IP in Table I.

Notably the field can be divided into those protocols that support in-built reliability by packet retransmission, and those that do not. The performance of these vary widely, both in terms of physical link rate[1] and area requirements.

[1]The computation of achievable bandwidth for diverse protocols is complex, so we use link rate as a simple yardstick in this section. We provide more detailed case studies in our evaluation in Section VIII.

Ethernet has some restrictions for applications with tighter coupling. For example, [14] uses 37-bit payloads over Ethernet. To use the links efficiently they must aggregate these into packets, which results in latencies of 10 µs or more.

In addition Ethernet provides no native guarantee of packet delivery. In a cluster there may be thousands of links sending gigabits per second, so errors are inevitable and reliability is a necessity. TCP/IP is the conventional means of retransmission, but is very expensive to handle in hardware [7]: clusters [10] and [14] did not consider it. For latency-sensitive applications, handling reliability in software is not an option. An alternative reliable protocol could be implemented on top of Ethernet – another example of a custom communication system.

PCI Express is commonly used for connecting FPGAs to a host PC. However it introduces a lot of complexity, being an emulation of traditional PCI over switched interconnect. For this reason FPGAs often have PCIe hard cores – but it is unusual for an FPGA to have more than one.

Interlaken is commonly used as a backplane interconnect in high-end switches. It is also very scalable, and relatively lightweight. There is also an optional retransmission extension. We tried to implement an Interlaken layer as an alternative to BlueLink, but came across the constraint that Altera's Stratix V IP requires groups of eight or twelve bonded links to implement 50G or 100G channels. This was incompatible with the physical topology of the commodity Stratix V boards available to us. Altera also provide an alternative Interlaken core which requires groups of four or more channels, but only works on the Stratix IV and has no reliability support.

Altera's SerialLite is an example of a lightweight vendor-provided protocol. SerialLite shares some similarities with BlueLink: SerialLite II provides packet retransmission of small packets. However it has been somewhat neglected - while it has been ported to modern FPGAs, its maximum channel rate is 6 Gbps. Published area numbers are for Stratix II, and it is commercial IP for which a license is required. It is also incompatible with non-Altera FPGAs. We managed to synthesize a 6G SerialLite II core on a Stratix V, but the licensing restrictions did not allow us to test it on an FPGA.

SerialLite III is a modern version that runs at 10 Gbps and beyond, however the protocol has been changed to support forward error correction preventing single bit errors. Across a cluster, where there may be thousands of links, cabling faults causing more substantial errors are likely so this protection is insufficient for our requirements. It is therefore only useful as a layer that does not guarantee correct packet transmission.

Aurora is Xilinx's equivalent to SerialLite, but has no reliability layer. This was used in systems such as an FPGA cluster [2] and an SoC prototyping system [9]. In both cases bit errors limited the usable interconnect data rate.

It became clear that using standard communication cores in an FPGA cluster can be fraught with practical difficulties:

1) Configuration constraints: available parameters such as data rate and number of bonded lanes may not be appropriate.

2) Fitting requirements: a standard may require particular clock frequencies, PLLs or clock routing.

3) Bonded links: useful on a custom PCB with skew-free parallel lanes between FPGAs. A commodity board and serial cabling may not have suitable configuration, either by not enough lanes, unsuitable placement, or skew over different cables. Bonding can reduce the dimensions of the cluster compared with single links, adding hops and thus latency.

4) Manufacturer specific: some protocols such as SerialLite and Aurora are only supported by one FPGA manufacturer. It is possible to implement these protocols on other FPGAs by reimplementing their specifications, but this would involve another IP vendor or a custom implementation.

5) FPGA support: IP may only support some FPGA families, may be withdrawn in new tools, or not updated for new devices. It may require extensive reworking or prohibit using a newer FPGA.

6) Licensing: designers must license IP from vendors, which can be expensive and can make evaluation difficult, particularly as a simulation of a link does not capture physical effects and so a license may be required for evaluation on a physical FPGA.

VIII. Evaluation

We evaluated BlueLink by synthesizing it on a Stratix V GX FPGA on a Terasic DE5-Net board and comparing with an implementation of Altera's existing 10G Ethernet MAC. The Stratix V platform was chosen to make a fair comparison between Ethernet and other existing standards that use 10G links – BlueLink is also capable of using lower-speed, lower-cost FPGAs at 3G or 6G where Ethernet is often limited to 1G. Ethernet does not provide reliable transmission while BlueLink does, so in practice another layer would be required above Ethernet. We attempted implementations of SerialLiteII and Interlaken but these were frustrated as described above.

