Reliably Prototyping Large SoCs Using FPGA Clusters

Paul J Fox, A Theodore Markettos and Simon W Moore
Computer Laboratory
University of Cambridge
Cambridge, United Kingdom
{*paul.fox, theo.markettos, simon.moore*}*@cl.cam.ac.uk*

***Abstract*—Prototyping large SoCs (Systems on Chip) using multiple FPGAs introduces a risk of errors on inter-FPGA links. This raises the question of how we can prove the correctness of a SoC prototyped using multiple FPGAs. We propose using high-speed serial interconnect between FPGAs, with a transparent error detection and correction protocol working on a link-by-link basis. Our inter-FPGA interconnect has an interface that resembles that of a network-on-chip, providing a consistent interface to a prototype SoC and masking the difference between on-chip and off-chip interconnect. Low-latency communication and low area usage are favoured at the expense of a little bandwidth inefficiency, a trade-off we believe is appropriate given the high bandwidth of inter-FPGA links.**

***Keywords*-SoC; prototyping; reliability; interconnect; FPGA; communication;**

I. Introduction

Prototyping a large SoC using FPGAs needs multiple devices as there is not enough space in a single FPGA. These FPGAs must be connected, both physically and logically, to bridge a partitioned prototype SoC. High bandwidth, low latency interconnect allows a tight coupling across partitions. For a trustworthy prototype we require this interconnect to not introduce errors into the system under test.

With high data rates come bit errors. All high-speed serial communication standards we have encountered have methods for detecting bit errors on physical links. Therefore we need to both detect and correct errors on inter-FPGA links so that they cannot silently corrupt the operation of a multi-FPGA SoC prototyping system.

The probability of bit errors on inter-FPGA links can be reduced by placing the FPGAs on a single large PCB, but these are very expensive and lack flexibility (e.g. to change interconnect topology). In addition, the possibility of errors is still not eliminated and so the question of confidence in results remains. To provide a lower cost, more flexible, SoC prototyping system we use multiple commodity single-FPGA PCBs linked by high-speed serial interconnect. While the probability of bit errors on these links is higher than on a large PCB, by implementing a transparent error detection and correction protocol we provide confidence in results produced by our system.

To simplify SoC prototyping we propose a network-on-chip style interconnect [1] that extends between FPGAs to provide a consistent interface to a prototype SoC and mask the difference between on-chip and off-chip interconnect. This means that our interconnect must have a low latency, and so error correction techniques used in local-area networks such as TCP are unsuitable as they favour bandwidth at the expense of latency, and also have high buffering requirements and hence large area. In contrast we sacrifice some bandwidth to produce an error detection and correction protocol with reduced latency, a trade-off we believe is appropriate given the high bandwidth of inter-FPGA links and the demands of SoC prototyping.

II. Requirements

The interconnect requirements for multi-FPGA SoC prototyping are quite distinct from those of conventional networking (e.g. for PC clusters):

1) Reliability: We need an interconnect that is *reliable* in the face of bit errors so that we can have high confidence in the correctness of a prototype SoC. Due to our need to interface to a prototype SoC at a hardware level, we cannot leave error handling to higher layers (as in IP networking), so we must handle it in hardware next to the physical links.

2) High bandwidth and ultra-low latency: We would like to minimise the effort involved in partitioning a SoC over multiple FPGAs. If bandwidth is high and latency is kept low, multiple FPGAs can be treated as if they were one large chip. This is particularly helpful if the SoC has no obvious partition boundaries. For those parts that must run in lock-step, dilation of the simulated SoC clock enables communication to happen within a single cycle. The length of that cycle and so SoC performance is thus very much dependent on the link latency.

3) Minimal overhead: The interconnect must have low overheads because they increase latency and reduce throughput. Overhead scales linearly with hop count, so is especially critical for multi-hop traffic.

4) Frequent communication: Communication can be expected to happen frequently between partitions in a prototype SoC, particularly as these partitions are likely to be essentially arbitrary, and so there will be as much communication off-FPGA as there is within a FPGA.

5) Short messages: The combination of the requirements for low latency and frequent communication leads to a requirement for the messages used for off-FPGA communication to be short. Sending large messages which may

978-1-4799-5810-8/14/$31.00

be mostly empty would increase the latency due to the interconnect being busy.

6) Simple interfacing: We wish to mask the difference between on-chip and off-chip interconnect to provide a consistent interface to a prototype SoC and simplify partitioning over multiple FPGAs.

7) Scalability: The *raison d'etre* of multi-FPGA prototyping is that one FPGA is not enough. Therefore an interconnect should not impose its own size limits – we should be able to scale to hundreds of FPGAs.

III. Existing approaches

Many previous approaches to SoC prototyping using FPGAs have used a single FPGA [2], [3]. Multiple FPGAs connected using a high-speed serial interconnect have been used [4], [5], but in both cases it is assumed that inter-FPGA links are error free. In our experience this is an incorrect assumption, particularly if we are to have confidence in the correctness of a prototype SoC.

