Fault Tolerance Overhead in Network-on-Chip Flow Control Schemes

Antonio Pullini
DEIS, Università di Bologna
Bologna 40136, Italy
antonio.pullini@studio.unibo.it

Federico Angiolini
DEIS, Università di Bologna
Bologna 40136, Italy
fangiolini@deis.unibo.it

Davide Bertozzi
Università di Ferrara
Ferrara 44100, Italy
dbertozzi@ing.unife.it

Luca Benini
DEIS, Università di Bologna
Bologna 40136, Italy
lbenini@deis.unibo.it

ABSTRACT

Flow control mechanisms in Network-on-Chip (NoC) architectures are critical for fast packet propagation across the network and for low idling of network resources. Buffer management and allocation are fundamental tasks of each flow control scheme. Buffered flow control is the focus of this work. We consider alternative schemes (STALL/GO, T-Error, ACK/NACK) for buffer and channel bandwidth allocation in presence of pipelined switch-to-switch links. These protocols provide varying degrees of fault tolerance support, resulting in different area and power tradeoffs. Our analysis is aimed at determining the overhead of such support when running in error-free environments, which are the typical operating mode. Implementation in the ×pipes NoC architecture and functional simulation by means of a virtual platform allowed us to capture application perceived performance, thus providing guidelines for NoC designers.

Categories and Subject Descriptors

B.4.5 [**Input/Output And Data Communications**]: Reliability, Testing, and Fault-Tolerance; B.4.4 [**Input/Output And Data Communications**]: Performance Analysis and Design Aids

General Terms

Performance, Design, Reliability, Experimentation

Keywords

Network on Chip, Flow Control, Error Correction, Fault Tolerance

1. INTRODUCTION

The high level of system integration characterizing Multi-Processor Systems-on-Chip (MPSoCs) is raising the scalability issue for communication architectures. In this direction, traditional system interconnects based on shared buses are evolving both from the protocol and the topology viewpoint. Advanced bus protocols aim at a better exploitation of available bandwidth, while more parallel topologies are instead being introduced in order to provide more bandwidth [17]. In the long run, many researchers and SoC designers agree on the fact that this trend converges to the Network-on-Chip (NoC) solution, *i.e.* to a modular architecture where scalability is a key design objective [14].

A NoC is instantiated with a mix of just three components: switches, Network Interfaces (NIs) and links. The switches can be arbitrarily connected to each other and to NIs, based on a specified topology. They include routing, switching and flow control logic, as well as error control circuitry. NIs decouple communication from computation, are responsible for packetization/depacketization and implement the service levels associated with each transaction. The ability to hide network implementation details from the connected cores is a valuable feature of NIs, and standard interfaces allowing reuse of IP modules are typically used, such as OCP [18] and AXI [1]. Repeater insertion is a well-known technique to address the problem of high-frequency transmission over data wires [13]. However, even with repeaters, wire delay can exceed one clock period and multiple clock cycles will be needed to transfer data from one synchronous block to another. By inserting latches to improve throughput, wire pipelining techniques deal with this concern [21].

Design features of NoC components affect key performance metrics of the communication fabric, such as latency and throughput. While bandwidth is exceedingly available in NoCs, a lot of effort is being devoted to cutting down on network latency, which might violate performance specifications of applications. Under heavy traffic conditions, fast packet propagation across the network and low idling of network resources are key requirements for efficient communication architectures. *Flow control* determines how network resources are allocated to packets traversing the network, and can be seen either as a problem of resource allocation or one of contention resolution [22].

In circuit-switched NoCs providing Quality of Service (QoS) guarantees, minimum buffering flow control can be used: a circuit is formed from source to destination nodes by means of resource reservation, over which data propagation occurs in a contention free regime. For packet-switched best effort networks, typically buffering increases efficiency of flow control mechanisms. The amount of buffering resources in the network depends on the target performance and on the implemented switching technique. Switches need to hold entire packets when store-and-forward or virtual-cut-through switching are chosen, but only packet chunks (called *flits*, flow control units) when wormhole switching is used.

