An Admission Control System for QoS Provision on a Best-effort GALS Interconnect

Shufan Yang, Steve B. Furber, Yebin Shi, Luis A. Plana
School of Computer Science, The University of Manchester
Oxford Road, Manchester M13 9PL
Email: shufan.yang, sfurber, shiy, plana @cs.man.ac.uk

Abstract

A central admission control mechanism is introduced in order to provide efficient Quality-of-Service (QoS) support for different types of application over a best-effort Globally-Asynchronous Locally-Synchronous (GALS) interconnect fabric. The mechanism is applied at the ingress edges of the fabric using tokens to allocate dynamic network resources and prevent network saturation. Analysis and simulation results are presented to show the effectiveness of the method. The control provides service guarantees on the network while using a modest physical area because of the simplicity of the control logic. This is a cost-effective way to provide QoS in packet-switched interconnect because of its independence from other network components.

1. Introduction

A typical System-on-Chip (SoC) design requires a wide variety of functional blocks and I/O interfaces. However, it is difficult to manage growing numbers of on-chip blocks using globally clocked interconnect [1]. The Globally-Asynchronous Locally-Synchronous (GALS) methodology provides a solution. It keeps the efficiency in gate count of a synchronous implementation at the local level while removing the need for global timing convergence for the full SoC.

Currently proposed GALS techniques are attractive but are, in general, best-effort interconnects. Best-effort interconnects are not designed to meet application performance requirements all the time. Quality-of-Service (QoS) is a form of quality assurance that can be used to tackle this problem. In the context of GALS interconnect, QoS is a communication service that makes guarantees regarding the speed with which data will be transmitted to the target [2]. However, best effort interconnects are unlikely to meet QoS policy objectives in terms of bandwidth and latency guarantees without additional resources [3].

In this paper, we introduce a token-managed admission control mechanism for QoS to satisfy the communication demands of the applications in the SpiNNaker system [4]. The SpiNNaker system is a massively-parallel multiprocessor based on a high-performance, GALS on-chip interconnect. SpiNNaker provides a complete, verified and practical platform for GALS experiments and will be fabricated soon on a 130nm silicon process. The applicability of the low-cost QoS support mechanism presented here is not limited to the SpiNNaker chip and will result in guidelines for designers of industry-relevant multi-processor System-on-chip (MPSoC).

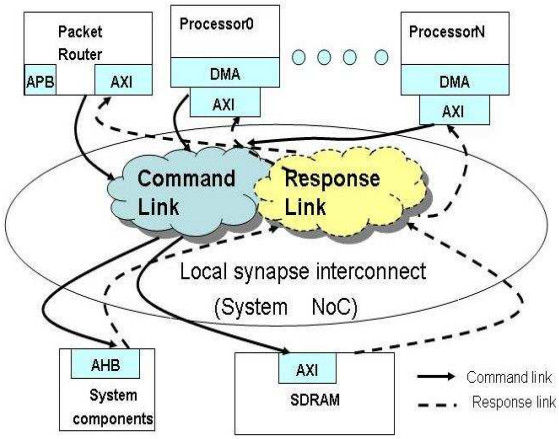

Figure 1. SpiNNaker chip interconnect

2. Related Work

Well-known QoS support schemes for on-chip interconnect are based on resource scheduling and reservation. For example, a scheme with prioritised reservation of specific

978 -1-4244-1839-8/08/$25.00 © 2008 IEEE

links [5] uses dedicated buffers in the network switches for storing priority classification information. Unfortunately, these buffers usually incur a large area penalty [5]. Another scheme uses reservations by virtual channels. The Mango project [6] uses asynchronous latency guarantee (ALG) scheduling on virtual channels to provide QoS hard guarantees. However, it is a costly implementation. For a case the capacity is reserved but then not used by one client, then it is unavailable to other clients. Clearly, a cost-effective way to provide QoS in GALS interconnect is needed.

The mechanism of admission control for QoS support has been addressed in a few papers [7, 8, 9, 5], where the modules are evaluated by high level simulation using OPNET. Even though OPNET is a convenient tool for hierarchical network modelling, the accuracy of the results from the high-level simulation is not very convincing in practice because software never performs tasks in parallel, as does hardware. Nollet *et al* describe another more complex admission control machanism [8, 9]. However, the value of such theoretical work is usually not revealed until some practical applications can be run on the real chip platform.

