# On-Chip Stochastic Communication

**by**
**Tudor A. Dumitraş**

A dissertation submitted in partial satisfaction of the requirements for the degree of

Master of Science

in

Electrical and Computer Engineering

Date: May 1st, 2003

Advisor: Prof. Radu Mărculescu
Second Reader: Prof. Priya Narasimhan

Department of Electrical and Computer Engineering

Carnegie Mellon University

Pittsburgh, Pennsylvania, USA

# Abstract

As CMOS technology scales down into the deep-submicron (DSM) domain, the Systems-On-Chip (SoCs) are becoming increasingly complex and the costs of design and verification are rapidly increasing due to the inefficiency of traditional CAD tools. Relaxing the requirement of "100% correctness" for devices and interconnects drastically reduces the costs of design but, at the same time, requires that SoCs be designed with some degree of system-level fault-tolerance. This thesis introduces a novel communication paradigm for SoCs, called *stochastic communication*. The newly proposed scheme not only separates communication from computation, but also provides the required built-in fault-tolerance to DSM failures, is scalable and cheap to implement. For a generic network-on-chip (NoC) architecture, we show how a ubiquitous multimedia application (an MP3 encoder) can be implemented using stochastic communication in an efficient and robust manner. More precisely, by using this communication scheme, up to 70% data upsets, 80% packet losses because of buffer overflow, and severe levels of synchronization failures can be tolerated, while providing a much lower latency than a traditional, bus-based implementation.

The present thesis also introduces a new concept, called *on-chip diversity*, which means mixing different architectures and/or technologies in a multiple voltage/frequency island setup in order to achieve the highest levels of performance, fault-tolerance, and the needed flexibility in SoC design. We outline how stochastic communication can enable the overall integration of such systems, and how it can become the base for several hybrid communication architectures. We believe that the ideas and the results presented here will open up a whole new area of research with deep implications for on-chip network design and the future generations of SoCs.

**Keywords:** System-on-Chip, Network-on-Chip, Fault-Tolerant Communication, High Performance Architectures, Multimedia Systems, Stochastic Communication, On-Chip Diversity

# Table of Contents

# List of Figures

# Acknowledgements

I would like to thank my advisor, Professor Radu Mărculescu, for his constant guidance and encouragement, which have led to the achievement of the research presented in this thesis. Thanks are also due to Sam Kerner, who has developed a simulator for stochastic communication and has imagined many experiments relevant to the topics discussed in Chapter 5, and to Jingcao Hu, for his insightful remarks on the results of our experiments.

Although they are not directly related with the present work, I would like to express my gratitude for my undergraduate advisor, Professor Jean-Marc Steyaert from the Ecole Polytechnique of Paris, who opened my eyes to research and who guided my footsteps towards graduate school, for my father, who has always been a role model for me and who gave me an outstanding example of scientific conduct, and, above all, for my mother, who taught me the importance of intellectual achievement and who, through her endless love and devotion, has helped me live up to all the challenges that I have faced.

# 1 Introduction and Objectives

As the modern VLSI chips are becoming increasingly complex and as the manufacturing technologies are scaling down into the deep-submicron (DSM) domain, the designers are facing several new challenges. Nowadays, application-specific integrated circuits (ASICs) have evolved into complicated systems-on-chip (SoCs), where dozens, and soon hundreds, of predesigned IP cores are assembled together to form large chips with complex functionality. Extensive research on how to integrate and connect these IPs is currently being conducted, but there remain many open issues that are difficult to address within the framework of existing CAD tools.

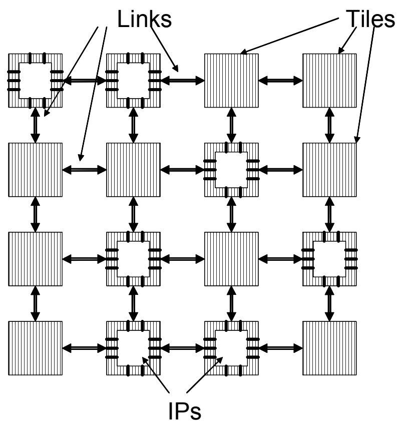
Indeed, shrinking transistor dimensions, smaller interconnect features and higher operating frequencies lead to a higher sensitivity of DSM circuits to neutron and alpha radiation, significantly higher soft-error rates, and an increasing number of timing violations [7]. These new types of failures are impossible to characterize using *deterministic* measurements and, thus, *probabilistic* metrics, such as average values and variances, are likely to be needed to quantify the critical design objectives, such as performance and power [2, 25]. It has become clear that, in order reduce the cost of design and verification, the "100% correctness" requirement for VLSI circuits has to be relaxed [34, 37]; this means that, in the future, circuits will have to be designed with some degree of architectural and system-level fault-tolerance embedded in their structure [3, 10, 40, 13].

Furthermore, it is emphasized in the ITRS (International Technology Roadmap for Semiconductors) 2001 [37] that it is very important, especially at the system level, to separate the *computation* from *communication*, as they are orthogonal and unrelated issues which should remain separate, whenever possible. A novel *communication paradigm* has to be developed in order to enable the separation between the design of the on-chip communication architecture and the design of the SoC's main functionality.

Traditionally, the IP cores that form a SoC are connected with an on-chip shared bus or a hierarchy of buses. Because a bus is a shared communication channel, it requires arbitration in order to ensure the mutual exclusion between the components accessing the channel. One problem arises when a bus needs to connect a large number of modules, as performance decreases drastically because of the contention for access to the shared medium. Therefore, this communication architecture is not well suited to the SoCs of the future which will incorporate hundreds of communicating IPs. On-chip buses will have to be supplemented or even replaced with a more scalable on-chip communication infrastructure [28].

A recently proposed platform for the on-chip interconnects is the network-on-chip (NoC) architecture [9, 22], where the IPs are placed on a rectangular grid of tiles (see Figure 1-1) and the communication between the tiles is implemented by a stack of networking protocols. These *regular structures* are very attractive because they can offer well-controlled electrical parameters, which enable high-performance circuits by reducing the latency and increasing the bandwidth. It has been predicted that, henceforth, SoC design will resemble more the creation of large-scale communication networks than traditional IC design practice [37].

**Figure 1-1.** Network-on-Chip (NoC)



However, defining *communication protocols* for these NoCs does not seem to be an easy matter, as mitigating the effects of on-chip failures on the network communication remains largely an open question. Furthermore, the resources used in traditional networks in order to achieve fault-tolerance are not easily available at the level of VLSI chips. A *static routing* approach involving the transmission of messages along a fixed path from source to destination, would fail if even a single tile or a link on the path is faulty. Generally, the *deterministic* algorithms do not behave very well in the presence of random failures [17, 30]. On the other hand, the cost of implementing adaptive *dynamic routing* for the on-chip networks is prohibitive because of the need for very large buffers, lookup tables and complex shortest-path algorithms [35]. Classic networking protocols, such as the Internet Protocol or the ATM Layer [31], require acknowledgements and *retransmissions* in order to deal with incorrect data and, therefore, cannot meet the strict requirements of low latency that are characteristic of modern SoCs.

## 1.1 Contributions of this thesis

The research presented in this thesis addresses the problem of on-chip fault-tolerant communication. In order to achieve this goal, the present thesis introduces a novel communication paradigm, called *on-chip stochastic communication*. More precisely, in a NoC such as the one in Figure 1-1, the IPs communicate using a probabilistic broadcast scheme, very similar to the *randomized gossip* protocols [11]. If a tile has a message that needs to be transmitted, this will be forwarded to a randomly chosen subset of the tiles in the neighborhood. This way, the messages are *diffused* through the network to all the tiles. Every IP then selects, from the set of received messages, only those messages whose destination field equals the ID of the tile. The behavior of this communication scheme is similar, but *not* identical, to the proliferation of an epidemic in a large population[1] [1].

1. This analogy explains intuitively the high performance and robustness of this protocol in the presence of failures.

This simple algorithm achieves many of the desired features of future NoCs. As shown in the following chapters, the algorithm provides:

- *Separation between computation and communication*, as the communication scheme is implemented in the network logic and is transparent to the IPs;
- *Fault-tolerance* since a message can still reach its destination despite severe levels of DSM failures on the chip;
- *Extremely low latency* since this communication scheme does not require retransmissions in order to deal with incorrect data;
- *Low production costs* because the fault-tolerant nature of the algorithm eliminates the need for detailed testing/verification;
- *Design flexibility* since it provides a mechanism to tune the trade-off between performance and energy consumption.

## 1.2 Related Work

Gossip protocols [4] have been used in computer networks and distributed databases. They are very attractive for applications that require *localized* communication, which is exactly the case for the architecture in Figure 1-1.

A distant ancestor is the USENET news protocol, NNTP [23], developed in the early 1980's. In this case, the news servers running the NNTP protocol exchange updates with their neighboring servers without knowing the entire set of hosts that are running NNTP worldwide. This way, a new message that has been sent to a newsgroup will propagate from server to server until it is known by all of them. One of the most interesting properties of this protocol is that the servers are *not* required to know about all of the other servers, and yet are able to broadcast the updates to the entire group. This reduces drastically the bandwidth required for the broadcast, which makes this protocol more scalable that most traditional distributed algorithms.