In some proposed NoC architectures, flow control is combined with error control in a unified mechanism. Error control is becoming a growing concern as technology scales toward deep submicron, because of the increased impact on signal reliability of noise sources such as crosstalk, power-supply noise, EMI and soft errors. Corrupted flits can be detected either in hardware by means of error correction or error detection/retransmission mechanisms, or can be handled at higher network layers (*e.g.*, connection oriented transport layer). However, fast error recovery requires a hardware implementation of error control, thus increasing switch and/or NI complexity. The re-use of flow control mechanisms for error handling allows to save some area and power and to avoid duplication of control lines.

This work focuses on the tradeoffs implied in providing different levels of fault tolerance support in buffered architectures. We consider different mechanisms for buffer and channel bandwidth allocation in the ×pipes NoC architecture [6]. The different flow control techniques account for different buffering location and management strategies and have been adapted to work with pipelined links, where data introduction rate can be decoupled from link length.

The selected schemes provide increasing levels of fault tolerance support, and this is reflected into the tradeoffs they span between area, power and error control capability. However, we do not consider error events for on-chip communication to be so frequent to violate performance specifications of applications. Therefore, we are interested in pointing out whether the overhead for implementing combined flow and error control in hardware is such to degrade application perceived performance in the normal error-free operating mode. Such enhanced flow control schemes run the risk of efficiently recovering from infrequent errors at the cost of a significant performance penalty in the typical fault-free regime. Quantifying such a cost is another objective of this work.

Experiments have been performed by means of extensive simulation runs on a highly accurate virtual platform, capturing the performance of flow control schemes in presence of different levels of system integration. This allowed us to assess the congestion recovery capability of the considered schemes, and to check its dependency on implemented error detection features.

The work is structured as follows. Section 2 discusses flow control policies adopted by previous literature. The flow control protocols under test are analyzed in Section 3. Section 4 shows experimental results that we achieved, and conclusions are presented in Section 5.

2. PREVIOUS WORK

The simplest flow control mechanisms are bufferless. Memoryless switches are employed in [9]: in case of congestion, packets are emitted in a non-ideal direction, also called deflective routing. The introduction of guaranteed bandwidth in [16] was made possible by loop containers and temporally disjoint networks.

Providing QoS by establishing circuits between communicating nodes requires some buffering resources. A novel hybrid circuit switching with packet based setup is reported in [8], which needs minimum buffering resources, capable of holding just a request packet. A circuit switched NoC using time division multiplexing is reported in [12].

In buffered flow control, NoC performance is tightly related to the amount of buffering resources implemented. A methodology to size the FIFOs in an interconnect channel containing one or more FIFOs in series as a function of system parameters (data production and consumption rate, burstiness, etc.) is reported in [21], pointing out the impact on performance.

Credit based flow control was described in [20], for use in ATM networks, and in [22] for use in interconnection networks. It is applied on a hop-by-hop basis. The upstream node keeps a count of the number of free flit buffers in each virtual channel downstream. Credit based flow control is used in [5, 2]. It is also employed in the asynchronous multi-service level QNoC router in [7].

Flow control in SoCIN NoC architecture is based on the handshake concept [3]. When a sender puts data on the link, it activates the related VALID signal. When the receiver is ready to consume the validated data, it activates the corresponding ACK signal. Both handshake and credit based flow control are supported in the revised SoCIN architecture called ParIs [4].

The router in [10] handles both best effort (BE) and guaranteed throughput (GT) traffic. The GT router relies on a time division multiplexing mechanism. Slot tables in the routers divide up bandwidth per link and switch data to the correct output at each time slot. Credit based flow control is used in the BE router at the link level, but also for end-to-end flow control in the network interfaces [2].

The link control mechanism of NoCGEN uses a request, grant and ready handshake to enable flow control on point-to-point links [11].

