Timing-Error-Tolerant Network-on-Chip Design Methodology

Rutuparna Tamhankar, *Member, IEEE*, Srinivasan Murali, *Student Member, IEEE*, Stergios Stergiou, Antonio Pullini, Federico Angiolini, Luca Benini, *Senior Member, IEEE*, and Giovanni De Micheli, *Fellow, IEEE*

***Abstract*—With technology scaling, the wire delay as a fraction of the total delay is increasing, and the communication architecture is becoming a major bottleneck for system performance in *systems on chip (SoCs)*. A communication-centric design paradigm, *networks on chip (NoCs)*, has been proposed recently to address the communication issues of *SoCs*. As the geometries of devices approach the physical limits of operation, *NoCs* will be susceptible to various noise sources such as crosstalk, coupling noise, process variations, etc. Designing systems under such uncertain conditions become a challenge, as it is harder to predict the timing behavior of the system. The use of conservative design methodologies that consider all possible delay variations due to the noise sources, targeting *safe* system operation under all conditions will result in poor system performance. An aggressive design approach that provides resilience against such timing errors is required for maximizing system performance. In this paper, we present *T-error*, which is a timing-error-tolerant aggressive design method to design the individual components of the NoC (such as switches, links, and network interfaces), so that the communication subsystem can be clocked at a much higher frequency than a traditional conservative design (up to 1.5× increase in frequency). The NoC is designed to tolerate timing errors that arise from *overclocking* without substantially affecting the latency for communication. We also present a way to dynamically configure the NoC between the *overclocked* mode and the *normal mode*, where the frequency of operation is lower than or equal to the traditional design's frequency, so that the error recovery penalty is completely hidden under normal operation. Experiments on several benchmark applications show large performance improvement (up to 33% reduction in average packet latency) for the proposed system when compared to traditional systems.**

***Index Terms*—Aggressive design, networks on chip (NoCs), systems on chip (SoCs), timing errors.**

I. Introduction

DUE TO shrinking feature sizes and increasing transistor densities, the number of processor/memory cores on a chip and their speed of operation are increasing. In future *systems on chip (SoCs)*, communication between the cores will become a major bottleneck for system performance as current bus-based communication architectures will be inefficient in terms of throughput, latency, and power consumption [1]–[6].

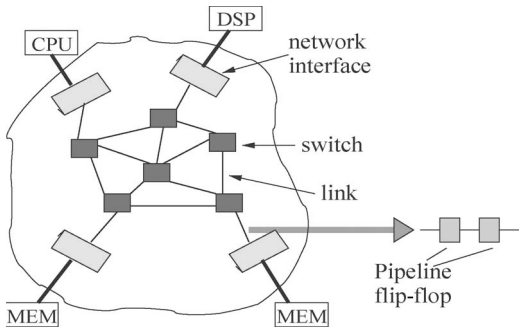

Fig. 1. Example NoC system with pipelined links.

Scaling of transistors is accompanied by a decrease in supply voltage and an increase in clock rate. This makes wires unreliable as the effect of various noise sources (such as crosstalk and coupling noise) increases. With technology scaling, the wire delay as a fraction of the total delay is increasing. The delay in crossing a chip diagonally for 45-nm technology is around 6–10 cycles, with only a small fraction of chip area (0.6%–1.4%) being reachable in a single clock cycle [7]. To effectively design future SoCs, *networks on chip (NoCs)*, a communication-centric design paradigm to counter the delay and reliability issues of wires, has been proposed [1]–[6].

A typical NoC consists of switches, links, and network interfaces (NIs), as shown in Fig. 1. An NI connects a core to the network and coordinates the transmission and reception of packets from/to the core. A packet is usually segmented into multiple *flow-control units (flits)*. The switches and links are used to connect the various cores and NIs together. To tackle the delay of long NoC links, a *latency-insensitive design* approach in which the links are pipelined can be utilized [8]. Link pipelining increases the link throughput and decouples the cycle time of the communication system from the link length.