Demers et al. [11] proposed the use of randomized gossip protocols for the lazy update of data objects in a database replicated at many sites. In that paper, the authors have shown how gossip communication is related to the mathematics underlying the propagation of epidemics [1], and developed a family of gossip-based multicast protocols. In such an approach, every site periodically chooses another site randomly and sends only the recent updates that it is aware of. It can be proved that, in this manner, the updates spread *exponentially* fast among the replicated instances of the database, and that the broadcast is accomplished with only a few retransmission rounds.

Several networking protocols, such as the Internet Muse protocol [32], the Scalable Reliable Multicast [16], and the XPress Transfer Protocol [41], were based on the same principles. Birman et al. [4] have shown how to interpret the reliability guarantees offered by the gossip-based multicast protocols; there is a high probability that *almost all or almost none* of the players will receive the broadcast, as opposed to the stronger "all or none" guarantee of the classical distributed algorithms. Therefore, these protocols are best suited for applications that can tolerate a small percentage of message losses, but need to be scalable and have a steady throughput.

More recently, these types of algorithms have been applied to the networks of sensors [15]. Their ability to limit the communication to local regions and to support light-weight protocols, while still accomplishing their task is appealing to applications where power, complexity and size constraints are very critical. We argue that this communication paradigm can be successfully applied to the SoC design as well, especially in a network-on-chip type architecture like the one in Figure 1-1.

From a design perspective, in order to deal with node failures, Valtonen et al. [40] proposed an architecture based on autonomous, error-tolerant cells. In their approach, all cells can be tested at any time for errors and, if needed, disconnected from the network by the other non-faulty cells. However, the authors do not adequately specify a protocol that would ensure the desired fault-tolerance on such architectures. Furthermore, the problem of data upsets (see Chapter 2) and their impact on NoC communication has not been addressed yet, so the research in this thesis fills an important gap in the area of on-chip networks.

## 1.3 Structure of this thesis

The remainder of this thesis is organized as follows: Chapter 2 introduces a novel failure model for NoCs, which captures the typical errors that may appear in a DSM circuit. Chapter 3 presents a novel communication paradigm, that is specially engineered to work in a failure-prone NoC environment. Chapter 4 presents the experimental results that we have obtained with two case studies and a complex multimedia application (an MP3 encoder), while Chapter 5 outlines a vision for the future of SoC design, which enables the creation of very complex, heterogeneous systems, supported by our communication paradigm. We then conclude by summarizing our main contributions.

# 2 A Failure Model for NoCs

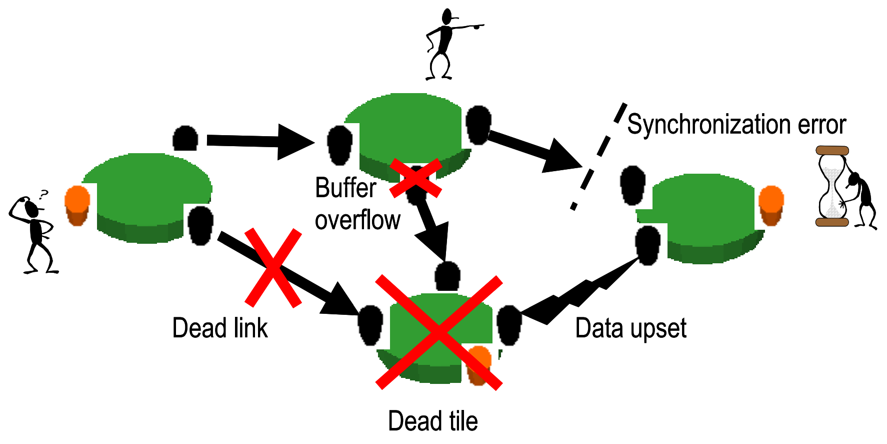
Several fault models have been identified in the traditional networking literature [19, 29]. *Crash failures* are *permanent* faults which occur when a tile halts prematurely or a link disconnects, after having behaved correctly until the failure. *Transient* faults can be either *omission failures*, when links lose some messages and tiles intermittently omit to send or receive, or *arbitrary failures* (also called Byzantine or malicious), when links and tiles deviate arbitrarily from their specification, corrupting or even generating spurious messages.

However, some of these models are not very relevant to the on-chip networks. In the DSM realm, the problems that are most likely to occur are fluxes of neutron and alpha particles, power supply and interconnect noise, electromagnetic interference, or electrostatic discharge, which cause soft errors, also known as *data upsets* [7]. The rate of occurrence of these errors increases as technology scales down into the deep submicron domain. Permanent failures occur infrequenly [33] and do not pose a serious threat to the mass production of VLSI chips. Furthermore, improvements of the semiconductor design and manufacturing techniques have led to a significant decrease of the permanent error rates during the past decade [7]. These trends emphasize the need for accurate failure models that are easy to manipulate and that realistically reflect the behavior of the circuits, in order to develop tools and methodologies for efficient system-level communication synthesis.

In the future, crosstalk and electromagnetic interference in DSM circuits will cause high levels of data transmission errors (data upsets) [37]. Simply stated, if noise in the interconnect causes a message to be scrambled, a data upset error will occur; these errors are subsequently characterized by a probability $p_{upset}$. Another common situation is when a message is lost because of buffer overflow; this is modeled by the probability $p_{overflow}$.

In the context of very complex SoCs, there are additional, more subtle error modes that can appear (see Figure 2-1). The high coupling capacities of the interconnect and the tighter integration favor the Miller effect which significantly affects on-chip delays [38]. As a consequence, it becomes very difficult to achieve predictable delays. Furthermore, as modern circuits span multiple clock domains (as in GALS architectures [5]), and can function at different voltages and frequencies (as in the "voltage and frequency island"-based architectures advocated by IBM [27]), the communication between domains has to be done through a special interface, which supports mixed clocks [6]. Because of the special handshake needed before the process of transferring data, the latency in communication may increase and *synchronization errors* are very hard to avoid. In our experiments, we have adopted a tile-based architecture in which every tile has its own clock domain. In this architecture, synchronization errors are normally distributed with a standard deviation $\sigma_{synchr}$.

**Figure 2-1.** A fault model for NoCs



Summarizing, the fault model we have developed depends on the following parameters:

- $p_{tiles}$ and $p_{links}$: probability that a tile/link is affected by a *crash failure*;
- $p_{upset}$: probability that a packet is scrambled because of a *data upset*;
- $p_{overflow}$: probability that a packet is dropped because of *buffer overflow*;

- $\sigma_{synchr}$: standard deviation error of the duration of a round ($T_R$) (see Section 3.3.1) which indicates the magnitude of *synchronization errors*.

If links are experiencing arbitrary failures, we must also consider how the transmitted information is altered [31]. If a message contains $n$ bits, the error vector is defined as: $\underset{\sim}{e} = (e_1, e_2, \ldots, e_n)$, where $e_i = 1$ if an error occurs in the $i^{\text{th}}$ transmitted bit and $e_i = 0$ otherwise. If all $2^n - 1$ non-null error vectors are equally likely to occur, we have the *random error vector model*. In this model, the probability of $\underset{\sim}{e}$ does *not* depend on the number of bit errors it contains, therefore:

$$p_{upset} = \sum_{\underset{\sim}{e} \neq 0} P[\underset{\sim}{e}] = (2^n - 1)p_v \cong 2^n p_v \Rightarrow p_v \cong \frac{p_{upset}}{2^n},$$

where $p_v$ is the probability of an error vector $\underset{\sim}{e}$. In contrast, in the *random bit error model*, $e_1 \ldots e_n$ are independent, so:

$$p_{upset} = 1 - \sum_{\underset{\sim}{e} = 0} P[\underset{\sim}{e}] = 1 - (1 - p_b)^n \cong np_b \Rightarrow p_b \cong \frac{p_{upset}}{n},$$

where $p_b$ is the probability of a bit error occuring.

We believe that establishing this stochastic failure model is a decisive step towards solving the fault-tolerant communication problem, as it emphasizes the nondeterministic nature of DSM faults. This suggests that a stochastic approach (described in Chapter 3) is best suited to deal with these realities.

# 3 Stochastic Communication

Traditionally, data networks have dealt with fault-tolerance by using complex algorithms, like the Internet Protocol or the ATM Layer [31]. However, these algorithms require many resources that are not available on chip they are not always able to guarantee a constant low latency, which is vital for SoCs. For example, in these protocols, the packets are protected by a *cyclic redundancy code (CRC)*, which is able to detect if a packet contains correct or upset data. If an error is detected, the receiver will ask for the retransmission of the scrambled messages. This method is known as the *automatic retransmission request (ARQ)* paradigm, and it has the disadvantage that it increases the communication latency. Another approach is the *forward error correction (FEC)*, where the errors are corrected directly by the receiver by using an error correction scheme, like the Reed-Solomon code. FEC is appropriate when a return channel is not available, as in deep-space communications or in audio CD recordings. FEC, however, is less reliable than ARQ and incurs significant additional processing complexity [31].