If we wish to build on the work in [4] while also providing error detection and correction we will need to implement an appropriate error detection and correction protocol over a high-speed serial interconnect. There are many protocols available, in addition to the option of creating a custom protocol, each with advantages and disadvantages for SoC prototyping.

Many cluster systems, and in particular PC clusters, use Internet Protocol over physical technologies such as Ethernet. This provides off-the-shelf switches, cables, routers, media converters and FPGA IP. The disadvantage of Ethernet is its focus on large packet sizes, which has an impact on latency and throughput if shorter data payloads are required. This is noticeable in the multi-FPGA SoC prototyping system presented in [5], which uses 1G Ethernet and exhibits inter-FPGA latency of the order of 10 µs in many cases, and sometimes even higher.

Interconnect protocols designed for motherboard-level communication include HyperTransport and PCIe. HyperTransport's minimum packet is 4 bytes of payload with 8 bytes of overhead, while PCIe has 4 bytes of payload for 20 bytes of overhead [6]. HyperTransport is primarily a motherboard-only protocol since it uses source-synchronous clocking [7]. PCIe uses high speed serial links so can travel along cabling (either directly or encapsulated in the form of Thunderbolt [8]). It consists of a 'root complex' interfacing a CPU to a PCIe switch which joins PCIe peripherals. Laid on top of this is a complex protocol intended to emulate all the features of traditional PCI – the PCIe version 3.0 specification runs to 860 pages [9]. PCIe architecture is primarily hierarchical - major usage is communication between CPU/memory and peripherals. Peer-to-peer communication is possible, but complex and focused on memory-mapped I/O [10].

Infiniband is a serial point-to-point interconnect often used by supercomputers. In these circumstances it is often used as a conduit for IP, but it represents a general-purpose fabric. The Infiniband Trade Association themselves quote an end-to-end latency of 1 µs [11].

RapidIO [12] is perhaps the protocol most similar in philosophy to the requirements we identified in the previous section. It has reliability, low-latency (1 µs end-to-end), and scalability [13]. However, it also has high overhead (12-16 bytes per packet) so is not ideal for creating a transparent interconnect for SoC prototyping.

Aurora is a protocol that is restricted to Xilinx FPGAs. It was used for SoC prototyping in [4] at a data rate of 3.125 Gbps. However inter-FPGA links were assumed to be error-free and so no error detection or correction was implemented. Another implementation of a multi-FPGA system using Aurora found that errors restricted usable data rate to 1.95 Gbps per lane [14].

AIREN [15] blends on-chip and off-chip networks using Aurora. On-chip a full crossbar with dimension-ordered routing is used. Off-chip packets are source-routed across the fabric, and are then presented to the target FPGA using remote DMA. It does not appear to implement error detection or correction.

IV. Reliable FPGA interconnect

Our reliable FPGA interconnect follows the requirements identified in Section II. Reliability is achieved at a per-link level using one *reliable transceiver* at each end of each inter-FPGA link, forming a *reliable link*. These reliable links are then combined with a *routing and switching system* to allow our interconnect to span multiple FPGAs. All of the components of the interconnect (other than the lowest level *transceiver interface*) have been implemented in Bluespec SystemVerilog. Our current implementation targets Altera FPGAs, though we believe that it is also compatible with FPGAs from Xilinx which have similar SERDES hard blocks.

A. High-level interface

A high-level view of the interconnect is shown in Figure 1. It is implemented as an Altera Qsys subsystem, which has four internal communication ports and a number of interfaces to inter-FPGA links, which each include a transmit and receive differential pair and a reference clock input.

Once the interconnect block is instantiated in a higher-level Qsys system, inter-FPGA link interfaces are exported and connected to appropriate pins. The prototype SoC is connected to the interconnect using bridges. These convert the SoCs interfaces to Altera's Avalon Streaming interfaces to allow connection to the interconnect using Qsys. Abstractly a streaming interfaces can be thought of as a buffered channel.

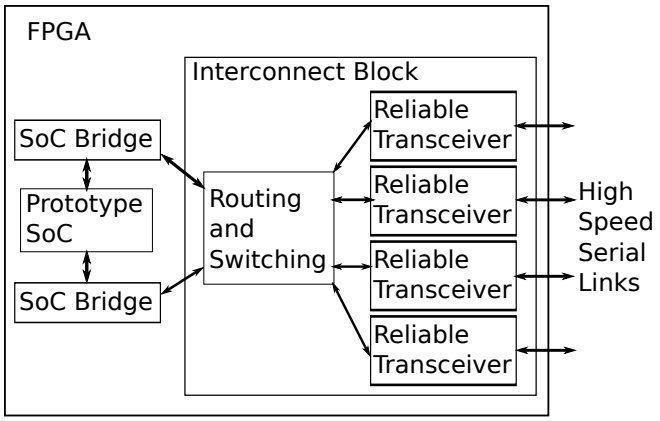

Figure 1. High-level view of FPGA interconnect

SoC bridges communicate using the inter-FPGA interconnect by sending and receiving *flits*, which contain a 64-bit payload and control bits to control routing and switching and to allow multiple flits to be formed into packets. The structure of a flit is described in more detail in Section IV-C

B. System architecture

The reliable FPGA interconnect is made up of a number of *reliable transceivers* (described in Section IV-D), each of which independently provides a reliable communication channel with FIFO semantics between two FPGAs, combined with a *routing and switching system* (described in Section IV-E) to allow SoC bridges to communicate with any other bridge in the multi-FPGA prototyping system, hence linking the partitioned prototype SoC.