3. GALS Interconnect

3.1. SpiNNaker chip

The SpiNNaker chip is a scalable processor chip containing multiple ARM cores to implement a general-purpose programmable neural device model [10]. The chip implements two distinct interconnects: one for communication between processing nodes and the other for connecting system components to the processing nodes [4]. The QoS scheme presented in this paper is applied to the system interconnect.

The system interconnect provides a GALS infrastructure for connecting processors to off-chip memory (e.g. SDRAM) and other system components, as shown in figure 1. The fabric we propose is based on a crossbar topology with two dedicated communication links: the command link is used by the initiator devices that can initiate a communication transaction (e.g. processor units and DMA) to send requests to the targets and the response link is used by the targets, such as SDRAM, to respond to transaction requests. The two-physical-link fabric is based on CHAIN technology [11]. A key feature of a crossbar is to enable multiple initiator devices to access different target devices simultaneously, although accesses to the same target would become a bottleneck.

The interconnect fabric, a best-effort interconnect, was implemented using the commercially-available CHAINworks tool suite [12, 13]. The interconnect fabric is implemented in a *self-timed* fashion, based on a handshake mechanism without requiring a clock signal and without relying on the notion of time [4]. When a sender puts data on a link, it activates the associated *VALID* signal and it should not send any more data until the receiver signals that it is ready. When the receiver has consumed the validated data it activates the corresponding *ACKNOWLEDGE* signal to allow the sender to proceed.

3.2. Network interface

To maintain the parallel processing model within the chip, we selected the AXI (AMBA Advanced eXtensible Interface) protocol [14] for use in the high-speed, system interconnect interfaces such as the processor and SDRAM interfaces. Other system components use the AHB protocol, as shown in Figure 1. AXI is targeted at high-performance, high-frequency system design. According to the AMBA AXI specification, the interconnect consists of five independent channels: *read address*, *read data*, *write address*, *write data* and *write response*. The use of five separate communication channels is intended to improve communication performance dramatically [14]. In the SpiNNaker GALS interconnect, the read address, write address and write data channels are driven by the initiator devices and are implemented using the command link in the fabric. The read data and write response channels are driven by the target devices and use the response link.

As mentioned earlier, the system interconnect is a best-effort interconnect. Furthermore, the AXI interfaces to the interconnect do not provide any means to control the initiators. These devices can issue new transaction requests even if the fabric is very busy and experiencing congestion. A sequence of burst-read requests to the same target may not create congestion in the command link but, if the bursts are long, could easily create problems in the response link. Consequently, saturation in the response link can result in back-pressure through the command links. In this case, the fabric will rapidly become saturated. A mechanism to control the admission of new transactions into the interconnect can avoid this fabric congestion.

3.3. Requirement for QoS support

QoS support can be classified into hard and soft guarantee services. Hard guarantee services make sure that the communication requirements are always met and are required only by critical, real-time applications. Soft guarantee services relax the guarantees. The soft requirements can be established in terms of a desired delay bound and a maximum percentage of packets arriving later than a given threshold. Most of the proposed QoS support schemes are hard guarantee services and incur large area costs. However, for many applications, it is not worth paying so high a

price. A soft guarantee service is adequate.

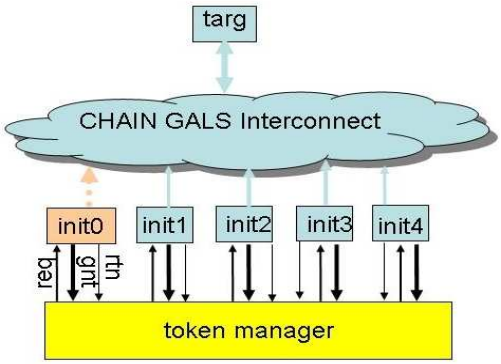

Figure 2. Conceptual view of the QoS system

Soft and hard guarantees can both be quantified in terms of performance parameters such as bandwidth, latency and loss probability. There are three types of bounds for QoS support: bandwidth bounds for specific links, latency bounds for worst-case traffic patterns and loss bounds for reliable transmission [15]. The latency bound is not a useful metric if the bandwidth is too small. Moreover, in complex systems it is difficult to measure latency accurately given that latency includes device response time and transmission latency. Another metric, the loss bound, is not meaningful in the context of our GALS interconnect, given that the handshake mechanism introduced earlier guarantees that no transactions are lost. For these reasons, we focus on a soft bandwidth and soft latency guarantee service to provide adequate QoS support in the SpiNNaker chip.