In light of these considerations, we propose a *fast* and *computationally lightweight* paradigm for the on-chip communication, based on an error-detection / multiple-transmissions scheme. The key observation behind our strategy is that, at the chip level, the bandwidth is less expensive than in traditional networks, because of existing high-speed buses and interconnection fabrics which can be used for the implementation of a NoC. Therefore, we can afford to have more packet transmissions than in the previous protocols in order to simplify the communication scheme and to guarantee low latencies. This chapter describes a technique called *on-chip stochastic communication*, which is built on this principle.

## 3.1 Foundations of stochastic communication

With this technique, the NoC communication is implemented using a *probabilistic broadcast* algorithm [12]. The behavior of such an algorithm is similar to the dissemination of a rumor within a large group of friends. Assume that, initially, only one person in the group knows the rumor. Upon learning the rumor, this person (the initiator) passes it to someone (confidant) chosen at random. At the next round, both the initiator and the confidant, if there is one, select independently of each other someone else to pass the rumor to. The process continues in the same fashion; that is, everyone informed after $t$ rounds passes the rumor at the $(t+1)^{\text{th}}$ round to someone selected at random, independently of all other past and present selections. Such a scheme is called a gossip algorithm and is known to model the spreading of an epidemic in immunology [1].

Let $I(t)$ be the number of people who have become aware of the rumor after $t$ rounds ($I(0) = 1$) and let $S_n = min\ \{t : I(t) = n\}$ be the number of rounds until $n$ people are informed. We want to estimate $S_n$, in order to evaluate how fast the rumor is spread. A fundamental result states that $I(t)$ is very close to its deterministic approximation defined by a difference equation:

$$I(t+1) = n - [n - I(t)]e^{-\frac{I(t)}{n}}, \qquad I(0) = 1$$

and

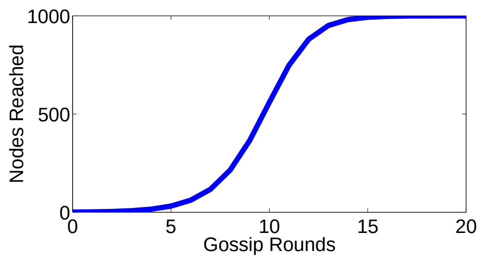
$$S_n = \log_2 n + \ln n + O(1) \qquad \text{as} \qquad n \to \infty \tag{1}$$

with probability 1 [36]. Therefore, after $O(\log_2 n)$ rounds ($n$ represents the number of nodes), all the nodes have received the message *with high probability (w.h.p.)*[1] [26]. For instance, in Figure 3-1,

1. The term *with high probability* means with probability at least $1 - O(n^{-\alpha})$ for some $\alpha>0$.

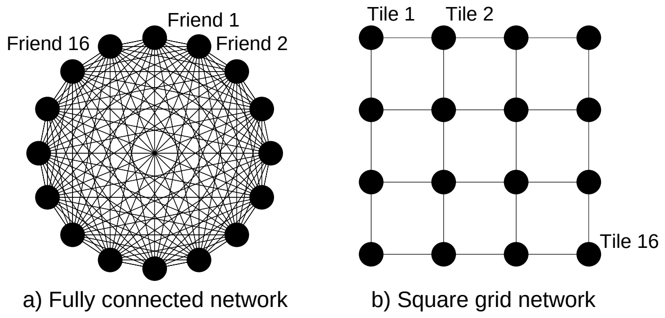
in less than 20 rounds, as many as 1000 nodes can be reached. Conversely, after *t* rounds the number of people that have become aware of the rumor is an exponential function of *t*, so this algorithm spreads rumors *exponentially fast*. The spread of the broadcast can therefore be stopped after $O(\ln n)$ rounds and, by then, the message will have reached its destination w.h.p.

**Figure 3-1.** Message spreading in a 1000 node fully connected network



The analogy we make for the case of NoCs is that tiles are the “gossiping friends” and that the packets transmitted between them are the “rumors” (see Figure 3-2). Since any friend in the original setup is able to communicate with anyone else in the group, the above analysis can be applied directly only to the case of a fully-connected network, like the one in Figure 3-2a. However, because of its high wiring demands, such an architecture is *not* a reasonable solution for SoCs. Therefore, for NoCs, we consider a grid-based topology (Figure 3-2b), since this is much easier and cheaper to implement on silicon. Although the theoretical analysis in this case is an open research question, our experimental results show that the messages can be disseminated explosively fast among the tiles of the NoC for this topology as well. To the best of our knowledge, this represents the first evidence that gossip protocols can be applied to SoC communication as well.

**Figure 3-2.** Different topologies for a 16-node network



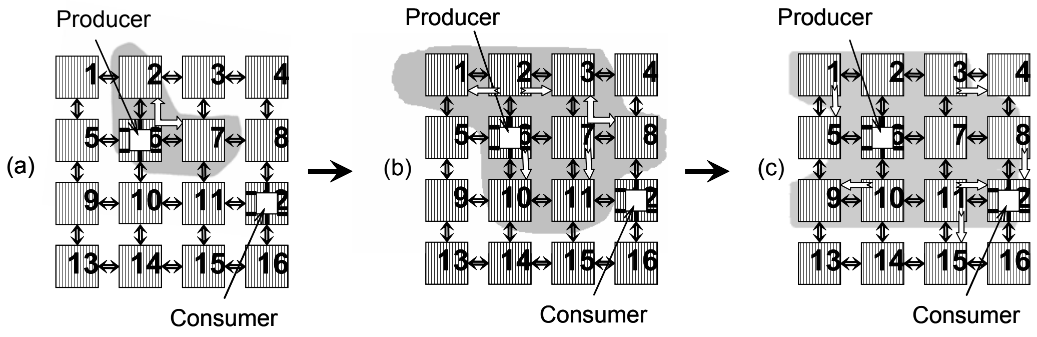
## 3.2 Stochastic communication for NoCs

### 3.2.1 Example: NoC Producer - Consumer application

In Figure 3-3, we provide the example of a simple Producer – Consumer application. On a NoC with 16 tiles, the Producer is placed on tile 6 and the Consumer on tile 12. Suppose the Producer needs to send a message to the Consumer. Initially the Producer sends the message to a randomly chosen subset of its neighbors (e.g., tiles 2 and 7 in Figure 3-3a). At the second gossip round, tiles 6, 2, and 7 (the Producer and the tiles that have received the message during the first round) forward it in the same manner. After this round, eight tiles (6, 2, 7, 1, 3, 8, 10, and 11) become aware of the message and are ready to send it to the rest of the network. At the third gossip round, the Consumer finally receives the packet from tiles 8 and 11. Note that:

- The Producer needs *not* know the location of the Consumer, the message will arrive at the destination *w.h.p.*, and
- The message reaches the Consumer *before* the full broadcast is completed; for instance, by the time the Consumer gets the message, tiles 13 - 16 have not yet received the message.

**Figure 3-3.** Producer - Consumer application in a stochastically communicating NoC

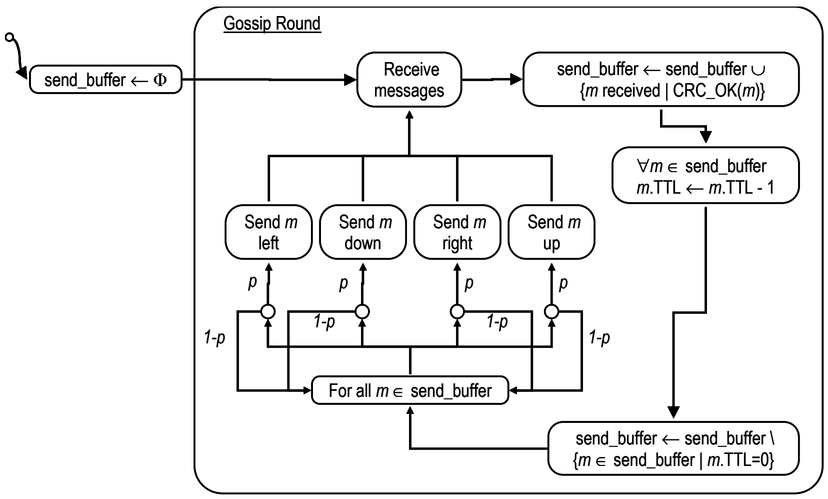


The most appealing feature of this algorithm is its excellent performance in the presence of failures. For instance, suppose the packet transmitted by tile 8 is affected by a data upset. The Consumer will discard it, as it receives from tile 11 another copy of the same packet anyway. The presence of such an upset is detected by implementing an error-detection scheme on each tile. The key observation is that, at the chip level, bandwidth is *less* expensive than in traditional macro-networks, because of existing high-speed buses and interconnection fabrics which can be used for the implementation of a NoC. Therefore, we can afford more packet transmissions than in previous protocols, in order to simplify the communication scheme and to guarantee low latencies.