The ×pipes NoC makes use of the ACK/NACK flow control scheme, adapted to pipelined links [6]. Multiple flits are transmitted from the source switch before the ACK/NACK of the first oustanding flit is received. In case of ACK, the flit is discarded from the source buffer, otherwise it is retransmitted.

T-Error [19] is a flow control protocol aimed at gaining performance by aggressively tackling timing constraints. Links are either overclocked or spatially stretched. As a result, some transmission errors are caused by timing violations. By sampling data on a delayed clock signal, faults can be detected and corrected. The authors report a frequency boost of around 50% while introducing a much smaller correction overhead.

Finally, ON/OFF flow control can greatly reduce the amount of upstream signaling ([22]). The upstream internal state is a single control bit that represents whether the node is permitted to send (ON) or not (OFF). A feedback signal is sent upstream only when it is necessary to change this state, for instance when the number of free downstream buffers falls below a certain threshold.

3. FLOW CONTROL PROTOCOLS

In this paper, analysis will revolve around three flow control protocols, namely STALL/GO, T-Error and ACK/NACK. Each of these offers different fault tolerance features at different performance/power/area points, as sketched in Table 1. STALL/GO is a low-overhead scheme which assumes reliable flit delivery. T-Error is much more complex, and provides logic to detect timing errors in data transmission; this support is however only partial, and usually exploited to improve performance rather than to add reliability. Finally, ACK/NACK is designed to support thorough fault detection and handling by means of retransmissions.

	STALL/GO	T-Error	ACK/NACK
Buffer area	$2N + 2$	$> 3M + 2$	$3N + k$
Logic area	low	high	medium
Performance	good	good	depends
Power (est.)	low	medium/high	high
Fault tolerance	unavailable	partial	supported

Table 1: Flow control protocols at a glance

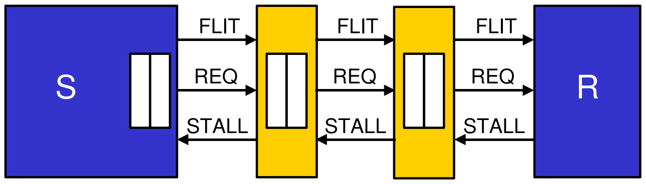


Figure 1: STALL/GO protocol implementation

We implemented each of these flow control protocols to support links having a variable physical length. Clock frequency was kept invaried by pipelining the links with repeater stages, trading off latency for clock speed.

3.1 STALL/GO Protocol

STALL/GO is a very simple realization of an ON/OFF flow control protocol (Fig. 1). It requires just two control wires: one going forward and flagging data availability, and one going backward and signaling either a condition of buffers filled ("STALL") or of buffers free ("GO"). STALL/GO can be implemented with distributed buffering along the link; namely, every repeater can be designed as a very simple two-stage FIFO. The sender only needs two buffers to cope with stalls in the very first link repeater, thus resulting in an overall buffer requirement of $2N + 2$ registers, with minimal control logic. Power is minimized since any congestion issue simply results in no unneeded transitions over the data wires. Performance is also good, since the maximum sustained throughput in absence of congestion is of one flit per cycle by design, and recovery from congestion is instantaneous (stalled flits get queued along the link towards the receiver, ready for flow resumption).

In the NoC domain with pipelined links, STALL/GO indirectly reflects the performance of credit-based policies, since they exhibit equivalent behaviour.

The main drawback of STALL/GO is that no provision whatsoever is available for fault handling. Should any flit get corrupted, some complex higher-level protocol must be triggered.

3.2 T-Error Protocol

The T-Error protocol (Fig. 2, [19]) aggressively deals with communication over physical links, either stretching the distance among repeaters or increasing the operating frequency with respect to a conventional design. As a result, timing errors become likely on the link. Faults are handled by a repeater architecture leveraging upon a second delayed clock to resample input data, to detect any inconsistency and to emit a VALID control signal (Fig. 3). If the surrounding logic is to be kept unchanged, as we assume in this paper, a resynchronization stage must be added between the end of the link and the receiving switch. This logic handles the offset among the original and the delayed clocks, thus realigning the timing of DATA and VALID wires; this incurs a one-cycle latency penalty.