Another effect of the *deep submicrometer (DSM)* technology is the significant delay variations across the wires. Wires are becoming thicker and taller, but their widths are not increasing proportionally, thereby increasing the effect of coupling capacitance on the delay of wires. As an example, the delay of a wire can vary between τ and $(1 + 4\lambda)\tau$ (where τ is the delay of the wire without any capacitive coupling, and λ is the ratio of the coupling capacitance to the bulk capacitance) [32]. The wire

Manuscript received July 26, 2005; revised April 16, 2006. This work was supported in part by the U.S. National Science Foundation under Contract CCR-0305718 and in part by a grant from STMicroelectronics to Dipartimento di Elettronica, Informatica e Sistemistica (DEIS). This paper was recommended by Associate Editor R. Gupta.

R. Tamhankar is with Marvell Semiconductors Inc., Santa Clara, CA 95054 USA (e-mail: rutu@marvell.com).

S. Murali and S. Stergiou are with the Computer Systems Laboratory (CSL), Stanford University, Stanford, CA 94305 USA (e-mail: smurali@stanford.edu; utopcell@stanford.edu).

A. Pullini is with the Politecnico di Torino, 10129 Turin, Italy (e-mail: apullini@deis.unibo.it).

F. Angiolini, and L. Benini are with the University of Bologna, 40136 Bologna, Italy (e-mail: fangiolini@deis.unibo.it; lbenini@deis.unibo.it).

G. De Micheli is with the École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland (e-mail: giovanni.demicheli@epfl.ch).

Digital Object Identifier 10.1109/TCAD.2007.891371

delay for data transfer on a communication bus depends on the data patterns transferred on the bus. As presented in [17], the data-dependent variations in wire delay can be as large as 50% for the different switching patterns. With technology scaling, the device characteristics fluctuate to a large extent due to process variations and can cause significant variations in wire delay [18]. Wire delay is also affected by other forms of interference such as supply bounce, transmission line effects, etc. [20], [21].

Current design methodologies are based on a worst case design approach that considers all the delay variations that can possibly occur due to the various noise sources and environmental effects and targets a *safe* operation of the system under all conditions. The system state is considered *safe* if there are no timing violations for all operating conditions and in the presence of the various noise sources. Such a conservative design approach targets timing-error-free operation of the system. In *Razor* [10], [11], an aggressive better-than-worst-case design approach was presented for processor pipelines. In such a design, the voltage margins that traditional methodologies require are eliminated, and the system is designed to dynamically detect and correct circuit timing errors that may occur when the worst case noise variations occur. *Dynamic voltage scaling (DVS)* is used along with the aggressive design methodology, allowing the system to operate robustly with minimum power consumption.

In this paper, we present timing-error-tolerant design methods (we refer to them as *T-error*) to aggressively design the NoC components (switches, links, and NIs) to support higher operating frequencies than designs based on conservative approaches. Aggressive design of the communication architecture has several implications when compared to the design of processor pipelines. First, the hardware overhead required to recover from timing errors can be minimized by smart utilization of the buffering resources available in the NoC. Second, the error recovery penalty can be mostly hidden under the network operation so that the large performance benefits can be obtained. Finally, the switches and NIs should be redesigned to handle errors, as they may receive a wrong piece of data before the right one.

In many SoCs, *dynamic frequency scaling (DFS)* and *dynamic power management (DPM)* policies are used to reduce the operating power of the SoC [41]. In such systems, at the application level, the voltage and frequency of the components are selected to match the performance level of the application. The NoC can also be dynamically tuned at runtime. When a communication-intensive application requires fast execution, the NoC can be overclocked to higher operating frequencies. When an application does not require a fast NoC, the frequency of the NoC can be lowered to reduce the power consumption of the system. Unlike many of the earlier works (refer to Section III), where the system's error rate is constantly monitored to tune the voltage or frequency, we envision our *T-error*-based NoCs to be utilized in systems with application-level DFS/DPM policies. Thus, complex network error-rate monitoring controllers are not needed in the design. Moreover, the large delay incurred to change the frequency/voltage to reduce errors is avoided. The required voltage and frequency parameters of the network for the different applications can be stored in programmable registers or memories and can be accessed by the operating system upon task switches among the applications that are running on the SoC.