The area comparison can be seen in Table II. 10G BlueLink is 65% of the logic and registers of 10G Ethernet – indeed 40G BlueLink using bonded lanes will fit in about the same area as a single 10G Ethernet MAC. BlueLink also uses 15% of the memory of 10G Ethernet. By comparison with standard IP on Stratix V in I BlueLink is more efficient than all the standards that support reliability and the majority that do not.

In terms of throughput, we have shown the overhead of BlueLink and Ethernet-based packet structures in Figure 3. Our focus is on small packets, but BlueLink has higher throughput up to 256 bits. Using IP and/or TCP over Ethernet for reliability only serves to add additional overhead.

We plot the latency of BlueLink compared with Ethernet in Figure 4. We compare the latency of a link where the input queue is empty, and one where the link constantly receives input as fast as it can transmit. Both are tested on short physical links that have low error rates. Despite addition of a reliability layer with CRC checking, BlueLink's latency is about equivalent to Ethernet in the fully-loaded case. In the lightly-loaded case, BlueLink's latency is much lower as flits can be accepted in a single cycle, rather than nine cycles that Altera's Ethernet core takes. As more transceivers are used on an FPGA it becomes more likely that links can be operated in this state where they are not fully congested.

Any FPGA system designer is faced with an area/performance tradeoff. This is particularly acute in modern

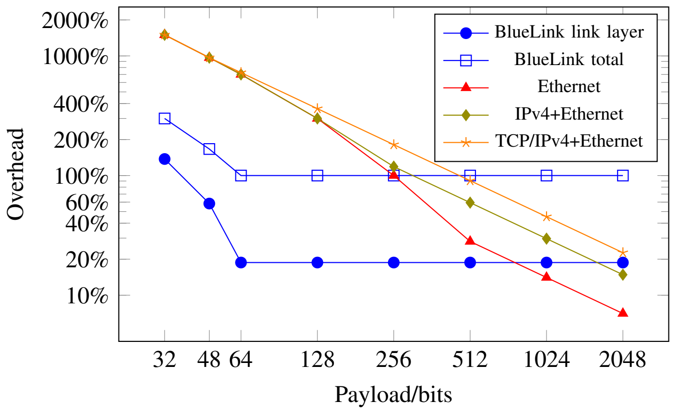

Fig. 3. Overhead of BlueLink against Ethernet-based standards for small packets. BlueLink makes considerably better use of bandwidth up to 256-bit packets.

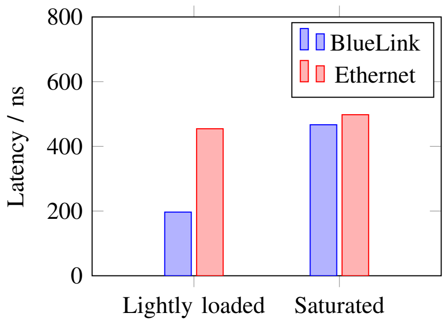

Fig. 4. Latency comparison between BlueLink and Ethernet MAC at 10 Gbps. BlueLink's latency compares favourably especially if many transceiver links can be used to keep the links lightly loaded.

FPGAs which have many transceivers. For comparison we take a Stratix V GX A7 FPGA, which is the lowest cost Stratix V that Terasic sell on an evaluation board. This FPGA has 48 transceivers each rated at 14.1 Gbps. We consider the situation that the designer wishes to use all the available transceivers. In Figure 5 we plot the area of FPGA that will be required for the different standards, against the raw bandwidth it will provide. All standards are limited to 10 Gbps per lane because this is the limit of commodity cabling (in theory BlueLink and SerialLite III will go higher). As can be seen, many standards have a considerable area penalty compared to a lightweight custom protocol such as BlueLink.

A. Application example

We used BlueLink as a key enabler for the Bluehive neural computation engine [12]. BlueLink was implemented on the DE4 Stratix IV 230 GX FPGA board from Terasic, which was chosen to maximise the number of DDR2 memory channels available. This is the middle of the Stratix IV range, much lower cost than high-end parts. To interconnect the boards we designed and open-sourced [17] a PCB to break out transceivers using PCI Express connectors into 6 Gbps SATA links (Figure 7). This enabled us to create a *pluggable topology* of low-cost SATA cables. Additional SATA cables were used directly in the FPGA boards' own SATA sockets.