4. Token-Managed Admission Control

We implement a token-managed admission control (TMAC) mechanism to provide soft guarantees for QoS traffic (traffic initiated by a high-demand initiator) over a best-effort (BE) interconnect. Concurrently, BE traffic (traffic that has no strict deadlines) still has an opportunity to access the fabric without experiencing heavy traffic congestion. Figure 2 illustrates the conceptual view of the system. Each initiator has an individual interface to the token manager. When the interconnect fabric is operational, all initiators with pending requests compete for bandwidth allocation. TMAC takes responsibility for deciding which initiator has priority to access the fabric.

TMAC manages access permission to the fabric. There are two basic token transactions, which are controlled by the token manager:

- token assignment: when one initiator requests access to the fabric, TMAC grants the request if there are available tokens. The initiator can then send a communication transaction to the fabric.
- token return: once the initiator completes the transaction, it returns the token to TMAC.

4.1. Principle of operation

The token-managed admission control (TMAC) is a simple, synchronous mechanism. A definition of each signal is given in table 1.

Table 1. TMAC signals

Signal	Function
request	request from initiator
grant	grant token to initiator
return	initiator ends transaction

4.1.1 Read transactions

The timing diagram of a TMAC read operation, shown in figure 3, illustrates both the token assignment and the token release behaviour. The area labelled A in the figure shows a successful transaction by initiator *init0*: the *request* signal is set when the initiator issues a request. If there are free tokens, the *grant* signal goes high, the *request* signal is reset accordingly and the read transaction (a 4-word burst) takes place. The completion of the read transaction is indicated by *return* going high and then *grant* goes down. The figure also illustrates what can happen when tokens are not available. The area labelled B in the figure shows that, although *init1_request* has been driven high to request a token, initiator *init1* must wait for an available token. When there is a free token, the request is granted.

TMAC is a priority-based mechanism. Every initiator is assigned a priority level and TMAC will grant tokens to higher priority devices first. A round-robin algorithm is used to decide between equal priority requests. Figure 3 shows that high-priority initiators *init2* and *init4* are granted tokens before low-priority *init1* and *init3*.

TMAC is a simple, low-overhead mechanism. The area labelled C in Figure 3 shows TMAC can grant available tokens in a single clock cycle. Additionally, back-to-back requests by the same initiator are allowed, i.e., an initiator can release a token and request a new one in the same clock cycle.

4.1.2 Write transactions

Figure 4 shows how write transactions operate. The area labelled A in the figure shows the first transaction of initiator *init0*. *Init0* issues a request and the first write data