In this context, we distinguish two possible operating modes for the NoC: *normal mode* and *overclocked mode*. In *normal mode*, the NoC operates at frequencies less than or equal to the frequency of a conservative design. Under *overclocked* mode, the frequency of operation can be higher than that of the traditional design. The NoC under the *overclocked* mode incurs some penalty for error resiliency even when there are no errors in the system (this is explained in detail in Section VI-B). Under *normal mode*, the NoC does not need to encounter the additional error resiliency penalty, as it operates at a *safe* operating frequency. To remove any additional overheads when in *normal mode*, we present a way to dynamically configure the NoC between the *normal* and *overclocked* modes of operation at the application level.

The *T-error* scheme for an NoC link is presented in [33]. In this paper, we present two robust link design methods. In the first scheme, link buffers are efficiently utilized, so that error resiliency is achieved without much additional hardware overhead. In the second scheme, more hardware resources are used to achieve higher performance. The two link schemes have the same timing relation and logic interpretation of control signals from/to the switch. The two schemes can be used in a *plug-and-play* fashion by the designer to suit the application and NoC architecture characteristics. We integrate the link designs with NoC flow control and present *T-error* schemes for switches/NIs.

We developed cycle-accurate SystemC models of the *T-error*-based switches, links, and NIs and integrated them onto the $\times$pipes NoC architecture [19]. Functional SystemC simulations on several benchmark applications have been carried out. Detailed case studies of the *T-error* design and comparisons with the traditional mechanisms are presented. Experiments show large performance improvements (up to 33% reduction in communication delay) for the benchmark applications for the aggressive NoC design methodologies when compared to traditional design methodologies. The application of DVS/DFS techniques result in 57% reduction in the NoC power consumption when compared to traditional design approaches.

II. Double Sampling Technique

In most NoC realizations, when errors are detected, corrupted packets are retransmitted. Unfortunately, retransmissions incur significant performance penalties [26]. Moreover, timing delay variations occurring due to higher operating frequencies can potentially affect multiple data bits in a packet, requiring complex multibit error-detecting/correcting codes that may be impractical to use [26], [42].

To recover from timing errors in a digital system, *double data sampling techniques* have been proposed and used by several researchers [10]–[16]. In such double sampling schemes, each pipeline flip-flop in the design (called *main flip-flop*) is augmented with an additional latch/flip-flop (called *delayed flip-flop*), as shown in Fig. 2. Both the *main* and the *delayed*

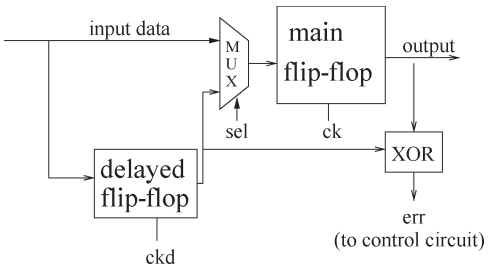

Fig. 2. Double data sampling technique.

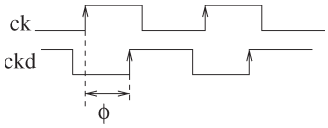

Fig. 3. Phase shift between clocks.

flip-flops have the same frequency of operation. However, the clock to the *delayed flip-flop* has a phase shift from the clock to the *main flip-flop*, and it samples data at delayed clock edge, as shown in Fig. 3.

Thus, the data sampled by the *delayed flip-flop* have more time to settle compared to the *main flip-flop*. The delayed clock is usually generated locally at the pipeline stage from the main clock using an inverter chain (delay element). After the *delayed flip-flop* has sampled data, the values of the two flip-flops are compared through an EXOR gate; if there is any difference, the data from the *delayed flip-flop* are assumed to be correct and are resent through the *main flip-flop* in the next clock cycle. The control circuitry also sends flow-control signals to the pipeline stages before and after the stage where the error occurred, so that they can recover from the error.

Let us consider a bit line of an NoC link with one pipeline stage, where the pipeline flip-flop (*main flip-flop*) is augmented with a *delayed flip-flop*. Let the maximum *safe* operating frequency of the link for the original design (without using any double sampling technique) be 1 GHz. If the double sampling technique is used, we can have a higher frequency of operation, as the link no longer needs to have a *safe* operation at the main flip-flop. As an example, if the delay or phase shift between the clocks to the main and delayed flip-flops [ϕ/(clock period) in Fig. 3] is 50%, the delayed flip-flop will sample the right data even when the link operates at 1.5 GHz. Even though the *main flip-flop* may incur timing errors, we can recover the right data from the *delayed flip-flop*.