## 3.2.2 The algorithm

The mechanism presented in the above example assumes that tiles can detect when data transmissions are affected by upsets. This is achieved by protecting packets with a cyclic redundancy code (CRC). If an error is discovered, then the packet will be simply discarded. Because a packet is retransmitted many times in the network, the receiver does *not* need to ask for retransmission, as it will receive the packet again anyway. This is the reason why stochastic communication can sustain low latencies even under severe levels of failures. Furthermore, CRC encoders and decoders are easy to implement in hardware, as they only require one shift register [31].

**Figure 3-4.** Stochastic Communication - The Algorithm



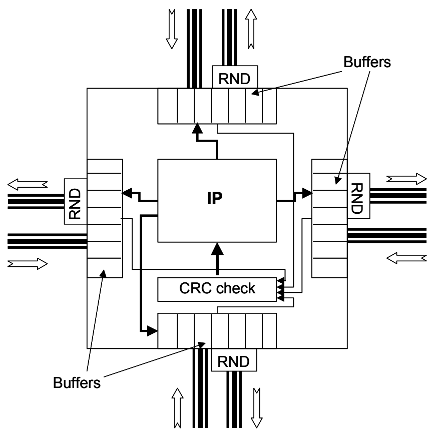
Gossip Round
send_buffer ← Φ
Receive messages
send_buffer ← send_buffer ∪ {m received | CRC_OK(m)}
∀m ∈ send_buffer m.TTL ← m.TTL - 1
Send m left
Send m down
Send m right
Send m up
p
1-p
For all m ∈ send_buffer
send_buffer ← send_buffer \ {m ∈ send_buffer | m.TTL=0}

Our algorithm is presented in Figure 3-4 (with standard set theory notations). The tasks executed by all the nodes are concurrent. A tile forwards the packet that is available for sending to all four output ports, and then a random decision is made (with probability $p$) whether or not to transmit the message to the next tile. We also note that, since a message might reach its destination before the broadcast is completed, the spread could be terminated even earlier in order to reduce the number of messages transmitted in the network. This is important because this number is directly connected to the bandwidth used and the energy dissipated (see Section 3.3.2). To do this, we assign a *time-to-live* (TTL) to every message upon creation and decrement it at every hop until it reaches 0; then, the message can be garbage-collected. In Chapter 4, we show how the probability *p* and the *TTL* can be used to tune the trade-off between performance and energy consumption.

### 3.2.3 The hardware interface

A typical tile of such a NoC is shown in Figure 3-5. The IP core is placed in the center of the tile. On the four edges of the tile, there exist buffers to hold the messages that are sent and received by the IP. A CRC decoding circuit checks all the received packets and when an error is discovered, the message is discarded before being fed into the IP.

**Figure 3-5.** A typical tile of a stochastically communicating NoC



The tile keeps a list of messages that have to be sent in an output buffer. The messages received during the last round and the new messages generated by the IP core are constantly added to the list. However, if a message is already present, a duplicate message will not be inserted. So, even if the message is received a second time from one of the tiles in the neighborhood, only one copy is kept in the send-buffer. The contents of this buffer will be sent to the neighbors during the next round. However, before being actually transmitted on a link, some messages will be randomly dropped by a specialized circuit. Such a circuit can be implemented by amplifying the thermal noise of a resistor and by sampling the signal at each round. The selected threshold voltage will determine the probability $p$ that a message is forwarded over a link. This is how the aforementioned random subset of neighbors is actually selected.

## 3.3 Performance evaluation of stochastic communication

This thesis proposes the stochastic communication paradigm with the intent to lower the overall costs of SoC design. A communication architecture developed on top of this paradigm does not need extensive verification and integration tests, since the NoC is designed to work even in the presence of faults. A validation of the chip’s correct functional behavior is still necessary, but lengthy simulations based on large sets of different input traces can be avoided. The behavior of the communication infrastructure can be characterized based on a small number of parameters, which are detailed in this section.

### 3.3.1 Performance metrics

A *broadcast round* is the time interval in which a tile has to finish sending all its messages to the next hops; this will usually take several clock cycles. The optimal duration of a round ($T_R$) can be determined using Equation 2, where $f$ is the maximum frequency of any link, $N_{packets/round}$ is the average number of packets that a link sends during one round (which is application-dependent), and $S$ is the average packet size.

$$T_R = \frac{N_{packets/round}S}{f} \tag{2}$$

As we show in Section 5, the fast dissemination of rumors in this algorithm makes it possible for us to achieve very low latencies. This protocol spreads the traffic onto all the links in the network, thereby reducing the chances that packets are delayed because of congestion. This is especially important in multimedia applications, where a sustainable constant bit-rate is highly desirable. Furthermore, the fact that we don't store or compute the shortest paths (as is the case of dynamic routing) makes this algorithm computationally lightweight, simpler and easier to customize for every application and interconnection network.

The following parameters are relevant to our analysis:

- The *number of broadcast rounds* needed, which is a direct measure of the inter-IP communication latency;
- The *total number of packets* sent in the network, which indicates the bandwidth required by the algorithm and can be controlled by varying the message TTL;
- The *fault-tolerance*, which evaluates the algorithm's resilience to abnormal functioning conditions in the network;
- The *energy consumption*, which is computed with Equation 3 (see next section).

### 3.3.2 Energy metrics

In estimating this algorithm’s energy consumption, we take into consideration the total number of packets sent in the NoC, since these transmissions account for the switching activity at the network level. This is expressed by Eq. 3, where $N_{packets}$ is the total number of messages generated in the network, $S$ is the average size of one packet (in bits) and $E_{bit}$ is the energy consumed per bit:

$$E_{total} = E_{computation} + E_{communication} = E_{computation} + N_{packets} S E_{bit} \tag{3}$$

$N_{packets}$ can be estimated by simulation, $S$ is application-dependent, and $E_{bit}$ is a parameter from the technology library. As shown in Equation 3, the total energy consumed by the chip will be influenced by the activity in the computational cores as well ($E_{computation}$). Since we are trying to analyze here the performance and properties of the *communication scheme*, estimating the energy required by the computation is not relevant to this discussion. This can be added, however, to our estimations from Chapter 4 in order to get the combined energy.

# 4 Practical Considerations and Experimental Results

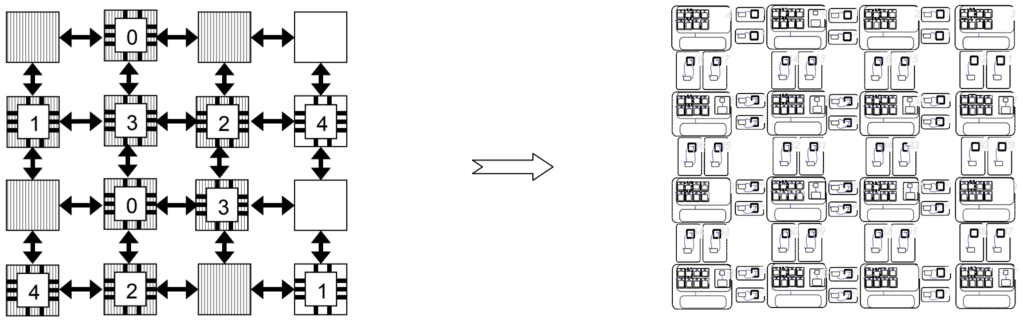
Stochastic communication can have wide applicability, ranging from parallel SAT solvers and multimedia applications to periodic data acquisition from non-critical sensors. We first demonstrate how our technique works on two simple case studies (a simple master/slave application and the two-dimensional Fast Fourier Transform), and later for a complex multimedia application (an MP3 encoder). We simulate these applications in a stochastically communicating NoC environment, which assumes the failure model described in Chapter 2. As realistic data about failure patterns in regular SoCs are currently unavailable, the whole parameter space of our fault model is *exhaustively* explored in this chapter.

Another important parameter we vary is $p$, the probability that a packet is forwarded over a link (see Section 3.2). For example, if we set the forwarding probability to 1, then we have a completely deterministic algorithm which floods the network with messages (every tile always sends messages to all its neighbors). This algorithm is optimal with respect to latency, since the number of intermediate hops between source and destination is always equal to the Manhattan distance, but it is extremely inefficient with respect to the bandwidth used and the energy consumed. Stochastic communication allows us to tune the trade-off between energy and performance by varying the probability of transmission $p$ between 0 and 1. The present chapter compares four versions of the stochastic communication, obtained for different values of this parameter ($p = 1$, $p = 0.75$, $p = 0.50$ and $p = 0.25$). In the following sections, we discuss our results for this experimental setup.

## 4.1 Case studies

We have implemented the gossip algorithm in Stateflow[1], which is a tool to describe and simulate concurrent behavior of complex systems. Stateflow uses a formalism defined in [20], where a system is described by a hierarchical state machine with both parallel and exclusive states (see Figure 4-1). Our models simulate grids of 16–25 tiles, which is relevant to the current realities of SoC design. However, the gossip algorithms are known to scale extremely well even beyond these dimensions; therefore, stochastic communication can applied to much larger designs as well.