System	Raw external link rate	Configuration	Constituent lane rate	LUTs	Registers	Memory bits
Systems with reliable transmission						
TCP/IP (in hardware) + Ethernet [7]	10G	1 lane	10G	<30000	Not quoted	Not quoted
TCP datapath acceleration [6] (Virtex 6)	10G	excluding CPU/MAC/PHY		6875	3889	221184
SerialLite II (Stratix II 16 bit CRC)	6G	1 lane	6G	1448	1236	90624
	24G	4 lanes	6G	2573	1659	176640
PCIe hard IP	5G	1x Gen 2	5G	100	100	0
	40G	8x Gen 2	5G	200	200	0
PCIe soft IP (Stratix IV)	5G	1x Gen 2	5G	5500	4100	82944
	20G	4x Gen 2	5G	7100	5100	239616
Serial RapidIO	5G	1x	5G	5700	7885	737280
	20G	4x	5G	7200	10728	901120
Fibre Channel (Stratix IV) [13]	8G	1x	8G	3300	3900	6144
Infiniband [15]	40G	LLC+TCA QDR 4x	10G	64105	63185	1584846
Systems that do not implement reliable transmission						
Infiniband [15]	40G	TCA QDR 4x	10G	36658	39912	1536807
SerialLite II (Stratix II)	6G	1x	6G	863	818	50688
SerialLite III [ab]	120G	12 lanes	10.3125G	5600	6200	983040
Aurora 8B/10B [16]	12G	4 lanes	3G	3473	3319	75776
Aurora 64B/66B [16]	14G	1 lane	14G	1600	1600	37920
Aurora 64B/66B [16]	56G	4 lanes	14G	3500	3900	43172
1000Mb Ethernet MAC (external PHY)	1G	1 port RGMII	125Mx4 DDR	1184	1704	204976
1000base-X Ethernet MAC	1G	1 lane	1.25G	1805	2365	204976
10/100/1000Mb Ethernet MAC (ext. PHY)	1G	1 port RGMII	125Mx4 DDR	3155	3522	328064
10/100/1000Mb Ethernet MAC (ext. PHY)	1Gx12	12 port GMII	125Mx8 SDR	27360	29272	1479168
10Gb Ethernet MAC	10G	1 lanes	10.3125G	2001	3077	0
40Gb Ethernet MAC	40G	4 lanes	10.3125G	13600	23500	184320
100Gb Ethernet MAC	100G	10 lanes	10.3125G	45100	87700	573440
Interlaken 100G [a]	124G	12 lanes	10.3125G	18900	36800	778240
Interlaken 50G [a]	50G	8 lanes	6.25G	12200	26300	942080
Interlaken 20G (Stratix IV)	25G	4 lanes	6.25G	12229	16774	479232

[a] Figures not available for optional reliability extension [b] Provides insufficiently robust optional single bit error protection

TABLE I

PUBLISHED AREA OF STANDARD INTERCONNECT IP. DATA IS FOR STRATIX V DEVICES AND FOR ALTERA IP FROM [1] UNLESS OTHERWISE STATED. IN EACH CASE THE MINIMAL DESIGN HAS BEEN TAKEN – EXCLUDING PERFORMANCE COUNTERS AND OTHER OPTIONS.

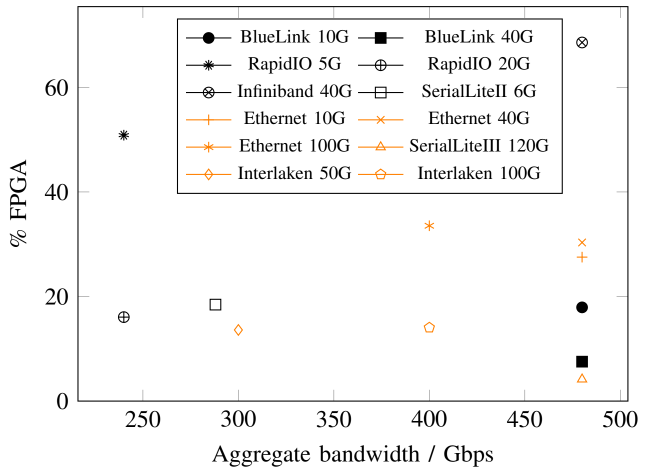

Fig. 5. Stratix V GX A7 logic utilization when instantiating each system as many times necessary to use the full transceiver resource. Protocols in black implement reliability, those in orange do not, area numbers from Tables I and II. Multi-lane systems can share area between multiple transceivers and hence have lower area overall, but have additional fitting and routing constraints that make them more difficult to use in practice.