The timing budget provided by the T-Error architecture can also

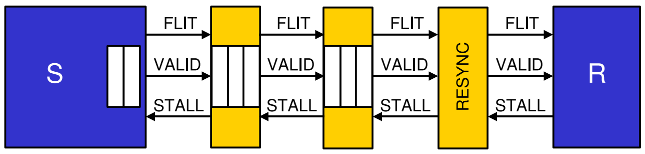


Figure 2: T-Error protocol implementation

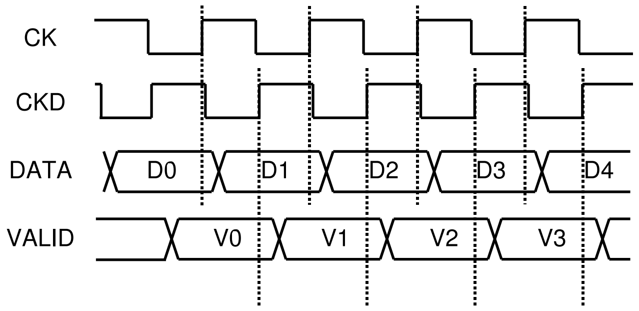


Figure 3: T-Error concept waveforms

be exploited to achieve greater system reliability, by configuring the links with spacing and frequency as conservative as in traditional protocols. However, T-Error lacks a really thorough fault handling: for example, errors with large time constants would not be detected. Mission-critical systems, or systems in noisy environments, may need to rely on higher-level fault correction protocols.

The area requirements of T-Error include three buffers in each repeater and two at the sender, plus the receiver device and quite a bit of overhead in control logic. A conservative estimate of the resulting area is $3M + 2$, with M being up to 50% lower than N if T-Error features are used to stretch the link spacing. Unnecessary flit retransmissions upon congestion are avoided, but a power overhead is still present due to the control logic. Performance is of course dependent on the amount of self-induced errors and will be analyzed in detail in Section 4.

3.3 ACK/NACK Protocol

The main idea behind the ACK/NACK flow control protocol (Fig. 4, [6]) is that transmission errors may happen during a transaction. For this reason, while flits are sent on a link, a copy is kept locally in a buffer at the sender. When flits are received, either an ACKnowledge (ACK) or Not ACKnowledge (NACK) is sent back. Upon receipt of an ACK, the sender deletes the local copy of the flit; upon receipt of a NACK, the sender rewinds its output queue and starts resending flits starting from the corrupted one, with a GO-BACK-N policy. This means that any other flit possibly in flight in the time window among the sending of the corrupted flit and its resending will be discarded and resent. Other retransmission policies are feasible, but they exhibit higher logic complexity. ACK/NACK can either be implemented as end-to-end over a whole fabric, or as switch-to-switch; due to complex issues with possible flit misrouting upon faults in packet headers, we implemented the latter. Fault tolerance is built in by design, provided encoders and decoders for error control codes are implemented at the source and destination respectively.

In an ACK/NACK flow control, a sustained throughput of one flit per cycle can be achieved, provided enough buffering. Repeaters on the link can be simple registers, while, with N repeaters, $2N + k$

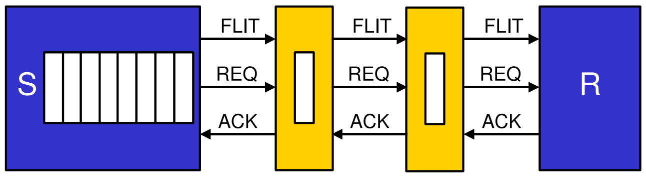


Figure 4: ACK/NACK protocol implementation

buffers are required at the source to guarantee maximum throughput, since ACK/NACK feedback at the sender is only sampled after a round-trip delay since the original flit injection. The value of k depends on the latency of the logic at the sending and receiving ends. Overall, the minimum buffer requirement to avoid incurring bandwidth penalties in a NACK-free environment is therefore $3N + k$.