**Figure 4-1.** Stochastic communication implemented in Stateflow



For these case studies we have considered a restricted version of the fault model from Chapter 2, which includes only the most cited failure modes in literature: crash failures and data upsets. We evaluate the *latency*, *the energy dissipation* and the *fault-tolerance* of our approach; all of the results presented in this section are averages obtained after several repeated simulations.

### 4.1.1 Master-Slave computation

One of the most common paradigms in concurrent applications is the Master – Slave model. An example of such an application is computing the value of π. This has been one of the oldest mathematical challenges in history, ever since Archimedes[2] has proved that $\frac{223}{71} < \pi < \frac{22}{7}$. A more modern approach is to estimate π using a parallel computing architecture. For instance, we can estimate π as:

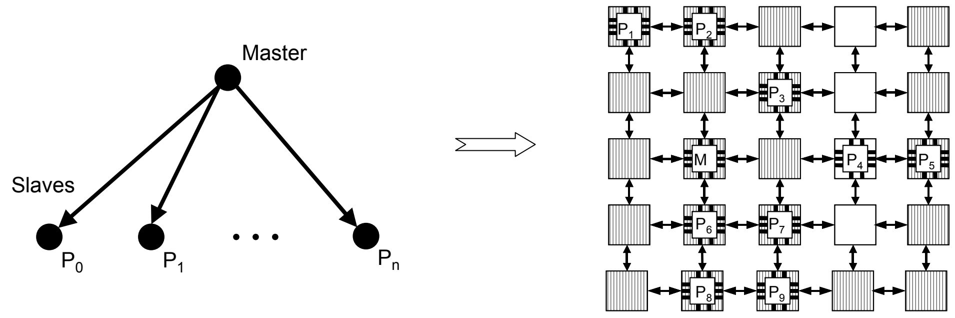
$$\pi = \int_0^1 \frac{4}{1+x^2}dx \cong \sum_{i=0}^{n} \frac{1}{n} \cdot \frac{4}{1+\left(\left(i-\frac{1}{2}\right)\cdot\frac{1}{n}\right)^2} \tag{4}$$

1. Stateflow is a module of Matlab.

2. Estimates of π had been known before, but they were obtained through observations and measurements, rather than a rigorous analysis.

The latter sum can be broken into *N* partial sums, which are computed in parallel by different modules. This is a typical Master – Slave problem, as seen in Figure 4-2. A master starts *N* slaves, sends them their corresponding summation limits, with each slave computing one partial sum. When all the slaves have finished, the master collects all the results and computes the final value.

**Figure 4-2.** Master - Slave scheme for computing the value of $\pi$



We implement this algorithm on a 5×5 grid. We map the master and eight slaves on the 25 tiles of the NoC, which are communicating using the gossip algorithm described in Chapter 3. As mentioned before, this algorithm provides very good fault-tolerance to the inter-IP *communication*. In order to increase the *computation*'s fault-tolerance as well, each slave can be duplicated, such that if one of the IPs computing $P_k$ is located on a dysfunctional tile, the remaining one will still be able to provide the partial result. Because the replicated tiles produce identical results, the Master IP does not have to wait for both versions and it will, therefore, process the partial results as soon as they arrive.

### 4.1.2 The 2D Fast Fourier Transform

The Fourier Transform is a method to describe and analyze the spectrum of a signal and has multiple applications in linear systems analysis, antenna studies, optics, random process modeling, probability theory, quantum physics, signal processing, or the restoration of astronomical data. Its most common implementation is known as the Fast Fourier Transform (FFT), which is currently one of the largest single consumers of floating-point cycles in modern CPUs[1] [8].

1. The algorithm was first discovered by Gauss, who used it for the interpolation of asteroidal orbits from a finite set of equally spaced observations.

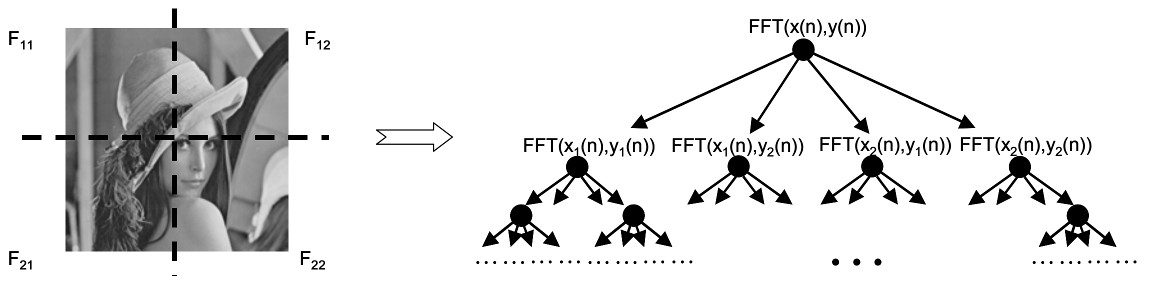
The Discrete Fourier Transform of *N* samples is defined as:

$$\hat{x}(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-\frac{2i\pi}{N}kn} = \begin{cases} \frac{1}{2}\left(\hat{x}_1(k) + e^{-\frac{2i\pi}{N}k}\hat{x}_2(k)\right), k \le \frac{1}{2} \\ \frac{1}{2}\left(\hat{x}_1(k) - e^{-\frac{2i\pi}{N}k}\hat{x}_2(k)\right), k > \frac{1}{2} \end{cases} \tag{5}$$

where $\hat{x}_1(k) = \hat{x}(2k)$ and $\hat{x}_2(k) = \hat{x}(2k+1)$ .

Hence, a recursive divide-and-conquer algorithm will be used to compute the FFT of N samples (Figure 4-3). This reduces the number of operations from $O(N^2)$ to $O(N \log_2 N)$. Using this scheme, the FFT algorithm can be implemented on a parallel architecture as well. Because of its widespread use in engineering problems and especially in image and multimedia processing, we have focused on the *two-dimensional FFT* algorithm (FFT2), where Equation 5 is applied to both dimensions.

**Figure 4-3.** Parallel scheme for computing FFT



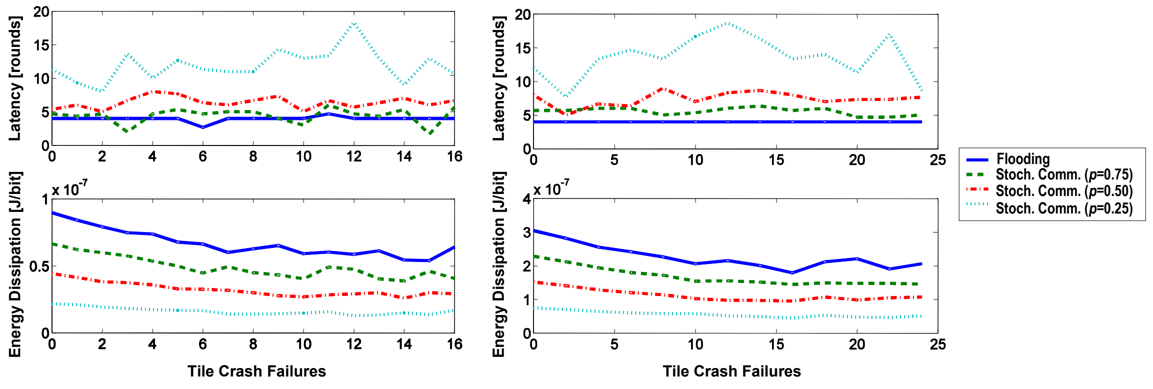
Every node in the tree from Figure 4-3 represents a parallel process. The leaves compute the FFT on a small number of samples and send the results back up to the root, which will finally assemble the full FFT. We mapped this application onto a 4×4 NoC, running the stochastic communication algorithm described in Chapter 3. In order to increase the tolerance to tile crash failures, the IPs can be duplicated as in the case of the Master - Slave computation.

## 4.1.3 Experimental results for the case studies

From the communication point of view, both these applications have two phases: first, the initial message (containing the summation limits or the raw image samples) has to reach all of the leaf nodes, and second, the computed results have to come back to the root node, which will assemble the final result. The experiments have shown that the first phase, the spread of the initial broadcast, is slow in the beginning, but soon reaches a stage of explosive growth until the entire network becomes aware of the message after a small number of rounds. This behavior is very similar to the one predicted by theory (see Figure 3-1).

The end of the second phase, i.e. the time when all the partial results have reached the root node, marks the completion of the application[1]. In the upper part of Figure 4-4, we can see that the stochastic communication algorithm with probability of transmission $p = 0.5$ has an overall latency of 6–9 rounds for Master-Slave and 5–8 rounds for FFT2, which are very close to the 4 rounds needed by the flooding algorithm (which is optimal with respect to latency). These numbers are dependent on the mapping of IPs to tiles; in some cases, we have noticed that the replies may come back before the full broadcast of the original message reaches all the tiles of the network. This indicates that the mapping phase of the system-level design has to take into account the communication performance in order to obtain an efficient design [21].

