Design and implementation of the Quarc Network on-Chip

M. Moadeli[1], P. P. Maji[2], W. Vanderbauwhede[1]
1: Department of Computing Science
University of Glasgow
Glasgow, UK
Email: {mahmoudm, wim}@dcs.gla.ac.uk
2 : Institute for System Level Integration
Livingston, UK
Email: partha.maji@sli-institute.ac.uk

Abstract

Networks-on-Chip (NoC) have emerged as alternative to buses to provide a packet-switched communication medium for modular development of large Systems-on-Chip. However, to successfully replace its predecessor, the NoC has to be able to efficiently exchange all types of traffic including collective communications. The latter is especially important for e.g. cache updates in multicore systems. The Quarc NoC architecture [9] has been introduced as a Networks-on-Chip which is highly efficient in exchanging all types of traffic including broadcast and multicast. In this paper we present the hardware implementation of the switch architecture and the network adapter (transceiver) of the Quarc NoC. Moreover, the paper presents an analysis and comparison of the cost and performance between the Quarc and the Spidergon NoCs implemented in Verilog targeting the Xilinx Virtex FPGA family. We demonstrate a dramatic improvement in performance over the Spidergon especially for broadcast traffic, at no additional hardware cost.

1 Introduction

Networks-on-Chip (NoC) are an emerging communication-centric concept to develop future complex System on-Chip (SoC) by providing a scalable, energy efficient and reliable communication platform. In a NoC-based system, different components such as computational cores, memories and specialized IP blocks exchange data using a packet-switched network as a communication infrastructure.

Designing a flexible on-chip communication network for a NoC-based SoC platform, which can provide the desired bandwidth and at the same time be reused across many applications, is a challenging task which requires trading off a large number of system attributes such as performance, cost and size. In addition to the technological process for implementing the SoC, the topology, switching method, routing algorithm and traffic patterns are key factors which have a direct impact on the performance of a NoC-based SoC platform.

A packet switched NoC consists of an interconnected set of routers that connect IP cores together to form a given topology in order to enable efficient communication between the cores. The underlying topology of this architecture is the key element of the on-chip network as it provides a low latency communication mechanism and, when compared to traditional bus-based approaches, resolves physical limitations due to wire latency providing higher bandwidth and parallelism.

Deterministic routing and wormhole switching are regarded as the dominant routing and switching mechanisms in the NoC domain [15]. Those options mainly originate from the resource constraints imposed by the SoC medium on intermediate routers [2, 15].

Most recent proposed NoC architectures have been founded on top of ring [4, 5], fat tree [10], butterfly-fat tree [11], mesh [12], torus [15], folded torus [16] topologies. Nostrum [7], Æthereal [3], and Xpipes [6] are some examples of architectures used for on-chip networks. The Spidergon NoC [5] is also one of the ring-based architectures proposed recently.

By adopting wormhole switching, deterministic routing and homogeneous, low-degree routers the Spidergon scheme aimed to address the demand for a fixed and optimized network on-chip architecture to realize cost effective MPSoC (Multi-Processor SoC) development. However, the edge-asymmetric property of the Spidergon causes the number of messages that cross each physical link to vary

978-1-4244-3750-4/09/$25.00 ©2009 IEEE

severely, resulting in an unbalanced traffic on network channels and thus leading to poor performance of the whole network. This situation is even exacerbated when the network is under bursty traffic as a result of some operations such as broadcast.

In this paper we present and overview of the Quarc [9] architecture and discuss the hardware implementation of different components including the switch and the transceiver. Also, the paper explains the routing protocols and ensuing packet format for unicast, broadcast and multicast traffic. Broadcast traffic in NoCs is particularly important in MPSoC as it is the key mechanism for keeping caches in sync. The paper presents a comparison between the Quarc and the Spidergon which is a very similar NoC architecture. We demonstrate how the Quarc architecture addresses the poor performance of the Spidergon for broadcast traffic and show that the improved performance does not involve increased area cost.

The paper is organized as follows. Section 2 introduces the Quarc NoC and investigates the architecture of the switches and the transceiver. This section also discusses routing discipline, including unicast, broadcast and multicast. This section ends with a discussion of the packet format and the link-layer interface. Section 3 presents a comparison between the Quarc and the Spidergon schemes in terms of performance and cost based on a Virtex-II PRo FPGA implementation. Section 4 concludes the paper.

2 The Quarc NoC Architecture

The topology of an on-chip network specifies the structure in which routers connect the IPs together. Fat tree, mesh, torus and variations of rings are among the topologies introduced or adopted for the NoC domain.

Typically, a particular topology is selected in order to trade-off between a number of cross-cutting measures such as performance and cost. A number of important characteristics that affect the decision on adopting a particular topology are network diameter, the highest degree of nodes in the network, regularity, scalability and synthesis cost for an architecture.