ACK/NACK provides ideal throughput and latency until no NACKs are issued. If NACKs were only due to sporadic errors, the impact on performance would be negligible. However, if NACKs have to be issued also upon congestion events, the round-trip delay in the notification causes a performance hit which is very pronounced especially with long pipelined links. This will be investigated in Section 4. Moreover, flit bouncing between sender and receiver devices causes a waste of power.

4. EXPERIMENTAL RESULTS

Fault tolerance is an ever more important feature as deep submicron lithographic processes get deployed, to counter increasingly prevalent noise sources. In this paper, the assumption is made that with a conservatively clocked design, errors can be made rare enough to have a negligible performance impact. For this reason, our benchmarking will assume an environment free from external faults and will instead explore performance during normal operation.

This holds also for T-Error. When used to increase system reliability, by deploying it with conservative link parameters, we will simulate fault-free communication. When used to aggressively space link repeaters, thus artificially causing and handling a non-trivial amount of data corruption in exchange for better performance, we will instead inject varying amounts of random transmission errors. These however attempt to reproduce self-induced corruption only, and not external faults.

To explore the performance of alternative flow control protocols, we implemented them on top of the MPARM [15] platform and the ×pipes [6] interconnect. MPARM is a cycle-accurate multiprocessor simulation environment, and allows instantiation of variable amounts of IP cores and flexible memory hierarchies. ×pipes is a configurable NoC, whose topology can be customized, and is based upon wormhole flit switching with output buffering. Routing is source-based and best-effort, *i.e.* no QoS provisions are made. The native flow control policy of ×pipes is ACK/NACK, to support error handling; we extended this support, as outlined in Section 3.

We chose a star-like topology, where up to eight clusters of three processors and their private memories can be instantiated. At the heart of each cluster is a 7x7 port ×pipes switch. Therefore, up to 24 processors can be deployed in the platform. Shared slaves exist and are attached to a central 11x11 switch (see Fig. 5). The central switch is connected to the computation clusters by means of ×pipes pipelined links, whose length can be customized. Depending on the amount of instantiated processors, the simultaneous traffic on the

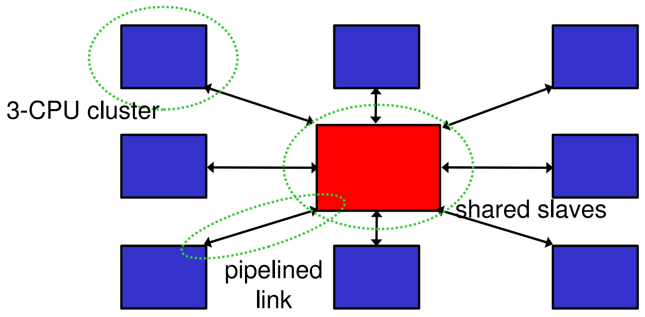


Figure 5: The star-like topology under test

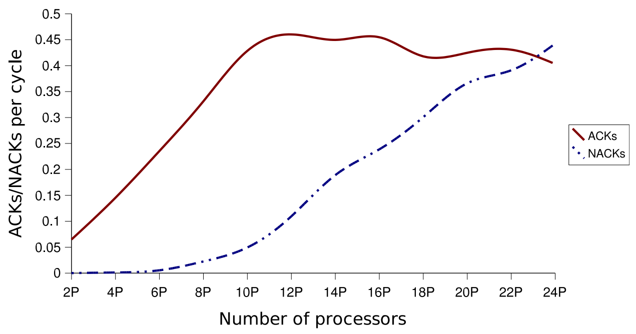


Figure 6: Link congestion under increasing traffic

links towards this central switch increases, resulting in congestion. All of the processors are executing the same benchmark, which encompasses local computation (which happens within the peripheral clusters) and performs communication and synchronization functions by accessing the shared slaves on the central switch.