SoC bridges communicate by specifying either a specific reliable link to an adjacent FPGAs (*direct link mode*) or alternatively by specifying the identity of a target FPGA, in which case hop-based routing is used to select appropriate links to reach the target FPGA, via intermediate FPGAs as needed (*routed mode*). In both cases the target bridge on the target FPGA is also specified.

Direct link mode is most useful for prototyping SoCs that can be partitioned by simple tiling as it introduces less latency than routed mode as it is not necessary to calculate the number of hops required to reach the target FPGA. It is also used when setting up the prototyping system, as it allows the identity of the FPGA at the other end of each link to be established and hence for routing tables to be constructed (see Section IV-E5).

Routed mode allows SoC bridges to communicate with bridges on other FPGAs in the prototyping system without needing to be aware of its topology and without the involvement of the SoC or bridges on any intermediate FPGAs, allowing the partitioning of more complex prototype SoCs.

C. Communication data structures

The fundamental unit of communication used on the interconnect is a *flit*. 76 bit flits are used by our SoC bridges.

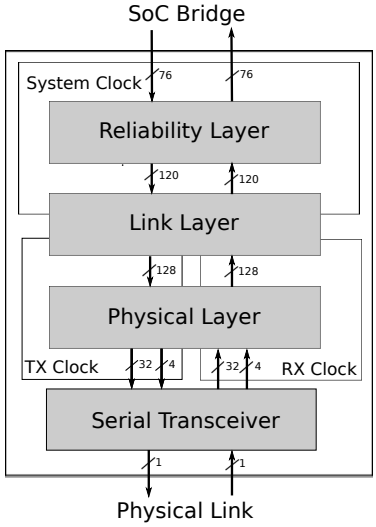

Figure 2. Diagram of the reliable transceiver, showing layers and clock domains

These are extended to 128 bits in the reliable transceiver with the information necessary to perform reliable transmission.

The fields visible to a bridge are:

Payload	Control	Start Packet	End Packet
64 bits	10 bits	1 bit	1 bit

The fields added by the reliable transceiver are:

CRC	Seq #	Ack #	Start of Day	Header
32 bits	$4+1$ bits	$4+1$ bits	2 bits	8 bits

Bridges are responsible for setting the *control* field appropriately to specify the target FPGA and SoC bridge (See Section IV-E2). Flits can be formed into packets by setting the start of packet flag set on the first flit and the end of packet flag on the last flit. Bridges are responsible for setting these flags appropriately, and for interpreting them to reconstruct packets.

D. Reliable transceiver

The *reliable transceiver* provides reliable communication between FPGAs with FIFO semantics, including ordering and back-pressure. It is made up of a number of layers, which are shown in Figure 2. In a similar manner to an Ethernet stack, each layer handles communication with decreasing levels of abstraction as it gets closer to the high-speed serial transceiver. Each of the layers that make up the reliable transceiver will now be discussed.

1) Reliability layer: The layout of this layer is shown in Figure 3. It provides a reliable communication channel with FIFO semantics by appending a CRC and sequence number to each flit, which are validated by this layer in the receiving reliable transceiver. An acknowledgement number may also appended to a flit to acknowledge correct receipt of a flits with that sequence number. If the receiver receives

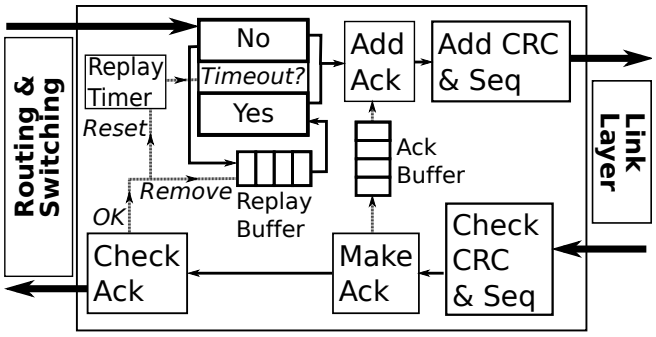

Figure 3. Diagram of reliability layer

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 speculative transmission [16]. Transmitted flits that have not yet been acknowledged being stored in a replay buffer. If a flit is not acknowledged after a timeout (either because the receiver detected an error or because an acknowledgement was lost), the first flit in the replay buffer is sent continuously until it is acknowledged, and the same for the other flits in the buffer, after which new flits are accepted from the input. We use an 8-flit retransmission window, and so the replay buffer only needs to hold 8×64 bit flits to store a whole window, which significantly reduces FPGA area requirements compared to protocols with larger flits and longer windows.