The topology of the Quarc NoC is quite similar to that of the Spidergon NoC. Therefore, the next section presents a brief description of the Spidergon NoC, followed by introduction of the Quarc NoC.

2.1 The Spidergon NoC

The Spidergon NoC [5] is a network architecture which has been recently proposed by STMicroelectronics [13]. The objective of the Spidergon topology has been to address the demand for a fixed and optimized topology to realize low cost multi-processor SoC (MPSoC) implementations. In the Spidergon topology an even number of nodes are connected by unidirectional links to the neighboring nodes in clockwise and counter-clockwise directions plus a cross connection for each pair of nodes. Each physical link is shared by two virtual channels in order to avoid deadlock. Fig. 1 depicts a Spidergon topology of size 16 and its layout on a chip.

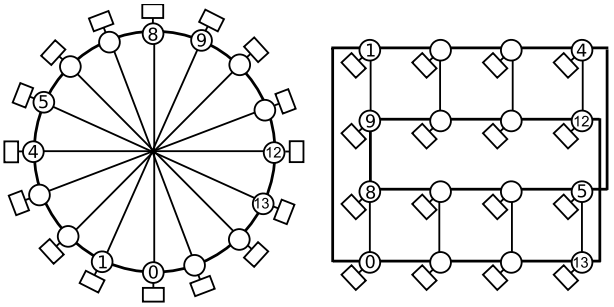

Figure 1. The Spidergon topology and the on chip layout.

The key characteristics of this topology include good network diameter, low node degree, homogeneous building blocks (the same router to compose the entire network), vertex symmetry and a simple routing scheme for unicast routing. Moreover, the Spidergon scheme employs packet-based wormhole routing which can provide low message latency at a low cost. Furthermore, the actual layout on-chip requires only a single crossing of metal layers.

In the Spidergon NoC, two links connecting a node to surrounding neighboring nodes on the "rim" carry messages destined for half of the nodes in the network, while the node is connected to the rest of the network via the cross link (or "spoke"). Therefore, the cross link can become a bottleneck. Also, since the router at each node of the Spidergon NoC is a typical one-port router, the messages may block on an occupied injection channel even when their required network channels are free. Moreover, performing broadcast communication in a Spidergon NoC of size N using the most efficient routing algorithm (presented in [9]) requires traversing $N - 1$ hops.

2.2 The Quarc Architecture

The Spidergon NoC is an efficient and low-cost NoC but has the main drawback of poor broadcast performance. Broadcasts are a key mechanism to maintain cache coherency in MPSoCs. As the number of cores in MPSoCs grows, cache synchronization will become a bottleneck in NoC-based MPSoCs unless the NoC has an efficient broadcast mechanism.

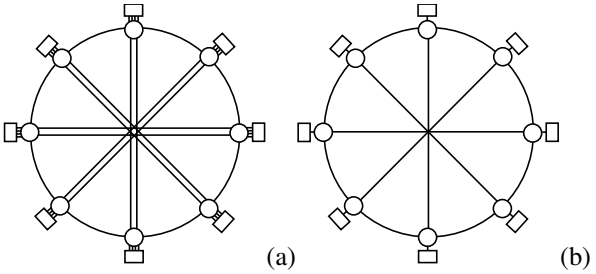

Figure 2. Quarc topology (a) vs Spidergon (b)

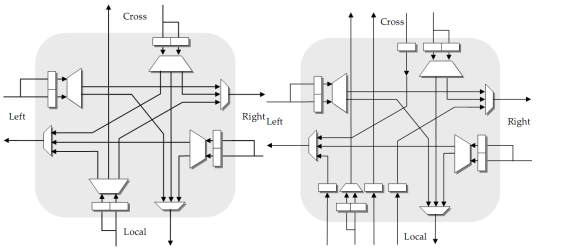

Figure 3. Minimal switch architectures for Spidergon (a) and Quarc (b) with deterministic routing

To address the issue of broadcast and multicast performance we propose the Quarc (Quad-arc) NoC, which improves on the Spidergon by making following changes: *(i)* adding an extra physical link to the cross link to separate right-cross-quarter from left-cross-quarter, *(ii)* enhancing the one-port router architecture to an all-port router architecture and *(iii)* enabling the routers to absorb-and-forward flits simultaneously. The Quarc preserves all other features of the Spidergon including the wormhole switching and deterministic shortest path routing algorithm, as well as the efficient on-chip layout.

The resulting topology for an 8-node NoC is represented in Fig. 2.