An analysis of the link congestion trend under increasing pressure by IP cores can be found in Fig. 6, which plots the density of ACKs and NACKs (ACKs and NACKs over clock cycles) when using ACK/NACK flow control in the system. The chart is only plotting figures for the links connecting the central switch to the clusters; the links are here assumed to be very long, with six repeaters. As can be seen, with more processors, initially the ACK density increases thanks to the increase in offered bandwidth. Just before hitting the value 0.5 (one ACK every two cycles), growing congestion and the intrinsic inefficiency of this flow control protocol impose a bandwidth ceiling, while a growing amount of NACKs can be observed.

To evaluate the efficiency of the three flow control protocols, we measured the average latency for communication transactions across the congested links and the overall benchmark execution time. Please note that clock frequency and physical layout were assumed to be the same for all schemes. Results are plotted as a function of the length of the pipelined links in Fig. 7. For T-Error, the same simulations were carried out in two scenarios. The first accounts for 50% less repeater stages (four instead of six, two instead of three) and assumes a 5% error rate. The second scenario is more conservative, with as many repeaters as in other protocols and no errors. In both cases, a resynchronization stage is needed before the receiving switch. In the direct connection scenario, where it is assumed that operation over the links can be safely achieved in a single clock cycle, T-Error logic is unneeded and this case reduces to the direct STALL/GO connection. Under all circumstances, both variants of T-Error and STALL/GO exhibit similar performance. The latency advantage gained over STALL/GO by the aggressive deployment of T-Error is mostly offset by the need for a resynchronization stage and by some penalty upon transmission errors; for short links and in low congestion environments, T-Error can even perform worse. The conservative T-Error links perform almost on par with the aggressive ones because they trade repeater stages for error-free operation. The conservative T-Error links always perform slightly worse than the corresponding STALL/GO schemes due to resynchronizer latency, but the transaction overhead is negligible. As expected, if links are very long (six repeaters: Fig. 7(a)), the round trip delay imposed by the ACK/NACK protocol proves to be a major drawback, and latencies increase steeply with congestion. With shorter links (three repeaters: Fig. 7(b)), the ACK/NACK overhead decreases, and substantial performance parity is achieved if the switches are directly attached (Fig. 7(c)).

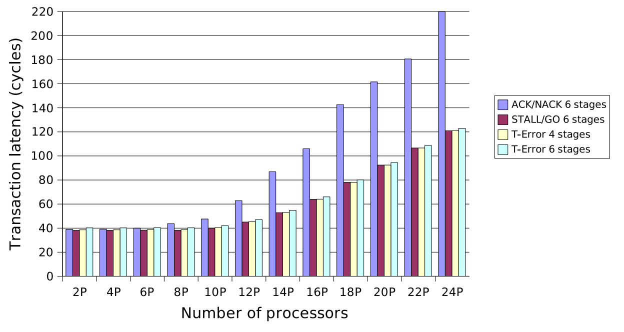
In Fig. 8, the overall benchmark execution time is reported just for the longest link scenario. The trend is of course similar to that of the communication latency over the congested links, but differences are smaller because they are masked by the time spent in local computation.

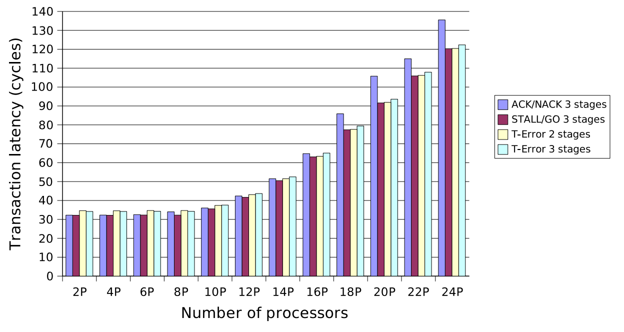
The aggressive T-Error variant achieves lower communication latencies by accepting a certain amount of transmission errors as a tradeoff. Determining this amount is not a focus of this work. So, in Fig. 9 we explored the design space by assuming different self-induced error probabilities. The plot is reporting figures for a long link, which is assumed to have a 0% to 27% error rate percentage. This number expresses the percentage of errors per clock cycle (not per transmitted flit), and is for the whole link. Latencies are normalized against the ideal error-free case. As the plot shows, under heavy congestion, T-Error is by design able to almost completely mask errors, because error penalties can be hidden behind congestion-triggered stalls. Under light traffic, transmission faults have a more noticeable impact, with 6% worse transmission latency when comparing a 27% link error probability against the ideal case. Still, T-Error is very good at minimizing the impact of faults on performance.