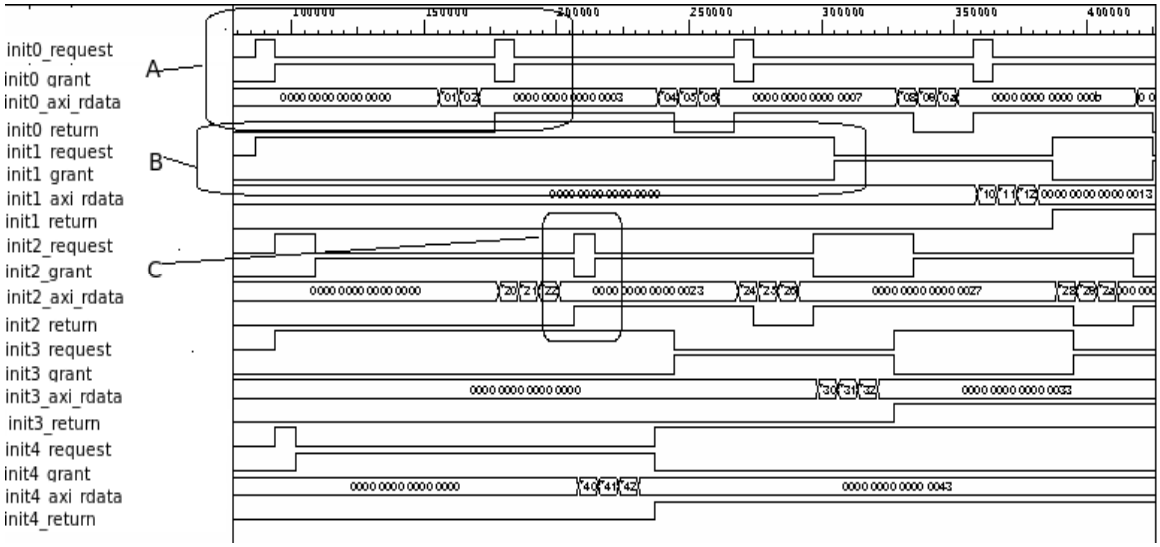

Figure 3. TMAC read transaction timing diagram

(labelled init0_axi_wdata_0) is sent out at same time. The token is granted and the (4-word burst) write transaction can take place. The token is eventually released by *init0*, thus completing the transaction. The areas labelled B show all initiators starting (4-word burst) writes. Even though the requests are issued at the same time, the write transactions start only when a token is granted. As shown in the figure, the tokens are granted in sequence.

Due to the sharing of the command-link bandwidth between write data and write addresses, write transactions are different from read transactions. It is inevitable that the command link is more easily saturated by write data. In addition, initiators can experience very long round-trip latencies since they have to wait for the transmission of the write response from the target device. The area labelled C in figure 4 shows that, even though a token has been granted quickly, initiator *init0* must wait a long time before starting its second transaction due to the saturation of the command link.

Although TMAC is able to influence the write bandwidth allocation, the dynamics may well be different due to the command link saturating. Fortunately, read transactions dominate system performance in typical applications (they are three times more frequent than write transactions [16]). Thus, the design has been driven by read transactions and the experimental results presented in this paper involve only read transactions.

4.2. Design view

Figure 5 shows a block diagram of the TMAC implementation. The scheme is fully programmable to be able to choose any initiator to have privilege for bandwidth reservation. This schematic shows the parametrised arbiter that implements two common schemes: round-robin arbitration and priority arbitration. In the figure, the priority memory stores priority settings and the scan memory registers the state of the round-robin algorithm.

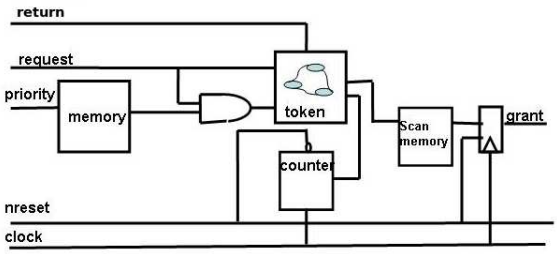

Figure 5. TMAC schematic view

With regard to the latency overhead of TMAC, we tried to make TMAC more efficient by using a parallel token assignment design. After evaluation of the completed design, we realised that supporting parallel assignment was not cost-effective. If there is a one cycle overhead for assigning one token, the performance of the QoS support decreases very little by assigning a single token on every clock cycle. Furthermore, time-critical applications normally have predictable performance demands. Avoiding high levels of congestion on the fabric is more crucial than slight latency overheads in TMAC.