System	LUTs	Registers	Memory bits
10G BlueLink reliability layer	1663	1277	2090
10G BlueLink link layer	179	413	960
10G BlueLink PHY	167	248	0
10G BlueLink total area	**2009**	**1938**	**3050**
40G BlueLink reliability layer	1965	1355	2090
40G BlueLink link layer	1127	1970	2736
40G BlueLink PHY	289	585	0
40G BlueLink total area	**3381**	**3910**	**4826**
10G Ethernet MAC	2986	3817	20972
10G Ethernet PHY	100	94	0
10G Ethernet total area	**3086**	**3911**	**20972**

TABLE II

AREA OF OUR IMPLEMENTATIONS OF BLUELINK AND ETHERNET ON STRATIX V FPGA. BLUELINK USES LESS OF ALL RESOURCES BUT PARTICULARLY MEMORY.

We put 16 DE4 boards together into a single Bluehive box (Figure 6), with the intention the system can scale to further boxes using eSATA cables. We are currently working on building enclosures for 150 FPGAs.

For a portable version of the system we designed a PCB to join three FPGA cards linked by their PCIe 8× connector (Figure 8) - this is able to additionally connect Stratix V FPGAs with 40 Gbps BlueLink bidirectional channels using groups of 4×10 Gbps lanes. SFP+ cables can also be used.

Each FPGA hosts two custom soft vector processors each driving a DDR2-800 memory channel. These compute neural state updates and generate synaptic messages. The messages are then routed via BlueLink to the other processors.

The system will successfully simulate two million neurons in near real-time. The application scales well – the limit on scaling is primarily compute bound, indicating that network bandwidth and latency have ceased to become a bottleneck.

IX. CONCLUSION

In this paper we have described how to build FPGA clusters at scale using commodity FPGA boards, high speed serial transceivers and commodity cabling. This resolves economic and physical problems, which leaves the decision of which

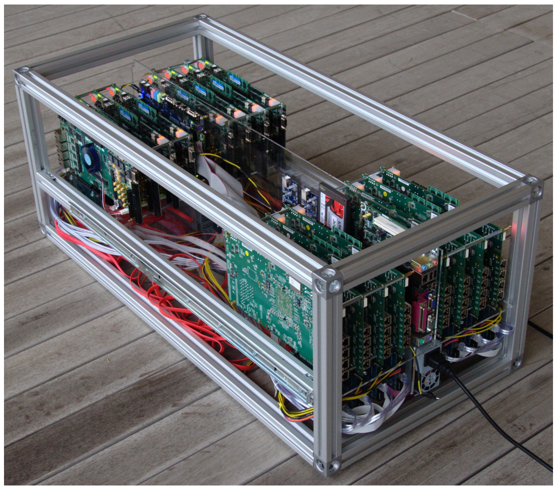
Fig. 6. Bluehive prototyping system

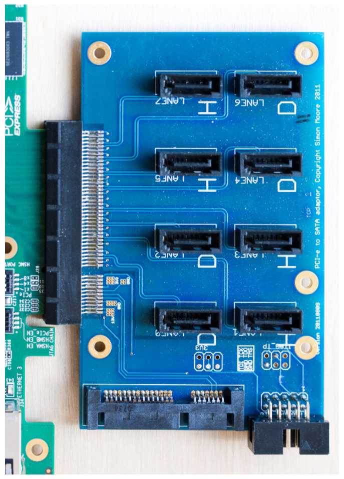
Fig. 7. PCIe to SATA breakout board

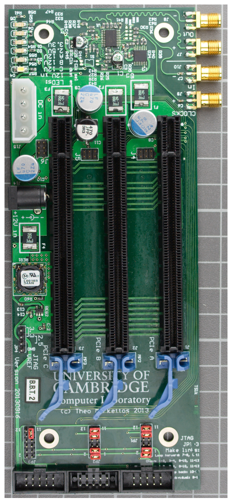
Fig. 8. PCB for BlueLink over PCIe connectors

protocol to use. The interconnect must be lightweight and flexible to maximise use of transceiver resources on the FPGA. It must also support reliable transmission of messages, because probabilities of error in a cluster are high and applications in hardware are not designed to handle packet error or loss.