5. CONCLUSIONS

As expected, as can be seen in Table 1 and by looking at Section 4, STALL/GO proves to be a low-overhead efficient design choice showing remarkable performance, but unfortunately is fault-sensitive.

T-Error can be either deployed to improve link performance, or to improve system reliability by catching timing errors. In the former design, we observed average latencies on par with STALL/GO, but no error detection capability was present; in the latter case, speed degraded slightly, in exchange for a partial but significant reliability boost. In both alternative schemes, some area and power overhead is incurred. Overall, we believe that using T-Error to decrease the number of pipeline stages does not bring significant performance benefits, while the partial detection capability can be effectively exploited in a conservative design. Another possible option is the conversion of the timing margin budget into a frequency overclock; while this choice holds good potential, we did not explore it in the present paper since it is only feasible if the surrounding NoC components (switches, network interfaces) are designed to work at the same extremely high frequencies.





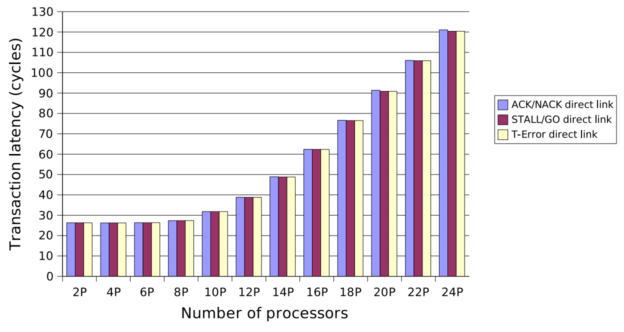


Figure 7: Communication latency over congested links of different lengths

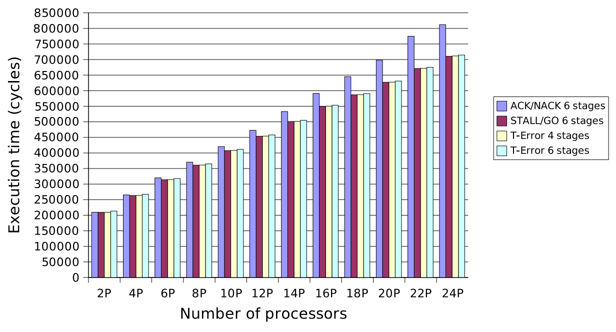


Figure 8: Benchmark execution time under increasing congestion, long links

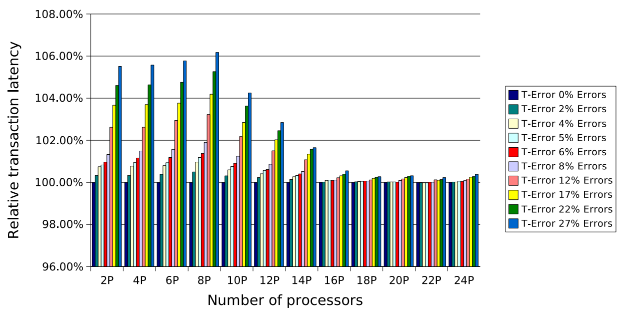


Figure 9: T-Error performance under varying error probabilities

ACK/NACK pays its most extensive fault handling support with signifi cant power and area overheads. Performance penalties were also noticed in presence of heavy congestion and long pipelined links. However, we expect that, in current and imminent design technologies, links will need no more than three repeaters, the very long link scenario being representative of a more distant future. In low-congestion or short-link scenarios, the application-perceived latency overhead of ACK/NACK turned out to be negligible.

