Design of a fast, low-level fault-tolerant protocol for Network on Chips

Muhammad Ali[1], Awais Adnan[2], Michael Welzl[1]
[1]Institute of Computer Science, University of Innsbruck, Austria
[2]Institute of Management Sciences, Peshawar, Pakistan

Abstract: *Network on a chip (NoC) has been proposed to address the inefficiency of buses in the current System on Chips (SoC). However as the chip scales, the probability of errors is also increasing, thus, making fault tolerance a key concern in scaling chips. Transient faults are becoming a major cause of errors in a packet based NoC. A transient error can either corrupt the header or the payload of packet requiring a retransmission from the source. Due to retransmissions, packets arrive out of order at the receiver side. Complex reordering algorithms are required at the receiver side to organize packets before sending them to associated resource. This adds a major overhead as the storage and logic capabilities are limited on a chip. We therefore provide a low-cost, fast and reliable end-to-end protocol for NoC which does not require reordering of the packets at the receiver end. Our protocol performs bitwise logical operations using binary representation of the addresses for the buffers to handle packets, hence making it much faster than conventional algorithms. Furthermore, the protocol is equally applicable to both static as well as dynamic routing environments on a chip.*

Keywords: Fault Tolerance, Network on a chip, Transient / soft errors.

I. INTRODUCTION

The International Technology Roadmap for Semiconductors (ITRS) projects that by the end of this decade chips would accommodate billions of transistors [1]. This means that in near future Application Specific Integrated Circuits (ASIC) will be made up of hundreds of communicating partners. However, it has been observed that the existing on-chip interconnects pose a serious threat toward achieving the billion transistor era. Buses that form an integral part of today's SoCs show serious degradation of performance beyond a certain number of communicating partners [2]. Considering the unmatched success of the Internet especially in terms of scalability, VLSI researchers have borrowed ideas from computer networks and proposed a packet based communication model for SoCs. This new paradigm is termed as *Network on Chips (NoC)* [3]. The idea is to send data in the form of packets over a network of switches/routers on a chip. This way communication and computation are kept transparent from each other, hereby achieving reusability of components and scalability of the whole network on-chip.

The increasing number of transistors on a chip is also contributing toward an increase in the faults, both temporary and permanent. With decreasing die size, cross talk, critical leakage currents and high field effects will increase the probability of permanent and temporary faults on a chip [4]. Permanent faults may cause one or more on-chip links or switches to fail, causing permanent damage to the component. Temporary faults, also known as transient failures or soft errors, do not cause permanent damage but may scramble one or more bits in a packet, hence making it invalid [5].

The frequency of transient errors is much higher than the permanent faults on-chip, making it necessary to provide built-in error recovery mechanism. With increasing transistor density on-chip, the soft error ratio is increasing exponentially [6]. Since today's SoCs are integrated into consumer products, it is important to equip them with some degree of fault tolerance. This can increase the overall yield of the chips by reducing the cost of production.

In this paper we present a fast and reliable end-to-end protocol for NoCs which ensures safe delivery of packets from source to destination. Since the protocol is implemented in the end systems, no complex logic algorithms and excessive storage capability is required at the intermediate routers. The significant aspect of our protocol is its simplicity in implementation on a chip as it uses bitwise logical operations to process and store the packets, eliminating any need for complex reordering and storing algorithms.

The paper is organized as follows; in the next section we discuss issues regarding error tolerance in NoCs along with related work. In section III, we provide a brief discussion of the NoC model we are using. Section IV elaborates the description of our protocol in detail, followed by conclusion and future work.

II. FAULT TOLERANCE IN NOCs

Traditionally, chips are equipped with error detection and correction mechanisms to deal with transient faults. However, packet based communication on-chip brings new challenges in terms of error resilience. The primary unit of packet based communication is a packet itself. A packet is usually composed of a *header* and a *payload*. The header of the packet contains identification information like source and destination addresses, routing path, CRC (Cyclic Redundancy Check) checks etc. The payload contains the actual data to be transported. A soft error can scramble either header or the payload of a packet. A bit flip, due to a soft error, in the header of the packet may cause it to be routed to a wrong destination. An error in the payload, on the other hand, may corrupt the contents of the packet, hence making it invalid although it might reach the destination. In both cases a retransmission of the misrouted or corrupt packet is required.