Unlike the Spidergon NoC, in the Quarc architecture a message will be blocked only when its requested network resources are occupied. This feature significantly enhances the performance of the network by reducing the waiting time at source node. Moreover, adding another physical link to the cross network links (making the topology edge-symmetric) improves access to the cross-network nodes. And last but not the least, the effect of the modification manifests itself most clearly when performing broadcast or multicast communication operations. In the Spidergon NoC, deadlock-free broadcast can only be achieved by consecutive unicast transmissions. The NoC switches must contain the logic to create the required packets on receipt of a broadcast-by-unicast packet. In contrast, the broadcast operation in the Quarc architecture is a true broadcast, leading to much simpler logic in the switch fabric; furthermore, the latency for broadcast traffic is dramatically reduced.

The analysis in Section 3 demonstrates that, surprisingly, the modifications proposed to the Spidergon topology and switch architecture to obtain the Quarc do not adversely affect area consumption of the resulting NoC compared to the original Spidergon. On the contrary, we demonstrate that the proposed modifications lead to both smaller switches and simpler routing logic.

2.3 Switch architecture

In this section we present the switch architectures of the Quarc and the Spidergon NoCs. Fig. 3 shows simplified diagrams for a Spidergon 4×4 switch with 1 local channel and 3 network channels (Fig 3(a)) and the Quarc architecture (Fig 3(b)). Both diagrams show minimal architectures for use with deterministic routing, i.e. the hardware is tailored to the paths allowed by the routing discipline.

The main differences are the number of local ingress ports (4 for Quarc) and the doubling of the cross-network link. Further differences are not obvious from the figure: the Quarc switch performs a true broadcast, i.e. the ingress multiplexers have a state that clones the flit; the decision logic is very simple (see 2.5). The Spidergon switch can only broadcast by unicast, and therefore needs a more complex logic to decide if a switch needs to clone a broadcast packet; furthermore, the ingress packet is not simply cloned but the header flit needs to be rewritten.

A top level block diagram of the Quarc switch is shown in Fig.4. The Quarc switch architecture consists of three fundamental modules, namely, *Input Port Controller* (IPC), *Switch*, and *Output Port Controller* (OPC). While IPC contains input buffer to store the flits, OPC does not contain any output buffer. This significantly reduces overall area of the Quarc switch. Any flit enters to the Quarc switch pass through four stages, namely, input buffering, routing, switching, and virtual channel allocation. The different modules responsible for controlling each of these stages are shown in Fig.4. The routing logic inside the Quarc switch is very minimal as a flit can either be destined for local node or needs to be forwarded on the same direction on the rim. Hence, the area occupied by the crossbar is very small due to its simplicity. The detailed description of each of these modules are given in the following section.

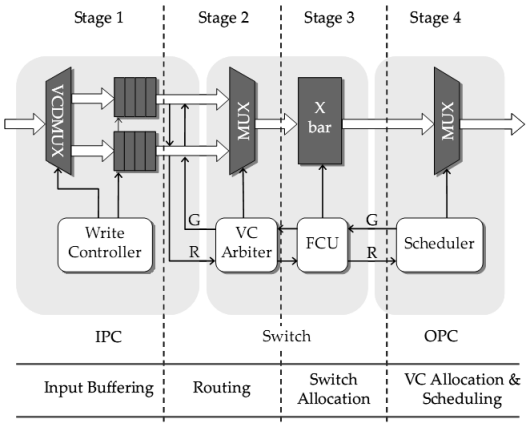

Figure 4. Functional block diagram of the Quarc Switch

2.3.1 Input Port Controller

When a flit arrives at a router it proceeds to an input port controller (IPC). The IPC, shown in Fig.4 performs two operations on incoming flits, namely, de-multiplexing and buffering. A *write-controller* reads the input handshaking signals and generates *write-enable* signals to store the flits into router's input buffer. The IPC incorporates two lanes of input buffers as shown in the Fig. 4. Therefore, in the current architecture, the Quarc switch is capable of supporting two virtual channels in parallel. The *full* signals generated by the buffer are used to construct the *channel-status* signal which is passed to the source of the message. The *empty* signal is passed to the next control block for routing and forwarding the flit to the destination. The buffers in the design are parametrized in width and depth.

The *Write controller* waits for the start-of-frame (*sof_in*) signal and stays in the *idle* state. Once it receives *sof_in* signal it goes to *write* stage and generate *write-enable* signal. The *write-enable* signal is also used with the *ch_to_store* to decide on which channel flit should be stored. The active low *eof_in* signal indicates end-of-frame to the *write controller* and the *write controller* goes back to *idle* stage again. The *reset_fsm_w* signal is used to reset the write controller if required.