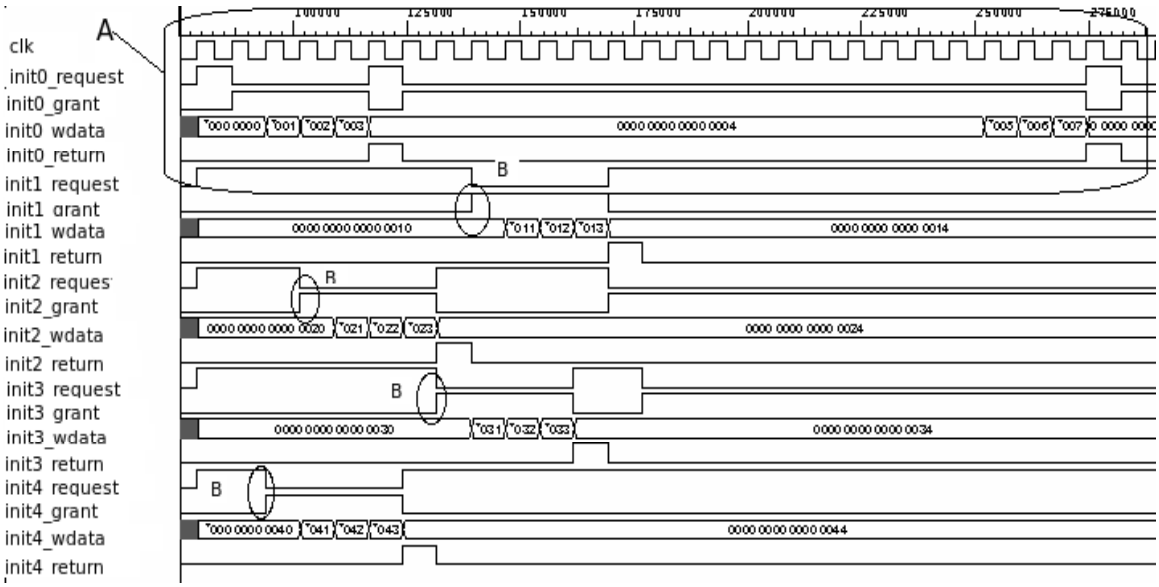

Figure 4. TMAC write transaction timing diagram

4.3. TMAC bandwidth guarantees

TMAC provides QoS by controlling the priority of token requests, but the possibility of bandwidth reservation depends on how many tokens are preserved for each initiator. For example, if there are 3 valid tokens in a five-initiator to one target system, the initiator that has the QoS requirement will get a 1/3 bandwidth guarantee as a result of the priority granting of a token; the other initiators will share the remaining 2/3 bandwidth. If the fabric can accommodate more initiators without heavy congestion, TMAC can be configured with one more token. In that case, it will give a lower bandwidth guarantee than that of the previous case, because the priority initiator would only own 1/4 of the bandwidth allocation. Another scenario is that an initiator can be allowed to send 2 outstanding commands, by granting 2 tokens in a 3 token system. In this case, the initiator will theoretically get a 2/3 bandwidth allocation.

To control the bandwidth reservation dynamically, our system would require a small number of changes. Firstly, the interface has to be modified to make it fully configurable. Secondly, the interface must be extended to support configuration setting, which would probably involve some control algorithms and support for application mapping.

4.4. Synthesis results

The whole system has been simulated using a back-annotated netlist on a UMC 130nm process. The TMAC implementation runs at 100 MHZ and the initiators run at 100 MHZ. The target (an SDRAM controller model) runs at 133 MHZ, which is a standard frequency supported by the ARM PL340 [17].

It is interesting to note the influence on the TMAC size of the number of initiators. We have run several experiments using CHAIN GALS interconnect with multiple initiators and one target. The results are given in Table 2, where the first column is the number of initiators connected to the one target. The second column shows the TMAC cell size. The TMAC cell size increases only slowly with the number of initiators. Hence, it has good scalability.

Num. of initiator	TMAC cell size
5	0.029mm^2
10	0.032mm^2
15	0.055mm^2
20	0.068mm^2

Table 2. TMAC area vs number of initiators

4.5. Scalability issues

TMAC is a scalable scheme for QoS provision. The current design can easily be adapted to operate on complex systems (e.g. 20 processor nodes on a chip). This only requires enough interfaces and the initialization of the priority state according the QoS demands of the initiators.

Unfortunately, the mechanism has potential scalability problems due to the latency of access to the centralized control TMAC. The latency for token assignment increases as

the system scale up. Clearly, these problems need to be addressed. However, for middle-scale systems, the TMAC mechanisms provide benefits and improvements over other QoS mechanisms, both in area and performance.

5. QoS support in action: an example

This section shows a simple example of how QoS support is provided and how congestion is prevented at the ingress of fabric.