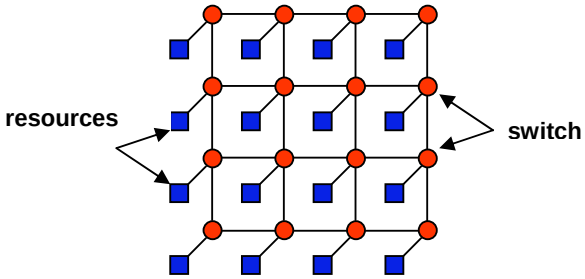

Figure 1: 2D mesh topology for NoCs

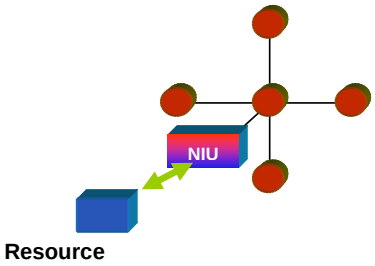

Figure 2: Abstract representation of NI

The authors of [7] have discussed and analyzed various error coding schemes for NoCs. In [8], T. Dumitras et al. present a stochastic communication model for NoCs based upon rumor spreading. In this mechanism a node spreads the packet to its neighbor which, in turn, spreads it to all its neighbors assuming that at least one packet will reach the destination. Though simple, the mechanism has high packet overhead especially when a large number of nodes are communicating. In [9], Bertozzi et al. present a link level flit[1] based retransmission protocol. Each flit is acknowledged by every intermediate router on successful reception. In case a flit is missing or is corrupt, a negative acknowledgement is generated. Such a mechanism adds complexity on part of intermediate routers besides buffering them.

We propose a fast, efficient, and fault tolerant scheme to deal with packet corruption due to transient errors in a NoC. The mechanism does not require intermediate routers to maintain buffers and thus forwards packets as they arrive. Moreover, the mechanism is equally effective in both dynamic and static routing environments of NoCs. The detailed description of the proposed protocol is discussed in section IV.

III. NOC MODEL

Various topologies are possible for NoCs like honeycomb [10], fat-tree [11], 2D mesh [12] etc. The choice of a topology can dramatically affect the network characteristics of a NoC in terms of number of hops, link delays etc. Although we intend to study the behavior of our mechanism with other topologies in future work, in this paper, we only discuss the most agreed upon topology for the sake of simplicity --- a *2D mesh*. A typical topology model is shown in figure 1 followed by its description:

Boxes represent resources in a NoC which is an end system having data to send or receive. Each resource in this scenario is connected to a router. They are considered black boxes as they can be anything: a MIPS[2] processor, a DSP[3], a memory module or even a combination of any or all of these elements. Hence in terms of a set of resources, it can be termed as *network within a network.*

Circles in figure 1 represent routers which are responsible for establishing a routing path between sender and receiver. Each router is directly connected to neighboring four routers (except for the ones at the edges), thus creating a mesh network.

Network Interface (NI) is a special device acting as a middle layer placed between the resource and the router. NI is responsible for creating packets from bit streams obtained from the connected resource and vice versa. It is a very significant unit of the NoC as it covers the discrepancies that exist between the homogeneous router network and different kind of resources emanating from a variety of vendors. NI makes it possible to reuse not only the resources but also the network architecture of a NoC. A typical example of a NI is shown in figure- 2.

IV. DESCRIPTION OF THE PROTOCOL

As mentioned earlier, we propose an end-to-end reliable packet delivery mechanism, restricting the intricacies of the protocol to the sender and receiver nodes. The sender sends a pre-defined number of packets to the receiver and after making sure their safe delivery, sends the next set. If there are X packets in a set, it means the sender and receiver need X buffers to store them. In our example setup, we take it as 16, which means the sender and receiver have to buffer 16 packets at a single time. Each packet in a set is identified by a unique packet ID from 0 to 15 (*0000 -1111 at lower four bits of address*). The receiver associates a flag value with each buffer which is initially *zero* . When a packet arrives at the receiver, the error detection code checks it for any inconsistencies (scrambled bits, for example). If a packet arrives with incorrect payload, it is dropped. The receiver sets the flag for each received packet as 1 and otherwise for packets not received.

Consider figure 4 which is an abstract representation of block diagram at the receiver side. The packet ID is first ANDed with equal number of bits where except for first 4 bits all the rest are zeros. This way, we can retrieve the first four bits of the packet ID which is unique for each packet in a set[4]. The packet number P is converted into 2^P by a temporary register, ***T,*** which is done by shifting 1 *p* times to the left. For instance, if packet 5 arrives (which has packet ID 4

[1] Packet divided into equal sized smaller units called *flits*
[2] *M*icroprocessor without *I*nterlocked *P*ipeline *S*tages
[3] *D*igital *S*ignal *P*rocessor

[4] Assuming that each set is of 16 packets, each packet in a set will have unique ID.