Back-pressure is achieved by sending acknowledgements with an extra flag to indicate that no more flits can be accepted. This prevents any further flits being transmitted, and so ultimately leads to the reliable transceiver's input FIFO becoming full. The start of day bits are used to ensure that a link resets correctly. The output to the *link layer* is a stream of 120 bit flits.

2) Link layer: This layer performs clock crossing between the system clock domain and the transmit and receive clock domains used by the *physical layer* and *serial transceiver*. It also serialises flits into 32 bit words. An 8 bit header is appended to each 120-bit flit from the reliability layer. These 128-bit flits are then serialised into $4 \times \{32$ bit words, 4 bit k symbol indicator$\}$ tuples for the *physical layer*. The header is used by this layer to identify the start of a 128 bit flit when de-serialising 32 bit words into 128 bit flits, and by the physical layer to perform word alignment.

3) Physical layer: This layer performs word alignment on the bytes in the 32 bit input received from the *serial transceiver* using the 8 bit flit headers as an alignment pattern. It also provides the *serial transceiver* with a continuous stream to transmit. If no valid input is received from the link layer, the symbol required for the transceiver to perform byte alignment is sent. This layer is clocked by the transmit and receive clocks provided by the serial transceiver.

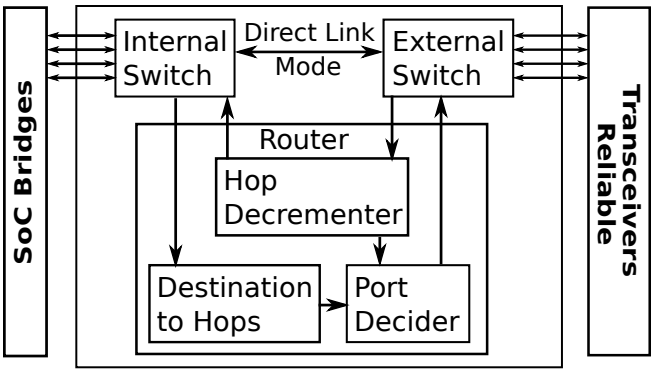

Figure 4. Diagram of the routing and switching system

4) Serial transceiver: A FPGA-specific transceiver block provided by the FPGA manufacturer. On an Altera Stratix IV this is an ALTGX megafunction, while on Stratix V it is a Custom PHY megafunction. While this has not been tested, it would appear that a Xilinx GTP transceiver could also be used for compatibility with Xilinx FPGAs. The transceiver block is configured to use 8b10b coding and byte alignment of received bits using an alignment pattern, an 8b10b k (or comma) symbol, which is the pattern sent by the physical layer when it has no valid data to transmit.

The transceiver block generates a transmit clock and recovers a receive clock from the incoming serial bit stream. Higher-layer logic needs to perform clock crossing between these clock domains and that used by the rest of the system.

We use a simple Verilog wrapper around the FPGA manufacturer's transceiver block to provide a consistent interface to higher layers in our stack. The transceiver block and this wrapper are the only parts of our system that are specific to a particular FPGA.

E. Routing and switching

The *routing and switching system* is shown in Figure 4. It connects SoC bridges on a FPGA with other bridges on other FPGAs. With each inter-FPGA link handled by a *reliable transceiver*, which provides a reliable communication channel with FIFO semantics, routing and switching flits requires only directing flits to an appropriate link and (for routed flits) re-writing a flit's control bits.

Hop-based routing is used to reduce latency at intermediate FPGAs between a source and target FPGA. Each intermediate FPGA only needs to decrement the hop count on an incoming flit and send it on an appropriate link to get it closer to its destination.

1) FPGA topology: The multi-FPGA prototyping system is assumed to be logically arranged as a 3D torus, with FPGAs each assigned a unique, sequential identifier. We assume that the FPGAs are fully connected in the x dimension, but potentially not in the y and z dimensions. Each FPGA has a unique, sequentially assigned identifier and knows the size of the FPGA system in each dimension.

2) Format of flit control bits: When used with the routing and switching system, the 10 control bits in a *flit* are subdivided into:

Direct Link?	Target FPGA / Link	Target Bridge
1 bit	7 bits	2 bits

The direct link flag is used to choose between either *direct link mode* or *routed mode*. Direct link mode sends a flit to the FPGA at the other end of the off-FPGA link specified by the *target FPGA or link* field. Routed mode sends a flit to the FPGA specified by this field, via intermediate FPGAs as necessary. These modes can be selected on a flit-by-flit basis, though the mode should be kept consistent within a packet.

3) Operation in direct link mode: In direct link mode the *internal switch* accepts an incoming flit, multiplexes it with flits from other applications and then passes it directly to the *external switch*. This sends the flit on the specified inter-FPGA link. The flit is received by the external switch on the target FPGA and multiplexed with flits from other links. It is then passed directly to the internal switch and then to the specified application. The *target FPGA or link* control field is re-written to indicate the link that the flit arrived on, which is useful when constructing routing tables.