The TMAC mechanism is triggered when an initiator issues a request. We consider QoS support on a GALS system and evaluate the performance during peak traffic loads. In this section, the performance of the proposed token-managed admission control (TMAC) is examined by means of back-annotated simulation using UMC 130nm process technology on a 48-bit fabric (using 3-of-6 encoding).

The QoS criteria we employ in this paper are "bandwidth reservation" and "bandwidth utilisation". The bandwidth reservation is the proportion of the total bandwidth allocated to a given initiator over a defined period of time. The period of time we use in the simulation is the simulated time for a given number of read transactions to complete. The bandwidth allocation can similarly be measured as the number of read transactions completed by the given initiator as a proportion of the total number of read transactions serviced by the target (on the assumption that all transactions carry the same volume of data).

Normally, bandwidth utilisation refers to the percentage link utilisation over a specified simulation time. However, the SpiNNaker chip employs an array of 20 ARM968 CPUs accessing a single off-chip SDRAM, thus the available bandwidth is determined by the SDRAM target interface [18, 4, 19]. Therefore the bandwidth utilisation here is simply the total percentage utilisation of the available target bandwidth.

In the following graphs, we also use "mean end-to-end latency", which accounts for different cases of uniform traffic. There are two possible waiting times within the latency in addition to the GALS fabric delay: the first is waiting for a token to become available, because there are only 3 tokens for 5 initiators, and the second is the response time of the target device. Section 5.2 elaborates on the experimental analysis of read transactions.

5.1. Evaluation platform

There are currently no public simulation tools available to aid SoC designers to generate extensive and varied regular traffic patterns and application-oriented traffic. Most current performance evaluation on GALS-based interconnect is based on packet generation from an infinite source queue. This method includes the input timing of each packet when it is generated, which is inaccurate [5]. Instead, we use a synthetic traffic pattern. These traffic profiles will be built manually using a fixed burst mode (e.g. 4-word burst or a 16-word burst).

To exemplify the efficiency of QoS support in a GALS interconnect, we use an "all-to-one" uniform traffic pattern. The memory in the SpiNNaker chip is currently an off-chip SDRAM with 1Gb capacity [10]. It is easy to expand available global memory simply by using a larger memory device. However, the competition among initiators for SDRAM utilisation will not be relieved. This case study of an "all-to-one" example is a useful indicator of the likely performance on the real problem.

The experimental scenarios are five initiator devices using the AXI protocol connected to one target device also using the AXI protocol (e.g. an SDRAM controller).

The following evaluation model is used:

- the 5-to-1 GALS interconnect netlist generated by CHAINworks [12]
- five verilog models of AXI initiator devices
- one verilog model of an AXI target device. Assuming the target buffer is infinite, the target produces one beat of data (64 bits of data) in one clock cycle
- the stimuli files (uniform random 4-word-burst read transactions)
- the System Verilog test bench

5.2. Evaluation of QoS support

A uniform workload model implements read transactions between five initiators and one target. The simulation shows that initiators with high priority get what they request, and the remainder receive an equally balanced service.

Figure 6 (A) shows a comparison of the mean latency of QoS traffic from initiator0 with best effort (BE) traffic from initiator1, initiator2, initiator3 and initiator4. As seen in figure 6 (A), the best effort traffic is noticeably affected by the traffic mode when bandwidth utilisation is above 50%. When the maximum fabric utilisation is approached, the mean latency of the BE traffic varies significantly compared with the QoS traffic. This shows that the token-managed admission control gives a boost to guarantee latency for the one initiator that has a QoS requirement.