**Figure 4-4.** Latency and energy dissipation for stochastic communication



a) Results for FFT b) Results for Master-Slave

While stochastic communication with $p = 0.50$ has a latency close to optimal, its energy dissipation is about half of the one of the flooding algorithm (see the lower part of Figure 4-4). This is

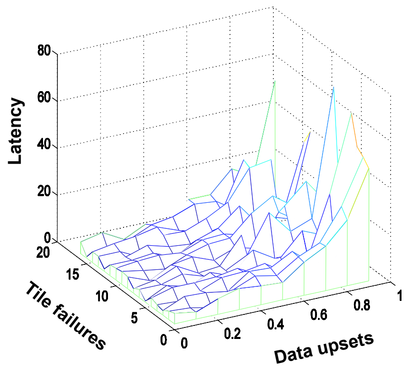
1. Because this thesis is primarily concerned with the on-chip communication, the computation times are generally assumed to be 0.

because the energy is proportional to the total number of messages generated in the network (see Section 3.3.2), which is controlled by the probability of transmission $p$. This observation indicates that, by modifying this parameter of the algorithm, a designer can tune the trade-off between the performance and the energy dissipation of the communication architecture.

Since the energy dissipation depends on the total number of messages, it will also be dependent on the dimensions of the network; this explains the higher power figure of the Master-Slave simulation, which was done on a 5×5 grid, while the FFT2 simulation was done on a 4×4 grid. However, the power consumption of the communication scheme does not depend on the degree of duplication of the modules. The redundant IPs generate the same messages, so the number of unique messages in the network will not increase.

Different types of faults have different effects on the performance of stochastic communication. From Figure 4-4, we can see that the number of tile failures does not have a big impact on latency. However, if a significant number of tile and link crashes occurs during the early stages of the algorithm (which is very unlikely with modern manufacturing technologies, as explained in Chapter 2), the applications will fail completely because too many important modules are not working or entire regions of the NoC are isolated.

**Figure 4-5.** Impact of defective tiles and data upsets on latency

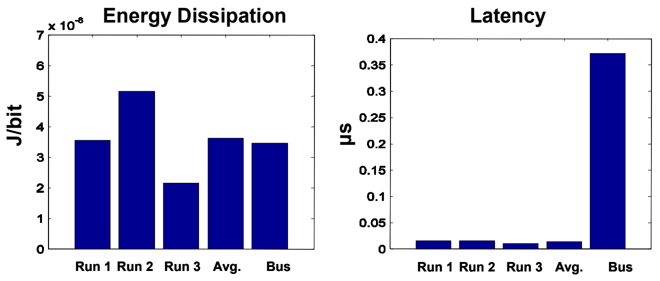


On the other hand, data upsets seem to have little influence on the probability that the applications will be able to terminate; however, upsets do have an impact on the latency, especially if $p_{upset}$ > 0.5. Figure 4-5 gives a visual interpretation of how the defective tiles and data upsets influence the latency of stochastic communication. Data upsets will, generally, not cause the communication to fail completely; the algorithm does not give up and eventually terminates with levels of data upsets as high as 90%, even if it requires 100 rounds to do so.

### 4.1.4 Comparison with a bus-based solution

In order to validate the benefits of our technique, we have also compared our results with those of a bus-based solution. Note that, because the shared bus represents a single point of failure, true fault-tolerance is hard to achieve in this architecture. In contrast, our stochastically communicating architecture still works and maintains low latencies, even under severe levels of failures. However, we felt that such a comparison, even if not entirely natural, would make sense because it provides an indication of how our method compares with existing techniques, in the case when the chip is functioning correctly.

**Figure 4-6.** Comparison with a bus-based solution



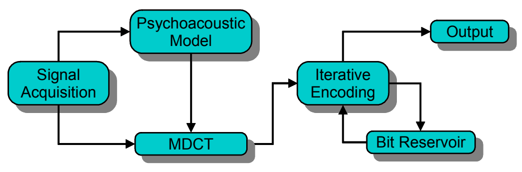
We have assumed that each tile contains one DSP module (M320C50) in the stochastic communication-based solution, and that the same modules are connected to a bus, in the case of the classic solution. Both implementations use a 0.25 μm technology. The length of the bus is equal to the side of the tile-based grid, and the modules were placed on both sides of the bus. Based on these parameters, we calculated that the bus has a maximum working frequency of 43 MHz and it dissipates $21.6 \cdot 10^{-10}$ J for every transmitted bit, while in the case of the tile-based architecture, a link has a maximum working frequency of 381 MHz and dissipates $2.4 \cdot 10^{-10}$ J per bit. In both cases, we have ignored the power consumed by the IPs and the time spent during the computation, since we wanted to compare the two communication techniques. For the bus, we have also ignored the overhead introduced by the arbitration scheme, which usually tends to be negligible when compared to the time and the power needed by the data transmissions. In the case of the tile-based architecture, the energy consumption was estimated with Equation 3, and the latency using Equation 2.

From Figure 4-6, we can see that, on average, the energy consumed by the tile-based chip was only 5% greater than in the case of the bus. However, the latency of the stochastic communication was *11 times better* than that of the bus, because the links are shorter, and thus faster, and because the messages can arrive on parallel ports and do not have to compete for shared resources. Furthermore, the energy×delay product is better for the stochastically communicating NoC ($7 \cdot 10^{-12}$ J·s per bit) than for the bus ($133 \cdot 10^{-12}$ J·s per bit).

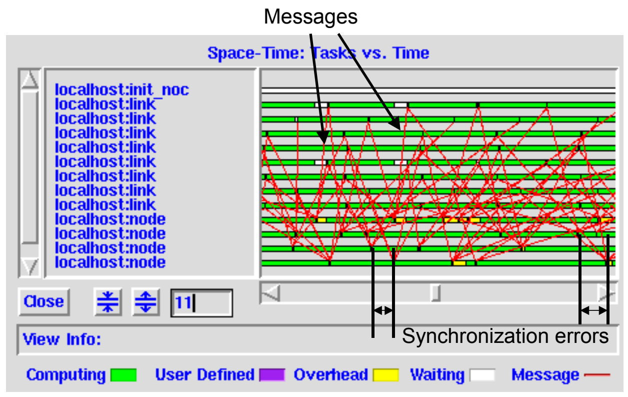
## 4.2 A complex application: MP3 encoder

MP3 encoders and decoders are ubiquitous in modern embedded systems, from PDAs and cell phones to digital sound stations, because of the excellent compression rates and their streaming capabilities. For the purposes of this complex application, we have developed a stochastic communication simulator using the Parallel Virtual Machine (PVM)[1] and implemented a parallel version of the LAME MP3 encoder [39] (see Figure 4-7).

**Figure 4-7.** Experimental setup of the MP3 application



a) Modules of the MP3 application



b) Snapshot of the simulator

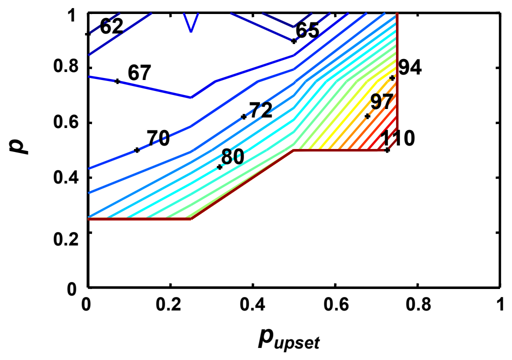
1. PVM is a programming platform which simulates a message passing multi-processor architecture [18].

The communication scheme follows the description from Section 3.2 and the fault model is that introduced in Chapter 2. In addition to the failure modes considered for the case studies, we also handle buffer overflows and synchronization errors. In this experiment, it is assumed that each NoC tile has its own local clock and that these clocks are *not* necessarily synchronized and the messages are transferred through a special interface [6]. From a simulation perspective, this means that the duration of each round is normally distributed around the round period $T_R$ (see Equation 2), with a standard deviation $\sigma_{synchr}$. The tiles have finite message buffers, which leads to a certain probability of overflow $p_{overflow}$; if such an overflow happens, the respective tile will lose some of the messages (the oldest ones are dropped first).

### 4.2.1 Latency

The latency is measured as the number of rounds that MP3 needs in order to finish encoding and it is influenced by several parameters: the value of $p$ (probability of forwarding a message over a link), the levels of data upsets in the network, the buffer overflows and the synchronization errors. Figure 4-8 shows how the latency varies with the probability of retransmission $p$ and with the probability of upsets $p_{upset}$. As expected, the lowest latency is when $p = 1$ (the messages are always forwarded over a link) and $p_{upset} = 0$ (when there are no upsets in the network), and increases when $p \to 0$ and $p_{upset} \to 1$ to the point where the encoding of the MP3 cannot finish because the packets fail to reach their destination too often. Note that $p_{upset}$ is dependent on the technology, while $p$ is a parameter that can be used directly by the designer in order to tune the behavior of the algorithm.