Table 1: showing packet numbers, their binary equivalent and corresponding 16 bit register values

Packet ID (p)	Packet ID (4 bits)	***T***= 2 ^ p	In 16 bits
0	0000	1	00000000 00000001
1	0001	2	00000000 00000010
2	0010	4	00000000 00000100
3	0011	8	00000000 00001000
4	0100	16	00000000 00010000
5	0101	32	00000000 00100000
6	0110	64	00000000 01000000
7	0111	128	00000000 10000000
8	1000	256	00000001 00000000
9	1001	512	00000010 00000000
10	1010	1024	00000100 00000000
11	1011	2048	00001000 00000000
12	1100	4096	00010000 00000000
13	1101	8192	00100000 00000000
14	1110	16384	01000000 00000000
15	1111	32768	10000000 00000000

since the count is from 0 to 15), then 2^4 gives 16 (10000) which is achieved by shifting 1 five times to the left of 16 bit register yielding 0000 0000 0001 0000. Similarly, for every different packet 1 has a uniquely significant position in 16 bits register. Table 1 shows the values for ***T*** and 16 bit register for all 16 packets in a single set. In order to detect any duplication in received packets, the temporary register ***T*** is first NANDed with the 16 bit register. If it yields a zero, it means the packet is a duplicate and is dropped, otherwise, the packet is accepted. These logic operations are primitive in nature and can be implemented simply on a chip. Figure 4 is an abstract illustration where in actual they are a group of 16 similar gates one for each bit.

To store packets directly in their respective locations, we consider the buffer as a continuous storage of size $Z=N*K$, where N is the number of packets (16 in our example), and K is the size of each packet (say 32 bits)[5]. Logically speaking, the buffer is divided into N sub-buffers each of size K. The starting address of each packet can be calculated by multiplying first four bits of its unique packet ID to the packet size. This can be achieved by shifting 5 bits left of the 4-bit packet ID. For example, to find out the location for 5^{th} packet in the buffer, we multiply its packet ID (4) with packet size 32, which yields 128 = 010000000. This binary value is obtained by shifting 5 bits to the left of packet ID

A. Retransmission policy:

The next important stage is for the receiver to let the sender know if it received all the packets or not. In case it received all the packets, it has to acknowledge the sender so that the sender can send the next set of packets. In case one or more packets were dropped, they need to be retransmitted before the buffers are relinquished. In the given scenario, there could be two methods to request for retransmissions which are based upon the routing strategy implemented.

A.1. Static routing

In static routing, paths are fixed and communication is carried out following the shortest path. One example of static routing is XY based routing where a packet is first routed in the X direction and when it comes in the column of the destination, it moves in the Y direction until it reaches the destination [12]. Since deterministic routing always takes the shortest possible path, it is likely that all the packets follow a particular path. In such a case, all the packets are supposed to reach in order at the receiver side. When a corrupt packet reaches the receiver, the receiver drops that packet and generates a negative acknowledgement for the retransmission of the packet. In such a situation, using our mechanism, the receiver will receive the packets as they come in and will put them in their respective buffers. Every time a packet is dropped either by the receiver or at some intermediate router, the receiver issues a negative acknowledgment to the sender. The sender, on receiving the negative acknowledgement resends the requested packet. Since our protocol uses bitwise addressing with shift registers to place the packet in their exact locations, no reordering is required. Once all the packets are received successfully, the receiver sends a positive acknowledgment upon which the sender relinquishes the buffers and fills them up with the next set of data. In case that a requested packet does not arrive, the receiver issues a second negative acknowledgement after a certain timeout[6].

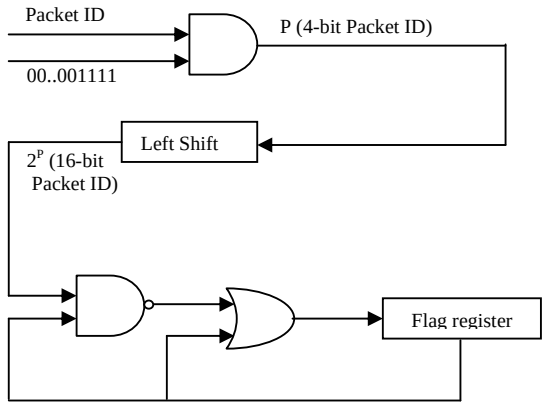

Figure 4: A block diagram for receiver

A.2. Dynamic routing

Dynamic routing, as the name shows, allows to change the routing path of packets in a NoC automatically in case a router fails or congestion occurs in the optimal path. In dynamic routing packets may take different paths to reach the destination. In this case, they may not arrive in order at the destination since different paths may have different latencies. Dynamic routing is more flexible in terms of congestion in the links or permanent failures; however, it is more complex in terms of implementation. Each router needs to be equipped with dynamic routing protocol to deal with changing situation in the NoC.

[5] This value is only for simplicity since the actual packet size can range from 32 to 64 bytes.

[6] timeout value depends upon the routing strategy implemented.