6. REFERENCES

[1] www.arm.com/products/solutions/AMBAAXI.html.

[2] A.Radulescu, J.Dielissen, S.G.Pestana, O.P.Gangwal, E.Rijpkema, P.Wielage, and K.Goossens. An effi cient on-chip ni offering guaranteed services, shared-memory abstraction, and flexible network confi guration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(1):4 – 17, 2005.

[3] C.A.Zeferino and A.A.Susin. SoCIN: a parametric and scalable network-on-chip. In *Symp. on Integrated Circuits and Systems Design*, pages 169 – 174, September 2003.

[4] C.A.Zeferino, F.G.M.E.Santo, and A.A.Susin. Paris: a parameterizable interconnect switch for networks-on-chip. In *Symp. on Integrated Circuits and Systems Design*, pages 204 – 209, September 2004.

[5] C.A.Zeferino, M.E.Kreutz, L.Carro, and A.A.Susin. A study on communication issues for systems-on-chip. In *Symp. on Integrated Circuits and Systems Design*, pages 121 – 126, September 2002.

[6] D.Bertozzi and L.Benini. ×pipes: a network on chip architecture for gigascale systems-on-chip. *IEEE Circuits and Systems Magazine*, 4(2):18 – 31, 2004.

[7] D.Rostislav, V.Vishnyakov, E.Friedman, and R.Ginosar. An asynchronous router for multiple service levels networks. In *Symp. on Integrated Circuits and Systems Design*, pages 204 – 209, March 2005.

[8] D.Wiklund and D.Liu. Socbus: switched network on chip for hard real time embedded systems. In *Int. Parallel and Distributed Processing Symposium*, page 8pp, April 2003.

[9] E.Nilsson, M.Millberg, J.Oberg, and A.Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *DATE03*, pages 1126 – 1127, 2003.

[10] E.Rijpkema, K.Goossens, A.Radulescu, 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. *IEE Proc. on Computers and Digital Techniques*, 150(5):294–302, 2003.

[11] J.Chan and S.Parameswaran. Nocgen:a template based reuse methodology for networks on chip architecture. In *Int. Conf. on VLSI Design*, pages 717 – 720, 2004.

[12] J.Liu, L. Zheng, and H.Tenhunen. A guaranteed-throughput switch for network-on-chip. In *Int. Symp. on System-on-Chip*, pages 31 – 34, November 2003.

[13] K.Banerjee and A.Mehrotra. A power-optimal repeater insertion methodology for global interconnects in nanometer designs. *IEEE Trans. on Electron Devices*, 49(11):2001 – 2007, 2002.

[14] L.Benini and G. Micheli. Networks on chips: a new soc paradigm. *IEEE Computer*, 35(1):70 – 78, 2002.

[15] M.Loghi, F.Angiolini, D.Bertozzi, L.Benini, and R.Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proceedings of the IEEE Design and Test in Europe Conference (DATE)*, pages 752–757, February 2004.

[16] M.Millberg, E.Nilsson, R.Thid, and A.Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *DATE04*, pages 890 – 895, February 2004.

[17] M.Ruggiero, F.Angiolini, F.Poletti, D.Bertozzi, L.Benini, and R.Zafalon. Scalability analysis of evolving SoC interconnect protocols. In *Int. Symp. on Systems-on-Chip*, November 2004.

[18] www.ocpip.org.

[19] R.Tamhankar, S.Murali, and G. Micheli. Performance driven reliable link for networks on chip. In *ASPDAC - Proceedings of the Asian Pacific Conference on Design Automation*, 2005.

[20] S.Khorsandi and A.L.Garcia. Robust non-probabilistic bounds for delay and throughput in credit-based flow control. In *INFOCOMM*, pages 577 – 584, 1996.

[21] V.Chandra, A.Xu, H.Schmit, and L.Pileggi. An interconnect channel design methodology for high performance integrated circuits. In *DATE04*, pages 1138 – 1143, February 2004.

[22] W.J.Dally and B.Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.