We propose *custom communication*, by analogy with *custom computation*. A designer should consider their communication requirements at the same time as considering their compute requirements. A communication system should then be designed from the ground up to support the application.

IP for standard protocols is seductive. It gives the promise of a 'drop-in' communication system, a black box where the user need not be concerned with the internals. However, many FPGA-FPGA applications are different to those for which the protocols were designed. Standard IP brings with it a host of practical restrictions that make implemention an arduous task.

Using the example of BlueLink, a custom interconnect toolkit we designed for a specific application, we have shown how FPGA application requirements can differ significantly from standard networking. Ethernet, which is a natural choice for networking, imposes significant overhead and latency penalties for the small packets used in our FPGA application. It also takes more area and is lacking in reliable transmission.

We have also evaluated a selection of other IP. Either it does not support reliability, leaves little area for the application, has bandwidth limitations, or has other restrictions. Resolving these problems can involve additional layers or wrappers to meet the application's requirements: an example of custom communication. A custom approach does not preclude the use of standard IP where it has the necessary properties; it may be one part of a multi-layer stack. Such a stack should be designed from the beginning. The designer should not reach for the standard IP as the panacea for their needs. As the number of transceiver links on modern FPGAs continues to multiply, we suggest a custom communication approach will be increasingly necessary to make best use of this growing communication resource. We expect this trend to continue for some time to come.

X. Acknowledgements

This work is part of the MRC2 Project that is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. The views, opinions, and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense. We acknowledge additional support from the UK research council, EPSRC, grant EP/G015783/1.

References

[1] Altera Corporation. Interface protocols. http://www.altera.co.uk/products/ip/iup/ipm-index.jsp.

[2] T. Bunker and S. Swanson. Latency-optimized networks for clustering FPGAs. In *Field-Programmable Custom Computing Machines (FCCM) 2013*, pages 129–136.

[3] Altera Corporation. Stratix V GX FPGA development kit. http://www.altera.co.uk/products/devkits/altera/kit-sv-gx-host.html.

[4] DigiKey Corporation. Embedded - FPGAs. http://www.digikey.com/product-search/en/integrated-circuits-ics/embedded-fpgas-field-programmable-gate-array/2556262.

[5] DINI. Big FPGA boards for code verification and ASIC prototyping. http://www.dinigroup.com/new/why.php.

[6] DINI. TCP Offload Engine. http://www.dinigroup.com/new/TOE.php.

[7] Intilop Corporation. 10 G bit TCP Offload Engine + MAC + Host_IF (SXTOE). http://www.intilop.com/ipcores.php.

[8] Y. Kono, K. Sano, and S. Yamamoto. Scalability analysis of tightly-coupled FPGA-cluster for lattice Boltzmann computation. In *Field Programmable Logic and Applications (FPL) 2012*, pages 120–127.

[9] Kouadri-Mostefaoui et al. Large scale on-chip networks : An accurate multi-FPGA emulation platform. In *Digital System Design Architectures, Methods and Tools, 2008. 11th EUROMICRO Conference on*, pages 3–9.

[10] Chen Liang et al. An FPGA-cluster-accelerated match engine for content-based image retrieval. In *Field-Programmable Technology (FPT) 2013*, pages 422–425.

[11] O. Mencer et al. Cube: A 512-FPGA cluster. In *Southern Conference on Programmable Logic (SPL) 2009*, pages 51–57.

[12] Simon W Moore, Paul J Fox, Steven J T Marsh, A Theodore Markettos, and Alan Mujumdar. Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation. In *Proceedings of FCCM 2012*, pages 133–140.

[13] MorethanIP. 1/2/4/8Gbps Fibre Channel Transport Core. http://www.morethanip.com/products_web/07_8gigabit_fibre/1_2_4_8_fc_fs_transport_pb_alt_v1.1.pdf.

[14] A.B. Nejad et al. An FPGA bridge preserving traffic quality of service for on-chip network-based systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6.

[15] Polybus Systems Corporation. InfiniBand cores. http://www.polybus.com/ib_link_layer_website/ib_cores_brochure_alt.pdf.

[16] ReFLEX CES. Focus on Aurora-like 8b/10b at 3Gbps. http://reflexces.com/focus.html?s=aurora-like-8b10b-3gbps.

[17] University of Cambridge. PCIe to SATA breakout board. http://www.cl.cam.ac.uk/research/comparch/opensource/pcie-sata/.