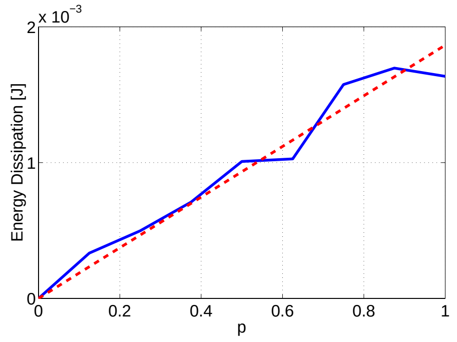
**Figure 4-8.** Latency of the MP3 application



### 4.2.2 Energy dissipation

We have estimated the energy dissipation of our algorithms using Equation 3. We can see from Figure 4-9 that the energy dissipation increases almost linearly with the probability *p* of resending. This is because the energy is proportional to the total number of packets transmitted in the network, a value dictated by *p*. As we can observe in Figures 4-8 and 4-9, high values of *p* mean low latency, but they also mean high energy dissipation. This shows the importance of this parameter, which can be used to tune the trade-off between performance and energy dissipation, making the stochastic communication flexible enough to fit the needs of a wide range of applications.

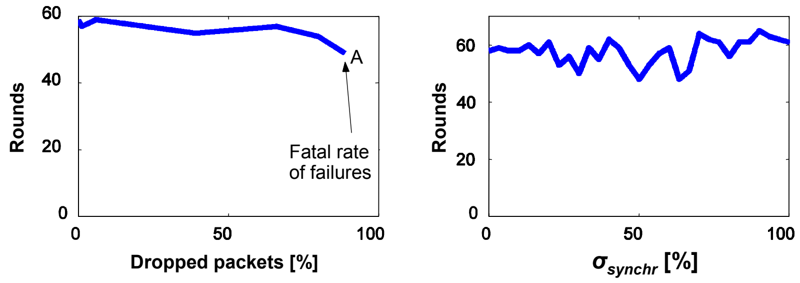
**Figure 4-9.** Energy Dissipation of the MP3 application



### 4.2.3 Fault-tolerance

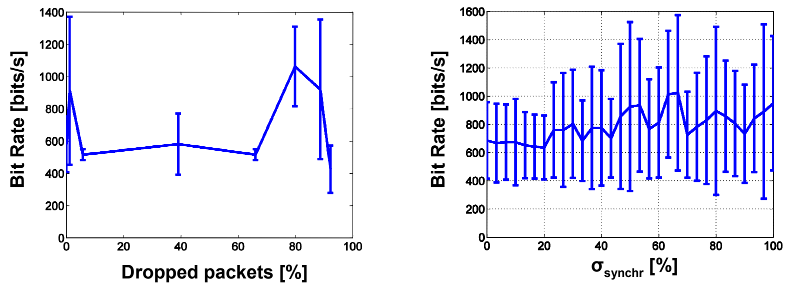
As for the case studies, different types of faults have different effects on the performance of stochastic communication. The levels of buffer overflow do not seem to have a big impact on latency (see the left part of Figure 4-10). However, the encoding will not be able to complete if these levels are too high (>80%, as in point A in Figure 4-10) because one or several packets are lost and none of the tiles has kept copies of those messages. From the right part of Figure 4-10, we can notice that synchronization errors do not prevent the application from terminating; however, they do cause a greater variability in the latency. Data upsets also seem to have little influence on the chances to finish encoding, but have a significant impact on the latency, especially if $p_{upset} > 0.7$ (see Figure 4-8). Summarizing, we can observe that buffer overflows and synchronization errors have little impact on the latency (buffer overflows may prevent the application from terminating and synchronization errors introduce a greater jitter), while data upsets increase the latency considerably.

**Figure 4-10.** Impact of the on-chip failures on the latency



We can reach similar conclusions by monitoring the continuous bit-rate at the output of the MP3 encoder. From Figure 4-11, we see that bit-rates are sustainable with as much as 60% of the packets dropped because of buffer overflows and under severe levels of synchronization failures. Note that even very important synchronization error levels do not have a great impact on the bit-rate, or on the jitter in the output bit-stream (depicted by the error bars in Figure 4-11). This proves that our method tolerates very high levels of synchronization errors extremely well.

**Figure 4-11.** Impact of on-chip failures on the bit rate

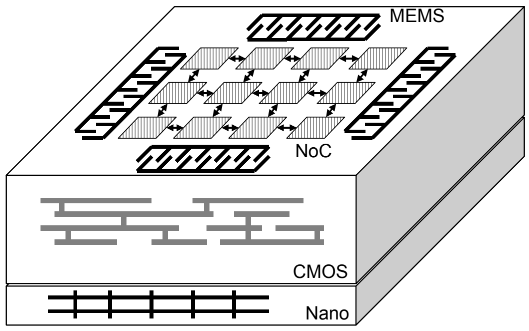


These results show that stochastic communication has a very good behavior in time with respect to latency and energy dissipation. In the worst case (very unlikely for typical levels of failures in NoCs), the protocol will not deliver the packet to the destination; therefore this communication paradigm may be better suited for applications which tolerate small levels of information loss, such as our MP3 encoder. In general, real-time streaming multimedia applications have requirements that match the behavior of our new communication paradigm well, as they can deal with small information losses as long as the bit-rate is steady. The results show that our MP3 encoder tolerates very high levels of failures with a graceful degradation in quality. If, however, the application requires strong reliability guarantees, these can be implemented by a higher level protocol built on top of the stochastic communication.

# 5 On-Chip Diversity

In the previous chapters we have presented a communication technique that can deal with the most common failure modes of deep sub-micron CMOS circuits. However, in addition to the high transistor count introduced by the new CMOS technologies, another dimension of complexity is currently emerging in SoC design: the need to integrate synchronous and asynchronous domains, mixed-clock and mixed-signal circuits, silicon and non-silicon technologies, etc. Because of power dissipation, parameter variations, and hard-to-predict side-effects that characterize DSM circuits, the full potential of the current technologies cannot be obtained out of CMOS alone [24, 25, 27]. Consequently, there is an increased demand for *hybrid systems* which combine different technologies and design styles in order to obtain high-performance, scalable SoCs, capable of meeting the functionality and robustness requirements. We epitomize this trend under the name of *on-chip diversity* (see Figure 5-1), and believe that defining a system-level design framework is the key to making these solutions affordable for designs that do not have massive production volumes.

**Figure 5-1.** On-chip diversity

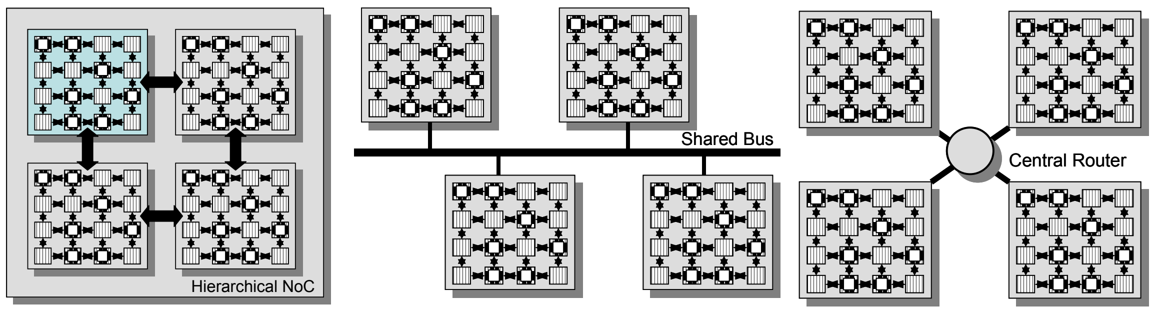


CMOS technologies offer high computational power and the advantage of well-established design methodologies, nanoelectronic solutions promise unprecedented levels of device density, as well as low-power and high frequencies, while MEMS add very precise sensing and actuation capabilities. CAD tools and methodologies have been developed for all these areas of design; however, a system-level integrating approach has yet to be defined. The generic system in Figure 5-1 depicts a new *design platform* for integrating multiple clock domains, CMOS technologies, nanoelectronics and MEMS-based devices in a unitary system that combines the best properties of all these structures: the system can sense, actuate and process information in a highly integrated manner. Characterizing the design space available for on-chip diversity is important for identifying the trade-offs that can be made for the design of efficient systems. Within the class of solutions that seem feasible in the current industrial context, we can identify two categories:

- The combination of different *architectural styles*, which means partitioning the chip into several islands with separate clocks [5] and voltages [27], with the purpose of optimizing a specific parameter, such as energy consumption;
- The combination of different *technologies*, which means assembling together, for instance, CMOS, nanoelectronics and/or MEMS devices, in order to combine the computational power of silicon with the very high device densities enabled by nano-technologies [43] and to add new features to the design.