2.3.2 The Switch

The switch is consisted of three main parts, namely, *Crossbar, VC Arbiter, Flow Control Unit (FCU)* as shown in the Fig. 4. The purpose of the *crossbar* is to forward the flits from IPC to a designated OPC decided from the destination address available in the header flit. The maximum throughput of the router is determined by the physical implementation of the crossbar. In the Quarc switch, left, right and one of the cross input port as shown in the Fig. 3, may require to send flits in maximum two possible destinations (i.e. local PE or forward to next node). The remaining input ports only have one possible destination OPC. This makes the Quarc switch very light weight compared to any other topology. For example, in *2D*-mesh topology every input can have four possible destinations which makes the crossbar very bulky.

The purpose of the *VC arbiter* is to allow only one of the input channels to send request to the *FCU* for access to *Output Port Controller (OPC)*. In normal traffic situation the *VC arbiter* might look like a redundant component. But in case of a header flit waiting at the router's input buffer and another header flit arrives at the other buffer of the same in-out port then the *VC arbiter* can allow the second packet to access to *OPC* if the destination route is free. The FSM for the *VC arbiter* has three states, namely, *idle, grant_0* and *grant_1*. A timer generates *times_up* signal to indicate that wait session is over in case of a flit is waiting for the *grant* signal and another flit has arrived at the other channel of the same input. Using this method of arbitration it is possible to generate equal opportunity between both channels of the same input port. The *VC arbiter* gets activated by the *empty* signals generated from the input buffer. If any of these signals becomes active the FSM moves to either of the *grant* state. A timer starts counting till timeout once FSM enters into any of these two stages. If there are two requests from both the input channels, then the FSM multiplexes between the input channels in case of a traffic block. When there is no request from the IPC, the FSM goes back to the *idle* state.

The *FCU* serves two functionality. First when it receives a request from the *VC arbiter*, it checks the header flit and sets the crossbar according to the destination address. Second, it sends a request to the corresponding *OPC* for access. If *OPC* does not send back the *grant* signal or otherwise the *OPC* is busy, then the *FCU* waits until either the *grant* signal or another request from the *VC arbiter* is obtained. If it receives the *grant* signal, then the *FCU* stores the switching information (i.e. the crossbar) till the tail flit of the same packet and sends the flit to the next router via its *OPC*. If it receives another request signal from the *VC arbiter* while waiting for a *grant* signal then it goes back to first state and follows the previous steps for the new flit from the other channel. If the *FCU* receives a body flit then it reads the switching information from the stored table and sends a request to corresponding *OPC*. In case of a tail flit, the *FCU* deletes the corresponding entry in the table as this is the last flit of the same packet.

2.3.3 Output Port Controller

The function of the *Output Port Controller* (*OPC*) is to schedule incoming requests and forward them to next node with proper handshaking signals. The *OPC* mainly consists of two main modules, namely, *scheduler* and a *data path multiplexer*. Note that the Quarc switch does not have any output buffer in the *OPC*. By not providing any output buffer the area requirement for the router is less. Although, extra logic is required to schedule incoming request but it occupies much lesser area than the buffers. There are four FSMs which governs the scheduler. Out of four, one is master FSM which handles requests from three different *IPCs*. It arbitrates between the requests and activates one of the slaves FSM. If this FSM gets into any of the grant state (*grant_a, grant_b* or *grant_c*) it activates one of the respective slave FSM. The slave FSM allocates one of the available channels as per the received *ch_status_n* signal from the next node. In case it has to multiplex between more than one *IPC* then it stores the virtual channel settings in a *VC allocation table*. The slave FSM has only three states, namely, *idle, vc_allocation* and *grant*. The *vc_allocation* state performs different activity according the type of the flit it receives. If it is a *header* flit then it checks the availability of channels and set the table with new allocation details. If it is a *body* type flit, then it reads from the table and follows the settings by made by the *header*. If it is a *tail* flit, it reads the table, follow the settings made by the *header* and then deletes the corresponding entry from the table.

2.4 The Transceiver Architecture

The Quarc NoC adopts an all-port router scheme. Therefore, the router has four ingress ports which are connected to four different links corresponding to four quadrants. Hence, before entering the router a flit must know its destination quadrant. This is done in the *network adapter* or *transceiver*. Fig. 5 shows a block diagram of such transceiver. The transceiver consists of five main components, namely, *write controller, quadrant calculator, buffer selector, FCU* and *buffers*. When a packet arrives at the transceiver, *write controller* divides the packet into a number of flits. The *write controller* also adds the flit type to the flit. For example, if a flit is of 32-bits after *write controller* adds its type, it becomes 34-bits which then traverse through the on-chip network. The *quadrant calculator* calculates the quadrant by comparing the source address from the router and the destination address. from the packet header. The detailed algorithm is explained in the section 2.5. According to calculated quadrant only one buffer receives write enabled by the *write controller*. On the other hand, *FCU* sends a request to the router's *OPC* unit for flit to be forwarded. Once *OPC* sends an *ack* signal, the *FCU* sends *read enable* signal to the corresponding buffer and the