The significance of a latency guarantee is that we constrain the injection of data from sources with admission control. Although TMAC introduces an extra time overhead, it is an efficient way to avoid fabric congestion thereby providing a latency guarantee for the interconnect. However, the different burst modes affect the latency guarantee slightly because of the bottleneck of the one target, as

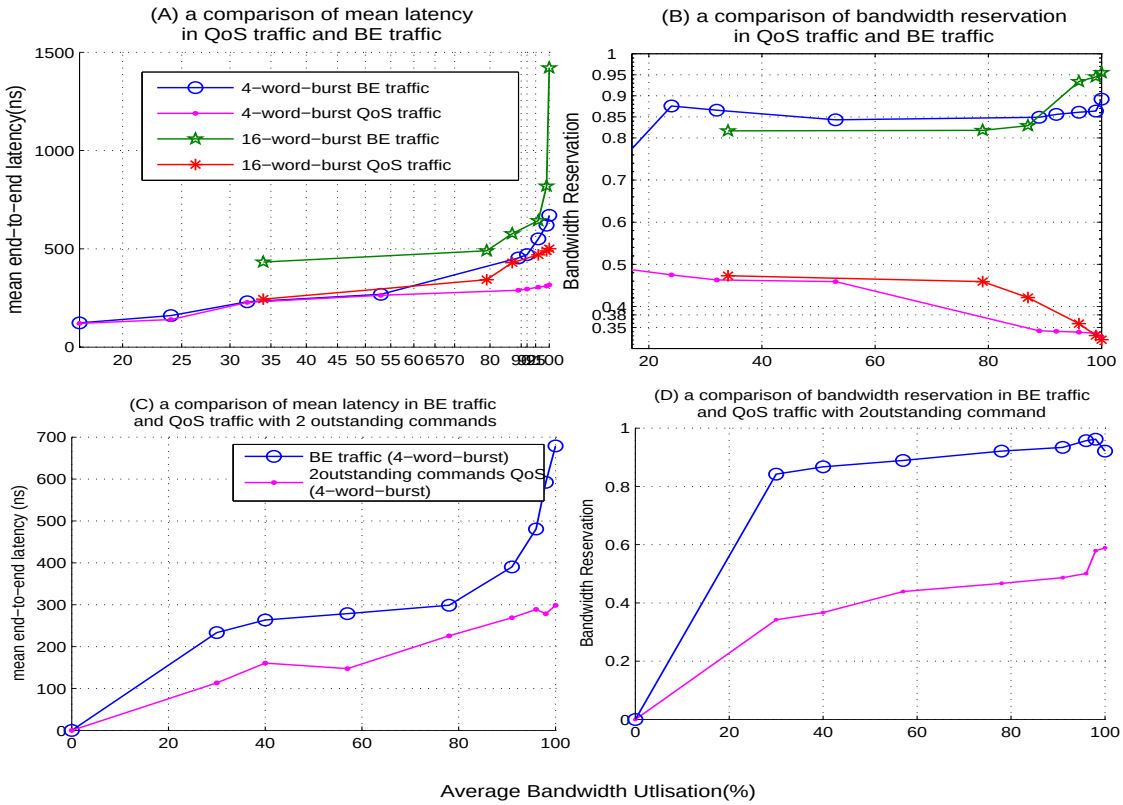

Figure 6. Mean end-to-end latency and bandwidth reservation vs bandwidth utilisation

shown in figure 6 (A), where the latency of QoS traffic using 16-word bursts is more than that of traffic using 4-word bursts. This is one of the reasons why the TMAC scheme only provides a soft latency guarantee.

Figure 6 (C) illustrates the situation when initiator0 can issue two outstanding commands, which means that before a new transaction starts, two pending commands can be issued by initiator0. As shown in figures 6(C&D), the results are what we expect. Initiator0 can get a bandwidth allocation of approximately 50%. Note that the bandwidth utilisation of the BE traffic shown in the graphs is the total of the other four initiators.

6. Conclusions

In this paper, we present a token-managed admission control for QoS support on our SpiNNaker platform. The TMAC is configurable to support specific QoS traffic requirements, and QoS is guaranteed by preferentially reserving tokens. TMAC combines round robin arbitration and priority schemes, and is fair and deterministic. Our experiments evaluate the performance of this low-cost QoS support mechanism and demonstrate the effectiveness of this novel strategy.

TMAC is a significant contribution to end-to-end QoS support. One contribution is its low area overhead. Compared with hard QoS support with buffers-in-switch or virtual channels, the total cell size of our system is very small. Another contribution is the flexibility of bandwidth allocation, which means that we can manage the percentage of bandwidth reservation by allocating different numbers of tokens.