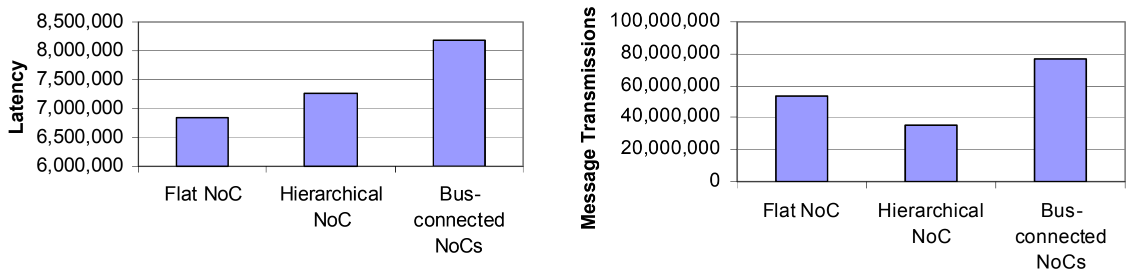
The goal of having true on-chip diversity depends essentially on solving the problem of *communication* between heterogeneous components. Indeed, an efficient and reliable system-level integration of the various technologies can only be achieved if an efficient communication infrastructure can be designed and implemented. The bus-based interconnects and the NoCs each have their own advantages: buses are very efficient when a only few communicating IPs are connected or when the application needs to broadcast a lot of data, while stochastically communicating NoCs are very scalable and can support a large number of IPs without suffering a significant performance degradation under the influence of on-chip failures. However, as most SoC applications do not have a uniform structure and in order to take advantage of the properties of on-chip diversity, a combination of these structures would be more appropriate (see Figure 5-2).

**Figure 5-2.** Possible communication architectures for on-chip diversity



Using a reliable and highly efficient communication scheme such as the one developed in this thesis might be the enabling factor for implementing on-chip diversity. Preliminary results from an experiment conducted on an acoustic beamforming application [42] indicate that stochastic communication running in a hierarchical NoC (the leftmost solution in Figure 5-2) has the lowest number of message transmissions, leading to the lowest power consumption, while the flat NoC (such as the one studied in the previous chapters) has a slightly better latency than the other solutions. Using a shared bus to connect four stochastically communicating NoCs (see the middle of Figure 5-2) is less efficient than the other architectures; however, because bus-based solutions are widely used today, such a hybrid structure would smoothen the transition to the novel communication architectures, instead of imposing a sudden paradigm shift toward NoCs [14].

**Figure 5-3.** On-chip diversity: comparing different architectures



Combining heterogeneous architectures and technologies in a multiple voltage/frequency island environment allows circuits to achieve the highest levels of performance. It also introduces a new dimension of flexibility in SoC design. By allowing such designs to be partially based on existing CAD tools and design practices, the transition to these novel structures would certainly be smoother than a sudden paradigm shift to a radically new technology.

# 6 Summary and Conclusions

This thesis emphasizes the need for on-chip fault-tolerance and proposes a novel on-chip communication paradigm based on stochastic communication. This approach takes advantage of the large bandwidth that is available on chip in order to provide the needed system-level tolerance to synchronization errors, data upsets and buffer overflows. The experimental results show that this approach is more robust and has a much lower latency compared to a traditional bus-based implementation, while keeping the energy consumption at about the same level. At the same time, this method drastically simplifies the design process by offering a low-cost and easy to customize solution for on-chip communication and it also enables the hybrid architectures that will be used for on-chip diversity. The author believes that the results presented within the pages of this thesis indicate the significant potential of the stochastic communication approach, with promising implications that further research in this area would lead to vital improvements of the SoC interconnect design.

# Bibliography

[1] Bailey, N. The Mathematical Theory of Infectious Diseases. Charles Griffin and Company, London, 2 edition, 1975.

[2] Benini, L. and De Micheli, G. Networks on chips: A new SoC paradigm. In IEEE Computer, 35(1): 70--78, January 2002.

[3] Bertozzi, D., Benini, L. and De Micheli, G. Low power error resilient encoding for on-chip data buses. In Proc. of DATE, 2002.

[4] Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., and Minsky, Y. Bimodal Multicast. ACM Transactions on Computer Systems, 17(2):41-88, 1999.

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

[6] Chelcea, T. and Nowick, S. Robust Interfaces for Mixed-Timing Systems with Application to Latency-Insensitive Protocols. In Proc. DAC, 2001.

[7] Constantinescu, C. Impact of Deep Submicron Technology on Dependability of VLSI Circuits. In Proc. DSN, 2002.

[8] Cooley, J. W. and Tukey, J. W. An algorithm for the machine calculation of complex Fourier series. In Mathematics of Computation, 19, 1965.

[9] Dally, W. and Towles, B. Route packets, not wires: On-chip interconnection networks. In Proc. of Design Automation Conference, June 2001.

[10] De Micheli, G. Designing Robust Systems with Uncertain Information. ASP-DAC 2003 Keynote Speech, January 2003.

[11] Demers, A., et. al. Epidemic algorithms for replicated database maintenance. In Proceedings of the ACM Symposium on Principles of Distributed Computing, August 1987.

[12] Dumitraş, T. and Mărculescu, R. On-chip stochastic communication. In Proc. DATE, 2003.

[13] Dumitraş, T., Kerner, S. and Mărculescu, R. Towards on-chip fault-tolerant communication. In Proc. ASP-DAC, 2003.

[14] Dumitraş, T., Kerner, S. and Mărculescu, R. Enabling On-Chip Diversity Through Architectural Communication Design. Technical Report CSSI 03-04, Carnegie Mellon University, Department of Electrical and Computer Engineering, April 2003.

[15] Estrin, D., Govindan, R., Heidemann J., and Kumar, S. Next century challenges: Scalable coordination in sensor networks. In Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking. ACM, August 1999.

[16] Floyd, S. et. al. A reliable multicast framework for light-weight sessions and application level framing. In IEEE/ACM Transactions on Networking, 5(6):784-803, December 1997.

[17] Gasieniec, L. and Pelc A. Adaptive broadcasting with faulty nodes. In Parallel Computing, 22(6):903–912, 1996.

[18] Geist, A., et al. PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, Massachusetts, 1994.

[19] Hadzilacos, V. and Toueg, S. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Department of Computer Science, May 1994.

[20] Harel, D. Statecharts: A visual formalism for complex systems. Sci. of Comp. Progr., 8(3):231–274, June 1987.

[21] Hu, J. and Mărculescu, R. Energy-aware mapping for tile-based NoC architectures under performance constraints. In Proc, of DATE, 2003.

[22] Jantsch, A. and Tenhunen, H. Networks on Chip. Kluwer, 2003.

[23] Kantor, B. and Lapsley, P. Network News Transfer Protocol. RFC 977, February 1986. Available from URL ftp://nis.nsf.net/documents/rfc/rfc0977.txt.

[24] Kao, J. et. al. Subthreshold Leakage Modelling and Reduction Techniques. In Proc. ICCAD, 2002

[25] Karnik, T. et. al. Sub-90nm Technologies--Challenges and Opportunities for CAD. In Proc. ICCAD, 2002

[26] Karp R. et. al. Randomized rumor spreading. In Proc. IEEE Symp. on Foundations of Computer Science, 2000.

[27] Lackey, D. et. al. Managing Power and Performance for System-on-Chip Designs using Voltage Islands. In Proceedings of the ICCAD, 2002.

[28] Lahiri, K., Raghunathan A., Dey, S. System-Level Performance Analysis for Designing On-Chip Communication Architectures. In IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 20(6), pages 768-783, June 2001.

[29] Laprie, J.C. Dependability: Basic concepts and terminology. Springer-Verlag, 1992.

[30] Leighton, F. T. et. al. On the fault tolerance of some popular bounded-degree networks. In IEEE Symp. on Foundations of Comp. Sci., 1992.

[31] Leon-Garcia, A. and Widjaja, I. Communication Networks. McGraw-Hill, 2000.

[32] Lidl, K. J. et. al. Drinking from the Firehose: Multicast USENET News. In Proceedings of the USENIX Winter, 1994.

[33] Lin, T. Y. and Siewiorek D. P. Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis. In IEEE Transactions on Reliability, Vol. 39, No. 4, 1990, pp. 419-432.

[34] Maly, V. IC design in high-cost nanometer technologies era. In Proc. of The 38th DAC, June 2001.

[35] Ni, L. M. and McKinley, P. K. A survey of wormhole routing techniques in direct networks. IEEE Computer, 1993.

[36] Pittel, B. On spreading a rumor. SIAM Journal of Appl. Math., 1987.

[37] Semiconductor Association. The International Technology Roadmap for Semiconductors (ITRS), 2001.

[38] Sylvester, D. and Keutzer, K. Rethinking deep-submicron circuit design. IEEE Computer, Nov. 1999.

[39] The LAME project. http://www.mp3dev.org/mp3/.

[40] Valtonen, T. et. al. Interconnection of autonomous error-tolerant cells. In Proc. ISCAS, 2002.

[41] XTP Forum, Xpress Transfer Protocol Specification, Revision 4.0, March 1995

[42] Zhang, F. et. al. Parallelization and performance of 3D ultrasound imaging beamforming algorithms on modern clusters. In Proc. of the Int. Conf. on Supercomputing, 2002.

[43] Ziegler, M. and Stan, M. A Case for CMOS/nano Co-design. In Proc. ICCAD, 2002.