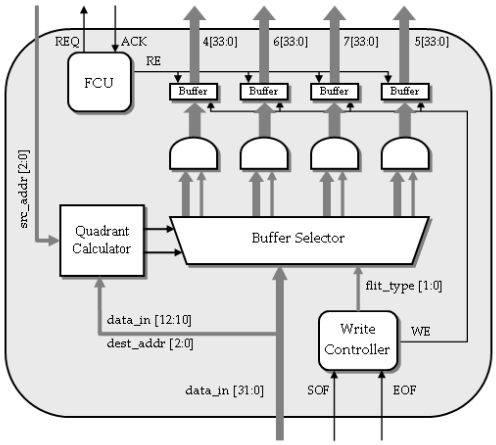

Figure 5. Functional block diagram of the transceiver

flit traverse to the next node via *OPC* of the source router.

2.5 Routing algorithm

2.5.1 Unicast routing

For the Quarc, the surprising observation is that there is no routing required by the switch: packets are either destined for the local port or forwarded to a single possible destination. Consequently, the proposed NoC switch requires no routing logic. The route is completely determined by the port in which the packet is injected by the source. Of course, the NoC interface (transceiver) of the source processing element (PE) must make this decision and therefore calculate the quadrant as outlined above. However, in general the PE transceiver must already be NoC-aware as it needs to create the header flit and therefore look up the address of the destination PE. Calculating the quadrant is a very small additional action.

2.5.2 Broadcast routing

Broadcast, the key motivator behind the Quarc topology, is elegant and efficient: The Quarc NoC adopts a BRCP (Base Routing Conformed Path) [1] approach to perform multicast/broadcast communications. BRCP is a type of path-based routing in which the collective communication operations follow the same route as unicasts do. Since the base routing algorithm in the Quarc NoC is deadlock-free, adopting BRCP technique ensures that the broadcast operation, regardless of the number of concurrent broadcast operations, is also deadlock-free.

To perform a broadcast communication the transceiver of the initiating node has to broadcast packet on each port

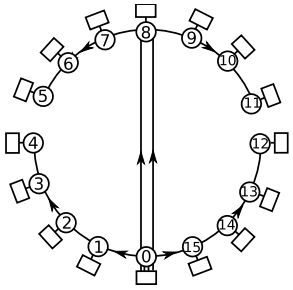

Figure 6. Broadcast in a Quarc of 16 nodes

of the all-port router. The transceiver tags the header flit of each of four packets destined to serve each branch as broadcast to distinguish it from other types of traffic. The transceiver also sets the destination address of each packet as the address of the last node that the flits stream may traverse according to the base routing. The receiving nodes simply check if the destination address at the header flit matches its local address. If so, the packet is received by the local node. Otherwise, if the header flit of the packet is tagged as broadcast, the flits of the packet at the same time are received by the local node and forwarded along the rim. This is simply achieved by setting a flag on the ingress multiplexer which causes it to clone the flits.

The broadcast in a Quarc NoC of size 16 is depicted in Fig. 6. Assuming that Node 0 initiates a broadcast, it tags the header flits of each stream as broadcast and sets the destination address of packets as 4, 5, 11 and 12 which are the address of the last node visited on left, cross-left, cross-right and right rims respectively. The intermediate nodes receive and forward the broadcast flit streams, while the destination node absorbs the stream.

2.5.3 Multicast routing

Similar to broadcast, in multicast operation, the last node to be visited must be specified as destination address in the header flit. For broadcast all nodes in the path from source to destination are the receiver nodes. In case of multicast the target addresses are specified in the *bitstring* field. Each bit in the *bitstring* represents a node; its hop-distance from the source node corresponds to position of the bit in the *bitstring*. The status of each bit indicates whether the visited node is a target of the multicast or not. Consequently, broadcast is simply a special case of multicast where every node is a target. Next section presents the packet format for different traffic types.

2.6 Packet Format in the Quarc NoC

The Quarc scheme is a packet switched network employing wormhole switching. In wormhole switching a packet is

Bits:	[33:31]	[30:14]	[13:8]	[7:2]	[1:0]
Unicast Header:	0		destination addr.	source addr.	0
Multicast Header:	1	bitstring	destination addr.	source addr.	0
Broadcast Header:	2		destination addr.	source addr.	0
Body:	Payload				1
Tail:	Payload				2

Figure 7. Flit type formats in the Quarc NoC

divided into elementary units called flits, each composed of a few bytes for transmission and flow control. The header flit governs the route and the remaining data flits follow it in a pipelined fashion. If the header flit blocks, the remaining flits are blocked in situ.