The retransmission policy in terms of dynamic routing is as follows; the receiver receives all the packets according to the method explained above. Then after a certain timeout, it scans all the flag registers and for those registers which are still zero are sent to the sender which resends all those packets. Alternatively, the receiver can simply send the flag registers to the sender which then scans them and resends all those packets for which the register value is still high. In either case, the missing packets are delivered to destination.

V. CONCLUSION AND FUTURE WORK

In this paper we have proposed an efficient, fast and reliable packet delivery protocol based in the end systems. The protocol entirely uses bitwise logic making it simple to implement on a chip. Buffers are only kept at the source and destination. Since the protocol uses registers and logical operations to store packets in their respective locations, therefore, no reordering algorithms are required. Considering the limited storage and logic capabilities on chip, our protocol proves to be cost effective besides providing reliability in the presence of soft errors.

We are currently working to implement our reliable protocol for NoCs in Network Simulator NS-2 [13]. The simulation setup is considered for a 4x4 2D mesh based NoC. The die size is assumed to be $22mm^2$ with a link size

Table 2: shows addresses for packet numbers from 0 to 15

Packet ID	4 Bit Packet ID	Packet Size K	Starting Address K_n=n*K	In binary format
0	0000	32	0	0000 00000
1	0001	32	32	0001 00000
. . .	. . .	. . .	. . .	. . .
14	1110	32	448	1110 00000
15	1111	32	480	1111 00000

of 4.5mm each. The capacity of each link is 2Gbits/sec with a link delay of 10 nano-secs. Furthermore, we propose a 4x4 cross bar router architecture. Each output may be connected to at most one input, whereas each input may be connected to as many outputs as possible. A crossbar router is critical in case of dynamic routing where a packet may be forwarded on any output channel due to network load or physical conditions.

At the moment we are only concentrating on unicast[7] routing. Multicasting[8], on the other hand, is a very complex issue and involves many parameters of a NoC to be reconsidered. For instance, if a source is communicating with many destinations, the key issue is about the communication paths; weather the same packet be sent via different paths or via same path especially when the destinations are situated at the same level. Another possibility is to send a single copy of packet until it reaches one router before the first destination. This router then replicates the packet and sends to other attached destinations. Also in a fault tolerant environment, the sender needs to keep track of packet corruption or losses for each individual receiver. All such issues are quite complex in nature but nevertheless worth looking at. We thereby are determined to implement and test various multicasting mechanisms for NoCs in our future work.

[7] single source sending packets to a single receiver
[8] single source communicating sending packets to many receivers

REFERENCES

[1] www.itrs.net

[2] Ahmed Jerraya, Hannu Tenhunen and Wayne Wolf, ``Multiprocessor System-on-chips'', *IEEE Computer magazine*, July 2005, Vol. 38, No. 7

[3] L. Benini and G. De Michelli, `Networks on Chip: A new paradigm for component-based MPSoC design'', *Multiprocessors Systems on Chips*, edited by A. Jerrraya and W. Wolf, Morgan Kaufman, 2004, pp. 49-80.

[4] Ming Shae Wu and Chung Len Lee, ``Using a Periodic Square Wave Test Signal to Detect Cross Talk Faults'', *IEEE Design & Test of Computers*, Volume 22, Issue 2, March-April 2005, Page(s): 160 - 169

[5] Michael L. Bushnell, Vishwani D. Agarwal, ``Essentials of Electronic Testing for Digital, Memory and Mixed-Signal Circuits'', *Kluwer Academic Publishers*, USA, 2000

[6] P. Shivakumar, M. Kistler et al. ``Modeling the effect of technology trends on the soft error rate of combinational logic'', Proc. DSN 2002, pp.389-398

[7] Srinivas Murali, et al. ``Analysis of Error Recovery Schemes for Networks on Chips'', IEEE Design & Test, Vol. 22, Issue 5, Sept. 2005

[8] T. Dumitras, R. Marculescu, ``On-Chip Stochastic Communication'', *Proceedings, DATE 2003*, pp. 790-795

[9] D. Bertozzi et al., ``Xpipes: A Network-on-chip Architecture for Gigascale System-on-chip'', *IEEE Circuits and Systems*, 2nd quarter, 2004, pp. 18-21

[10] Ahmed Hemani, Axel Jantsch, Shashi Kumar, Adam Postula, Johny Oberg, Mikael Millberg, Dan Lindqvist, ``Networks on a chip: An Architecture for billion transistor era'', *Proc. IEEE NorChip Conference*, November 2000

[11] P.Guerrier and A.Greiner ``A generic architecture for on-chip packet switched interconnections´´, *Proceeding Design and test Europe*, March 2000

[12] William J. Dally and Brian Towles, "Route packets, not wires: on-chip interconnection networks, Proceedings, *Design Automation Conference (DAC)*, pp. 684-689, Las Vegas, NV, June 2001

[13] http://www.isi.edu/nsnam/ns/