4) Operation in routed mode: In routed mode the internal switch and external switch both pass incoming flits to the *router*. Incoming flits from applications are checked to see if they target this FPGA (i.e. the source and destination FPGA are the same). If so they are sent straight back to the internal switch, which delivers them to the target application. Flits that target other FPGAs are passed to the *destination to hops converter*. This contains a routing table that maps each target FPGA to a number of hops in the x, y and z dimensions. The control bits are re-written to replace the *target FPGA field* with these hops, 3 bits for the x and y dimensions and 2 for z. Flits are then passed to the *port decider*. This uses a routing table to determine which link to use to get one hop closer to the target FPGA. z hops are prioritised, then y, then x, constant with our assumptions on the system topology. The port decider passes flits to the external switch and hence an inter-FPGA link.

When the router receives a flit, it is passed to the *hop decrementer*, which decrements a hop in the dimension corresponding to the link that the flit arrived on, using another routing table. Any flit which has no hops remaining in any dimension has reached its target FPGA, and is passed to the internal switch to be delivered to the target application. Flits with remaining hops are passed to the port decider to continue their journey to their target FPGA.

5) Routing table setup: Before the routing and switching system can be used in routed mode, its routing tables must be populated. This is currently done by software running on a NIOS II soft core, though in principle a hardware state machine could be used. Given the dimensions of the system and an FPGA's identifier, for each link it must determine which FPGA it connects to, its dimension and direction of travel in the torus. In cases where more than one link has the same dimension and number of hops, we determine which link appears to be the most reliable and hence has the lowest latency.

The process begins by each FPGA sending a *probe flit* on every link. These flits are sent in direct link mode, and contain the identifiers of the FPGA and the link in the payload. Each FPGA then polls all of its links to receive flits, and waits until it has received at least 1 flit in every dimension and direction. This ensures that all FPGAs are programmed and have started the setup process before any inferences are made about the reliability of links. Once probe flits are received, each FPGA can determine which FPGA each link connects to and its dimension and direction of travel in the torus.

To determine which link is most reliable in each dimension and direction, we would ideally count CRC failures. However, CRCs are checked at the receiving end of each link, and as we have not finished routing table setup there is no way for sending FPGAs to access this information. Instead we make use of the observation that less reliable links have lower throughput as they need to retransmit any failing flits. Using this observation, each FPGA sends a further 32 probe flits on each link. The link in each dimension and direction that finishes sending its probe flits first is assumed to be the most reliable, and is entered into the routing table as the link to be used for that dimension and direction. This completes routing table setup.

V. Platforms

Our initial prototyping platform was the Bluehive [17], shown in Figure 5. Bluehive is a custom FPGA cluster containing 16 Altera Stratix IV GX 230 FPGAs. These are mounted on commercial FPGA evaluation boards – the DE4 from Terasic – which reduces cost through economies of scale. Each DE4 has 4 SATA ports and an 8× PCI Express connector which are directly connected to transceivers.

The SATA ports are connected to external ports of our reliable FPGA interconnect and can be used to directly connect boards using ordinary SATA cables. Instead of using a conventional PCI Express backplane, which involves implementing the PCIe protocol and using a PCIe bridge chip which adds latency, we connect each lane in the PCI Express port directly to an external port of our interconnect.

To create a *pluggable topology* we use a small 4-layer custom PCB to convert the PCIe Express interface into 8 SATA links, at a physical layer only. This board is shown in Figure 6. Its design is freely available at [18].

Bluehive has virtually no bit errors with links running at 3 Gbps. At 6 Gbps some links occasionally exhibit bit error rates of up to 10^{-2}, which may be a result of faulty cabling, connectors, soldering on breakout boards or similar.