Since the Quarc scheme adopts a simple deterministic routing, the packet format for unicast and collective communication is quite simple. For a Quarc NoC employing flit size of 34 bits various flit types composing a packet are depicted in Fig.7. Bits $[1:0]$ denote the flit types namely: *header*, *body* and *tail*. And the last 3 bits of header flits represent traffic types which are shown for *unicast*, *multicast* and *broadcast.* Each packet must have the header and tail flits.

Please note that due to the scalability issues of the Quarc NoC, it is assumed that the network size may be up to 64 nodes (max diameter is $n/4$, if $n > 64$ then the max diameter is larger than the max diameter for mesh $2(\sqrt{n}-1)$). However, larger networks may employ flits of larger size or to use multi flit headers for specifying multi-addresses for multicast operations.

2.7 Link Layer Interface

The Quarc NoC uses the signals and handshaking mechanism of Xilinx's *LocalLink* protocol for the link layer interface. Fig. 8 shows a high level block diagram of the *LocalLink* interface used for the Quarc switch and illustrates its basic topology. In this 2-Channel (VC) example, *CH_STATUS_N [1:0]* bus shows that maximum two channels can accept data. According to the value of *CH_STATUS_N [1:0]*, the sender decides on which VC data has to be sent and corresponding value is sent through the *CH_TO_STORE* signal. The five steps to transfer a channelized frame with channel ready signaling are:

- The destination interface asserts the *CH_STATUS_N [1:0]* bus to indicate virtual channel availability. A typical application asserts a logic zero on the appropriate bus bits to indicate channels that can accommodate at least one full-sized LocalLink frame.

- The source interface responds by asserting *SRC_RDY_N*.

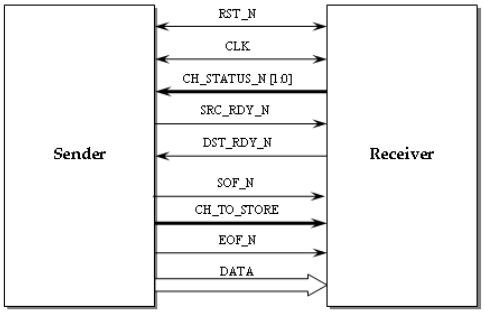

Figure 8. High level block diagram of local link specification

- The destination responds by asserting *DST_RDY_N*.
- The source interface responds by asserting *SOF_N*, driving the data bus, and driving the number of the channel that is transferring data to the destination onto the *CH_TO_STORE* bus (in this case WIDTH = 1).
- The source interface ends the transfer by asserting the *EOF_N* signal.

3 Cost and Performance Analysis

Deploying a particular NoC architecture typically involves a trade-off between performance and cost. This section presents a comparison between the Quarc and the Spidergon NoCs of the cost in terms of area and performance in terms of average latency.

3.1 Cost Analysis

In this section, we demonstrate that the Quarc switch is smaller in size and at the same time is less complex than the Spidergon switch and this saving in area outweighs the overheads incurred by additional ports and the area for additional links.

We assume that every node of NoC hosts a processing element (PE), typically a microprocessor with local memory. The difference in resource utilization at the PE between the Quarc and the Spidergon NoCs is very small. In both cases the packets are stored in RAM and the address of the packets are queued. For the Quarc NoC, the PE queues the addresses in four separate queues, effectively making the routing decision by doing so. For the Spidergon NoC, the PE will put the addresses in a single queue. As the variance on the occupation of the individual queues (σ for Quarc), is twice as large as the variance on the occupation of the combined queue ($\sigma/\sqrt{4}$ for Spidergon), the queues has to be twice as deep. This is, of course, a small memory overhead as the address size is a fraction of the packet size. Also, note that the actual packet memory requirements are identical for both the Quarc and the Spidergon NoCs.

To present a comparison between the two architectures, we have implemented 16, 32, and 64-bits versions of both the Quarc and the Spidergon switches in Verilog targeting the Xilinx Virtex-II Pro FPGA. In order to make assembling and upgrading of the switch simple, the switch architecture is designed in a modular fashion as shown in Fig.4.

Module	Slice Count
Input Buffers	735
Write Controller	7
Crossbar & Mux	186
VC Arbiter	30
Flow Control Unit (FCU)	64
Ouput Port Controller (OPC)	431

Table 1. Module-wise cost analysis of a 32-bits Quarc switch

For 32-bits version of a Quarc switch the number of occupied FPGA Slices is $1,453$, whereas similar version of the Spidergon switch occupies $1,700$ Slices. A more detailed module-wise area occupancy for a Quarc switch of 32-bits version is shown in the Table1. Note that the amount of area occupied by the crossbar and FCU are very minimal. This result supports the argument that the Quarc NoC does not have complex crossbar or routing logic, which saves the area of the switch. A comparison of the cost analysis in terms of *Slice count* for various versions between the two switches is shown in the Fig.12.