This strategy is valid for all packet-switched interconnects, where it is always possible to provide dedicated links for specific connections that require hard performance guarantees.

A future extension to this analysis might involve a fairness study on other SoC interconnects. We expect that the investigation of non-buffered switches such as wormhole-routing mechanisms will also be interesting.

Acknowledgements

The authors would like to acknowledge the support for this work of the Engineering and Physical Sciences Research Council, partly through the Advanced Processor Technologies Portfolio Partnership at the University of Manchester, and of ARM and Silistix. Steve Furber holds a Royal Society-Wolfson Research Merit Award. Shufan Yang would gratefully like to acknowledge the grants from the China Scholarship Council and to thank the School of Computer& Communication, Hunan University, China, for ongoing support.

References

[1] D.M. Chapiro. *Globally Asynchronous Locally Synchronous Systems*. PhD thesis, Stanford University, 1984.

[2] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1), 2006.

[3] E. Rijpkema, K. G. W. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip. In Rudy Lauwereins and Jan Madsen, editors, *Design Automation, and Test in Europe. The Most Influential Papers of 10 Years DATE*, Circuits & Systems, chapter 2 (Networks on Chip). Springer, January 2008.

[4] Luis A. Plana, Steve B. Furber, Steve Temple, Mukaram Khan, Yebin Shi, Jian Wu, and Shufan Yang. A GALS infrastructure for a massively parallel multiprocessor. *IEEE Design & Test of Computers*, 24(5):454–463, Sept. 2007.

[5] Tomaz Felicijan. *Quality-of-Service (QoS) for Asynchronous On-Chip Networks*. PhD thesis, University of Manchester, School of Computer Science, 2004.

[6] Tobias Bjerregaard and Jens Sparsø. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *ASYNC '05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 34–43, Washington, DC, USA, 2005. IEEE Computer Society.

[7] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Efficient link capacity and QoS design for network-on-chip. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 9–14, Mar 2006.

[8] V. Nollet, T. Marescaux, D. Verkest, J-Y. Mignolet, and S. Vernalde. Operating system controlled network-on-chip. In *Proceedings of Design Automation Conference (DAC)*, pages 256–259, San Diego, June 2004. ISBN 1-58113-916-0.

[9] Prabhat Avasare, Vincent Nollet, Jean-Yves Mignolet, Diederik Verkest, and Henk Corporaal. Centralized end-to-end flow control in a best-effort network-on-chip. In *EMSOFT*, pages 17–20, 2005.

[10] Steve B. Furber and Steve Temple. Neural systems engineering. *Journal of The Royal Society Interface*, 4(13):193–206, 2007.

[11] John Bainbridge and Steve B. Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[12] Silistix Ltd. Technical report, http://www.silistix.com, 2006.

[13] Steve B. Furber. Future trends in SoC interconnect. In *Proc. IEEE International Symposium on Design, Automation and Test (VLSI-TSA-DAT)*, pages 290–293, Hsinchu,TaiWan, 2005.

[14] ARM Ltd. AMBA specification. Technical report, http://www.arm.com, 2006.

[15] Praveen Vellanki, Nilanjan Banerjee, and Karam S. Chatha. Quality-of-service and error control techniques for mesh-based network-on-chip architectures. *Integration*, 38(3):353–382, 2005.

[16] Ashley Saulsbury, Fong Pong, and Andreas Nowatzyk. Missing the memory wall: the case for processor/memory integration. In *ISCA '96: Proceedings of the 23rd annual international symposium on Computer architecture*, pages 90–101, New York, NY, USA, 1996. ACM.

[17] ARM Ltd. PL340 technical report. Technical report, http://www.arm.com, 2007.

[18] Steve B. Furber, Steve Temple, and Andrew Brown. On-chip and inter-chip networks for modelling large-scale neural systems. In *International Symposium on Circuits and Systems, ISCAS-2006*, Kos, Greece, 21-24 May 2006.

[19] Alexander D. Rast, Shufan Yang, Mukaram Khan, and Steve B. Furber. Virtual synaptic interconnect using an asynchronous network-on-chip. In *Proc. 2008 Int'l Joint Conf. on Neural Networks (IJCNN2008)*, Jun 2008.