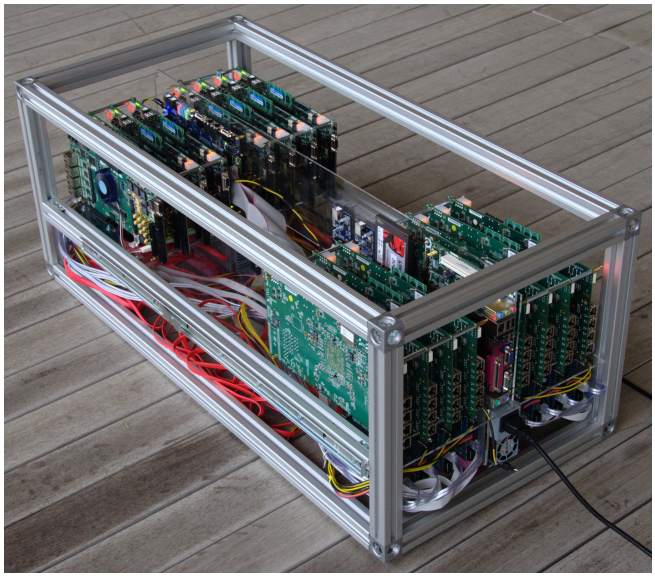

Figure 5. Bluehive prototyping system

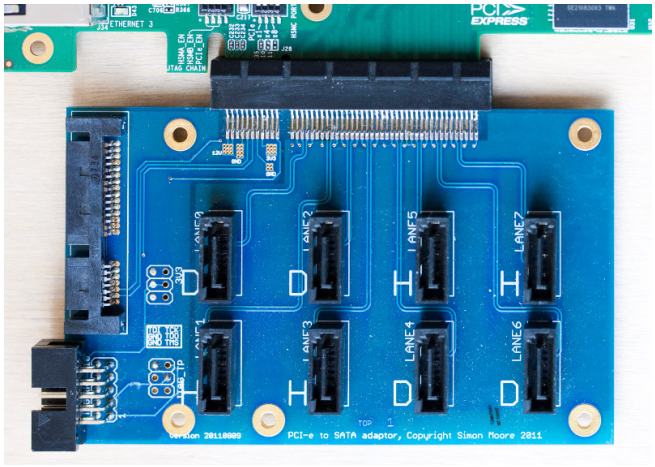

Figure 6. PCIe to SATA breakout board

Newer FPGA evaluation boards expose some of their transceiver interfaces as SFP+ cages. These can be used with a variety of physical interconnects, including direct attach cabling, optical transceivers and Ethernet.

We have created a small prototyping system using two Altera Stratix V GX EA7 FPGAs on Terasic DE5 evaluation boards, shown in Figure 7. The reliable FPGA interconnect implementation is identical to that used with the Stratix IV FPGA apart from use of the Stratix V specific transceiver.

VI. Example Applications

We have implemented two applications on our multi-FPGA systems using our interconnect. In [17] we presented a full-custom system for neural network simulation. The whole system was implemented in Bluespec SystemVerilog (BSV), which was interfaced to our interconnect via simple FIFO bridges.

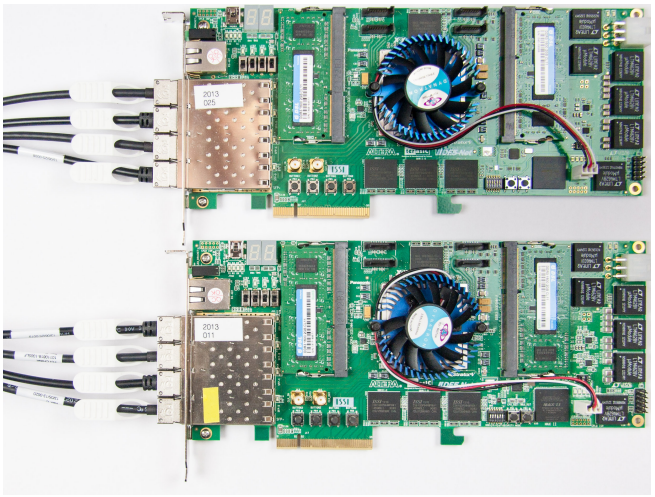

Figure 7. DE5 prototyping system with SFP+ direct attach cabling

We have also implemented a vector processing system for the same neural simulation task [19]. Altera NIOS II processors facilitate control the execution of our BlueVec vector processors and communicate using our interconnect via bridges that present a memory-mapped interface to our interconnect. Adapting a single core BlueVec system running on a single FPGA into a multi-core system spread over 16 FPGAs took one day of effort for the BlueVec developer.

VII. Evaluation

To evaluate our interconnect, we consider the effect of errors on the effective data rate of a link with a given bit error rate, the latency of an inter-FPGA link and finally FPGA resource requirements.

A. Measurement infrastructure

To measure throughput and latency, we built a BSV hardware block that sends and receives streams of flits, resulting in a stream of latency or throughput measurements that is either stored in a FIFO and later retrieved by a host PC (on hardware) or saved to a file (in simulation).

The effect of bit errors on throughout was modelled using Bluesim, the cycle-accurate BSV simulator. Two complete FPGA network-off-chip systems were instantiated (representing two FPGA boards) and connected using FIFOs to emulate physical links. While the physical links themselves were not modelled, the clock frequencies of the system, transmit and receive clocks (see Figure 2) were tuned such that in a case with no bit errors 1 flit could be sent or received every 10 clock cycles, matching the rate measured on the Bluehive with the physical links running at 3.125 Gbps.

Bit errors are approximated by forcing the CRC check in the receiver to fail for every nth flit. For example, a bit error rate of 2^{-14} is equivalent to one error in every 16384 bits. If errors are evenly distributed, this is equivalent to 1 error in 128 flits, so $n = 128$.