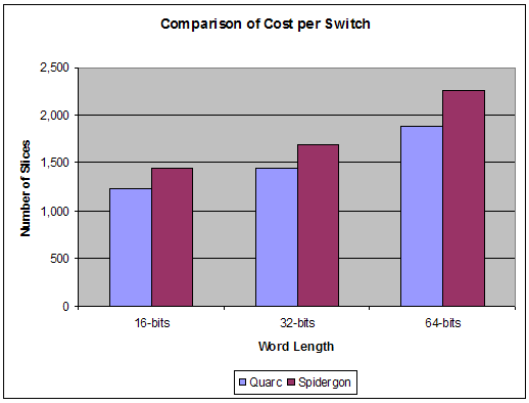

Figure 12. Cost comparison between Quarc and Spidergon switches

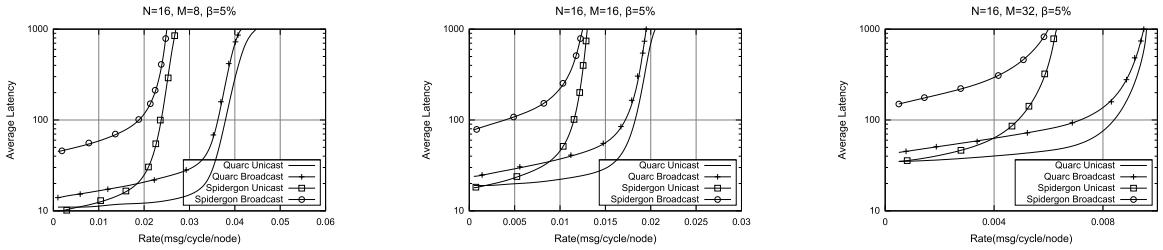

Figure 9. Comparison of Quarc and Spidergon for M=8,16,32

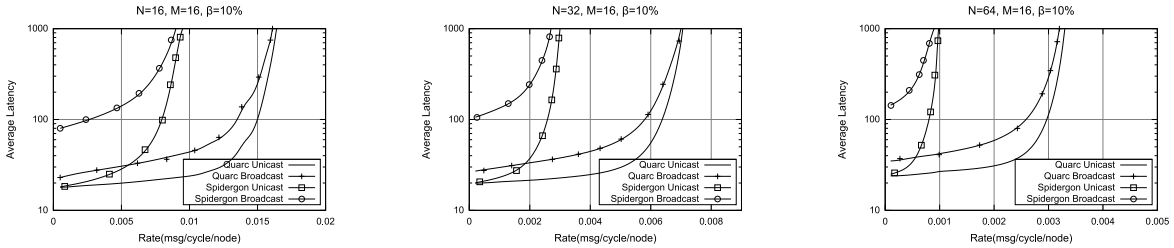

Figure 10. Comparison of Quarc and Spidergon for N=16,32,64

3.2 Performance Analysis

To evaluate the performance of the Quarc NoC architecture we have developed a discrete event simulator operating at flit level using OMNET++ [14]. The simulator has been verified extensively against analytical models for the Spidergon and mesh topologies employing wormhole routing [8].

The performance of the Quarc architecture has been evaluated against the Spidergon for numerous configurations by changing the network size, message length and the rate of broadcast traffic. In graphs, N, M and β represent the number of nodes, message length and rate of broadcast traffic respectively. The horizontal axis in the figures shows the message rate per node while the vertical axis describes the latency.

Fig. 9 shows the average latency experienced by unicast and broadcast traffic in the Quarc and the Spidergon NoCs in configurations where network size $N = 16$ and broadcast rate, $\beta = 5\%$ are fixed while the message length can be 8, 16 and 32. Fig. 10 compares the simulation results against the analysis for the networks ranging from 16 to 64 nodes with a fixed message length of 16 and 10% broadcast traffic.

As can be seen from the figures the Quarc NoC outperforms the Spidergon over the complete range of N, M and β . The most striking performance difference is clearly observed for broadcast traffic, with almost an order of magnitude improvement on the latency. However, the unicast latency is overall at least a factor of 2 lower. Also, the graphs clearly show that the Quarc NoC is capable of sustaining a much higher load before it saturates. This in turn indicates that the throughput of the Quarc NoC is significantly higher than the Spidergon NoC.

The graphs in Fig. 11 compare the average latency in the Quarc and Spidergon NoC for the configuration where the network size ($N = 64$) and message length ($M = 16$) are fixed while the broadcast rate, β, is varying between 0 to 10%. The graphs reveal the Quarc NoC is highly capable of sustaining the broadcast traffic. As can be seen the injection of the broadcast traffic into the Spidergon NoC severely reduces the sustainable load in the network. In the Quarc NoC the adverse impact of the broadcast traffic on the sustainable load and on the performance of the unicast is hardly appreciable.

4 Conclusion

The aim of the Quarc NoC was to provide an efficient NoC for exchanging all types of traffic including collective communications in MPSoC systems. In this paper we have presented the hardware design of the components of the Quarc NoC including the switch and transceiver archi-

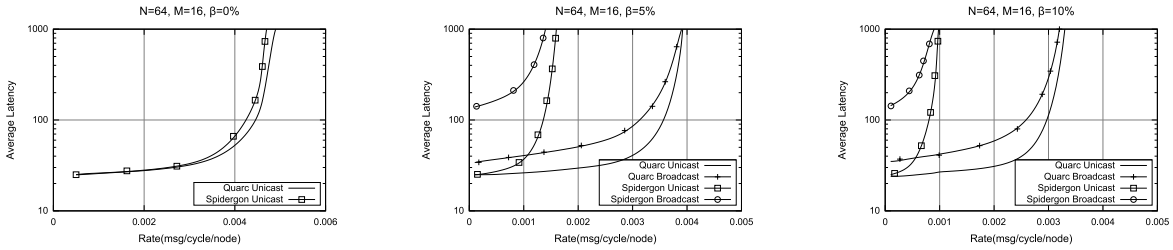

Figure 11. Comparison of Quarc and Spidergon for β= 0%, 5%, 10%

tecture, as well as results for an implementation in Verilog targeting the Xilinx Virtex-II Pro FPGA.

The paper has presented a comparison of performance and cost between the Quarc and Spidergon NoCs. Our analysis has shown that the Quarc outperforms the Spidergon over the complete range of number of nodes, message length and broadcast rate.

In particular we show a dramatic performance improvement for broadcast traffic which is of special importance in MPSoC systems as it is used to synchronise the caches.

Equally important, our cost analysis showed that, surprisingly, the additional performance gain obtained at no extra cost compared to the Spidergon NoC.

Our next objective is to compare the performance of the Quarc against other widely used NoC architectures such as mesh and torus.

References

[1] D.K. Panda et al. Multidestination Message Passing in Wormhole k-ary n-cube Networks with Base Routing Conformed Paths. *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[2] E. Bolotin, et. al. QoS architecture and design process for Networks-on- Chip. *Journal of Systems Arch*, 2004.

[3] E. Rijpkema, K. Goossens, and P. Wielage. Router Architecture for Networks on Silicon. *Progress , 2nd Workshop On Embedded Systems*, 2001.

[4] F. Karim et al. An Interconnection Architecture for Networking Systems on Chip. *IEEE Microprocessors*, 22(5):36–45, Sept. 2002.

[5] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. *Int'l Symposium on System-on-Chip*, 2004.

[6] M. Dall'Osso et al. xpipes: a Latency Insensitive Parameterized Network on-Chip Architecture for Multi-Processor SoCs. *Int'l Conf. on Computer Design*, 2003.

[7] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The nostrum backbone-a communication protocol stack for Networks on Chip. *Int'l Conf. on VLSI Design*, 2004.

[8] M. Moadeli et al. Communication Modeling of the Spidergon NoC with Virtual Channels. In *ICPP*, 2007.

[9] M. Moadeli, W. Vanderbauwhede, and A. Shahrabi. Quarc: A Novel Network 0n-Chip Architecture. *Parallel and Distributed Systems, International Conference on*, 2008.

[10] A. G. P. Guerriert. A generic architecture for on-chip packet-switched interconnections. *Design Automation Conf. (DAC)*, pages 683–689, 2001.

[11] P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of Switch for Network on Chip Applications. *Int'l Symposium on Circuit and Systems (ISCAS)*, 5:217–220, May 2003.

[12] S. Kumar et al. A network on chip architecture and design methodology. *Int't Symp. VLSI (ISVLSI)*, pages 117–124, 2002.

[13] STMicroelectronics. www.st.com.

[14] A. Varga. Omnet++. *IEEE Network Interactive, in the column Software Tools for Networking*, 2002.

[15] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *Design Automation Conf. (DAC)*, pages 683–689, 2001.

[16] W. J. Dally and C.L. Seitz. The Torus Routing Chip. Technical report, Technical Report 5208:TR: 86, Computer Science Dept. California Inst. of Technology, 1-19, 1986.