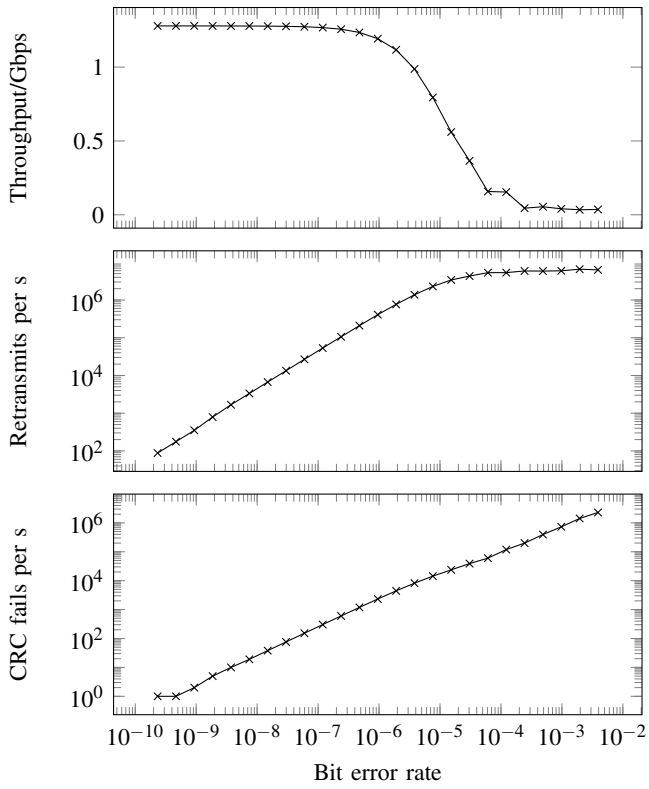

Figure 8. Performance for varying bit error rates (3.125 Gbps link rate)

B. Throughput and Effective data rate

The effect of bit errors on throughput is shown in Figure 8. Effective data rate is derived from throughput by assuming that each flit that is successfully transmitted carries 64 bits of useful data. Of particular note is the effective data rate of 1.2 Gbps with a low bit error rate of 10^{-10}. Given that 64 bits of payload are carried in a 128 bit flit and that 8b10b encoding gives an overhead of $10/8 = 1.25$, this gives an actual data rate on the physical link of 3.0 Gbps, which closely matches the intended physical link data rate of 3.125 Gbps.

It is notable that the data rate for high error rates (10^{-3} or higher) is around 40 Mbps. While this is clearly significantly less than the physical link rate of 3.125 Gbps, this will still provide a more than adequate communication link for setting up and monitoring a multi-FPGA cluster, for example to indicate to the operator that some physical links are showing high error rates and should be investigated.

C. Latency

The latency of a single inter-FPGA link was measured using two Stratix IV GX 230 FPGAs with line rates of 3.125 Gbps and 6.250 Gbps (connected using the PCIe–SATA breakout board and SATA cables) and also two Stratix V GX EA7 FPGAs with a line rate of 10.0 Gbps (connected using SFP+ direct attach cables). Figure 9 shows two types of measurement for these line rates: *unloaded latency* uses a single flit at a time on an otherwise unused link and *fully loaded latency* sends as many flits as possible, as for the throughout test.

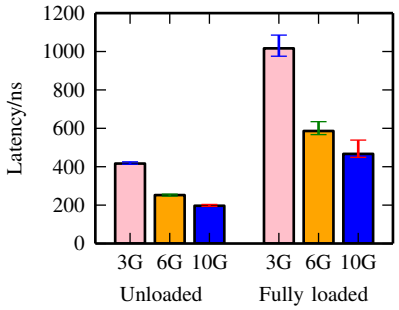

Figure 9. End-to-end latency at different link rates

System	Combinational ALUT	Logic Registers	M9K Memory
Reliable Transceiver	1533 (0.8%)	1936 (1.1%)	17 (1.4%)
RapidIO Core (1×)	7100 (3.9%)	7600 (4.2%)	51 (4.1%)
4× Reliable Transceiver	6132 (3.4%)	7744 (4.2%)	68 (5.5%)
RapidIO Core (4×)	9000 (5.0%)	10300 (5.7%)	51 (4.1%)
Network-off-Chip Block	28590 (16%)	25248 (14%)	217 (18%)
inc NIOS & DDR2 ctl	50667 (28%)	76359 (42%)	337 (27%)

Table I
FPGA RESOURCE REQUIREMENTS. FIGURES IN BRACKETS ARE % OF RESOURCES AVAILABLE ON AN ALTERA STRATIX IV GX 230

The difference between the unloaded and fully loaded latencies is primarily a result of 1 to 4 serialisation in the link layer, which becomes pronounced when transmit buffers become full. Fully loaded latency is around 1 µs at 3.125 Gbps, 600 ns at 6.25 Gbps and 450 ns at 10.0 Gbps, which compares with 1 µs quoted for RapidIO [13] (unclear if unloaded or fully loaded latency) at 10 Gbps line rate.

D. FPGA resources

Table I shows the FPGA resource requirements of our network-off-chip. We measured a single reliable transceiver, a network-off-chip block of 12 reliable transceivers (using 12 lanes from a DE4 board), and a full design with network-off-chip, 2× Altera NIOS II processors and 2× DDR2 memory controllers plus interfacing logic. The resource requirements of 1-lane and 4-lane Altera RapidIO cores (taken from [20]) are also shown for comparison.

It is clear that our reliable transceiver uses significantly less resources than the Altera RapidIO core, with 25% of the logic usage and 33% of the memory usage. The majority of the memory usage of the RapidIO core results from its minimum 8KB transmit buffer and 4KB receive buffer. This compares with the 8-flit and 16-flit transmit and receive buffers in the reliable transceiver, which use 574 bits and 1230 block memory bits respectively, both mapping to three M9K memory blocks. The remainder of the memory block usage comes from the clock crossing FIFOs in the link layer. The low buffer sizes that make this lower memory block utilisation possible are enabled by out short flit size and low end-to-end latency.

The 4-lane RapidIO core and 4 reliable transceivers are not directly comparable as the former has a single set of buffers and stripes each frame over 4 lanes while the latter are 4 independent channels. However the logic usage of 4 reliable transceivers is still lower than that of the 4-lane RapidIO core. Our higher memory block usage indicates that using 4 serial links per reliable transceiver will lead to promising future work as memory usage will be reduced. It could also eliminate the 1-to-4 serialisation in the link layer, which would significantly reduce latency.

Finally the resource usage of a complete prototyping design shows that use of our interconnect leaves ample FPGA resources for the prototype SoC after communication and memory interfaces have been instantiated.

VIII. Conclusions

Our interconnect for multi-FPGA SoC prototyping systems has low latency and is designed to scale to hundreds of FPGAs. Combined with some custom physical interconnect, it allows us to create multi-FPGA prototyping systems using commodity FPGA evaluation boards.

Errors introduced by inter-FPGA links are transparently detected and corrected, providing high confidence in the correctness of a prototype SoC. Low latency and short data units allow us to use much less buffering than comparable technologies such as RapidIO, and result in significantly lower latency and FPGA resource utilisation. This is at the expense of a little bandwidth inefficiency, a trade-off we believe is appropriate given the high bandwidth of inter-FPGA links.

The combination of transparent error correction, network-on-chip semantics, low latency and low resource usage simplifies partitioning of prototype SoCs over multiple FPGAs.

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. We would also like to thank Andrew Moore for his advice on high-speed communication protocols.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip inteconnection networks," in *Design Automation, Proceedings of the 38th Conference on*, 2001, pp. 684 – 689.

[2] D. Wang, N. Jerger, and J. Steffan, "DART: A programmable architecture for NoC simulation on FPGAs," in *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, May 2011, pp. 145–152.

[3] P. Wolkotte, P. Hölzenspies, and G. J. M. Smit, "Fast, accurate and detailed NoC simulations," in *Networks-on-Chip (NOCS) First International Symposium on*, May 2007, pp. 323–332.

[4] Kouadri-Mostefaoui, Abdellah-Medjadji, B. Senouci, and F. Petrot, "Large scale on-chip networks : An accurate multi-FPGA emulation platform," in *Digital System Design Architectures, Methods and Tools*, 2008, pp. 3–9.

[5] A. Nejad, M. Martinez, and K. Goossens, "An FPGA bridge preserving traffic quality of service for on-chip network-based systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pp. 1–6.

[6] B. Holden. (2006, November) Latency comparison between HyperTransport™ and PCI-Express™ in communications systems. [Online]. Available: http://www.hypertransport.org/docs/wp/Low_Latency_Final.pdf

[7] M. Cavalli. (2007, June) Advanced interconnect standards blend serial and parallel techniques for best performance and scalability. [Online]. Available: http://www.hypertransport.org/docs/wp/Serial_vs_Parallel.pdf

[8] Intel Corporation. (2012) Thunderbolt technology brief. [Online]. Available: http://www.intel.com/content/dam/doc/technology-brief/thunderbolt-technology-brief.pdf

[9] PCI-SIG, "PCI Express® base specification revision 3.0," November 2010.

[10] K. Kong. (2007) PCIe® as a multiprocessor system interconnect. [Online]. Available: http://www.pcisig.com/developers/main/training_materials/get_document?doc_id=8ac11bd327fad024de528e74830b3e6d8b220485

[11] Infiniband TA, "Advantages of Infiniband." [Online]. Available: http://www.infinibandta.org/content/pages.php?pg=about_us_advantages

[12] RapidIO Trade Association. RapidIO interconnect specification 3.0. [Online]. Available: http://www.rapidio.org/specs/current/2013-oct23.zip

[13] D. Paul, "Low latency servers with RapidIO," in *Open Server Summit*, October 2013. [Online]. Available: http://www.serverdesignsummit.com/English/Collaterals/Proceedings/2013/20131024_C201_Paul.pdf

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

[15] A. Schmidt, W. Kritikos, R. Sharma, and R. Sass, "AIREN: A novel integration of on-chip and off-chip FPGA networks," in *Field-Programmable Custom Computing Machines (FCCM)*, 2009, pp. 271–274.

[16] P. Watts, S. Moore, and A. Moore, "Energy implications of photonic networks with speculative transmission," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 4, no. 6, pp. 503–513, June 2012.

[17] S. W. Moore, P. J. Fox, S. J. T. Marsh, A. T. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 133–140.

[18] "PCIe to SATA breakout board." [Online]. Available: http://www.cl.cam.ac.uk/research/comparch/opensource/pcie-sata/

[19] M. Naylor, P. Fox, A. Markettos, and S. Moore, "Managing the FPGA memory wall: Custom computing or vector processing?" in *Field Programmable Logic and Applications (FPL)*, Sept 2013, pp. 1–6.

[20] RapidIO MegaCore function user guide. [Online]. Available: http://www.altera.co.uk/literature/ug/ug_rapidio.pdf