Note that a higher operating frequency can also be achieved by having a deeper pipeline in the NoC components. However, there are several advantages in using the *T-error*-based design than having a deeper pipeline.

1) When the NoC is operating in the *normal mode*, a deeper pipeline depth will result in a fixed increase in latency across the link, while in the *T-error*-based scheme, this latency is avoided (in fact, *T-error* design can be viewed as a way to dynamically change the pipeline depth of the NoC components).
2) As the traditional design frequency is conservative, even in the *overclocked* mode, the errors introduced due to *overclocking* may not be substantial. Thus, the *T-error* design can achieve the same frequency of a deeply pipelined design with a lower latency for the average case. This is because, in the *T-error* design, the pipeline depth changes dynamically according to the error rate, while the deeply pipelined design always incurs a high latency.
3) Significant redesign, verification, and timing validation of switches and NIs are needed to increase the pipeline depth, while the *T-error* design can be incorporated with lower design efforts. The normal first-in–first-out (FIFO) buffers used in the links, switches, and NIs need to be replaced by the *T-error* FIFOs, which can be designed and used as library elements.
4) *T-error* can always be used as an add-on to a deeply pipelined NoC system to improve the operating frequency of the system.

In this paper, we present the methods that address only the timing delay variations on the NoC that are introduced due to overclocking. Coping with other kinds of errors (such as soft errors, capacitive coupling-based crosstalk, data upsets, etc.) is assumed to be done by means of existing techniques (such as [22]–[31]). By operating the NoC at higher frequencies, the effect of these errors on the system may vary, and we assume that the techniques used to address them are designed to handle the maximum overclocked frequency of operation.

III. Previous Work

Several researchers have motivated the need for NoC-based designs [1]–[6]. In order to cope with the long link delays, a *latency-insensitive* design approach is presented in [8] and [9]. The use of repeaters to store data in an asynchronous link design is presented in [35]. The buffering mechanism on the wires in the scheme is asynchronously controlled by the receiver and is not integrated with the switches of the NoC. The use of FIFOs for data queuing is widely spread in many globally asynchronous, locally synchronous design methodologies such as [36].

Reliability and tolerance against timing errors on on-chip buses and NoCs have been addressed by many research works. The use of routing algorithms that decrease the probability of system failures has been explored in [37] and [39]. The use of coding techniques for detecting and correcting on-chip communication errors has been analyzed in [22]–[26]. In most of these coding schemes, once an error is detected, retransmission of the data is required. Worm *et al.* [38] present a method to monitor the error rate on the links to dynamically vary the supply voltage and reduce power dissipation. Li *et al.* [28] monitor the data bus to detect adverse switching patterns (that increase the wire delay) and change the clock timing to avoid timing errors on the bus. Many low-power bus-encoding techniques have been presented to reduce the power consumption of buses [27]. The use of bus encoding to reduce crosstalk between wires and avoid adversarial switching patterns on the bus has been presented in [29]–[31]. These works complement the work presented in this paper.

There have been several approaches in the design space to detect and correct timing errors. The use of *double data*

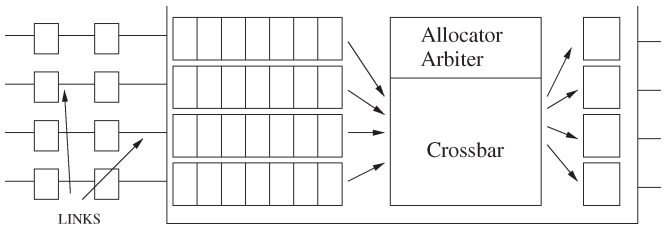

Fig. 4. Input-queued switch.

sampling techniques has been shown in self-checking testing circuits [13] and for clock recovery in digital systems [16]. Recently, these techniques have been used for online timing and soft error recovery in systems. The *TEAtime* [12] architecture tracks logic delay variations and dynamically adjusts the clock frequency to accommodate the changes in logic delay. In *Razor* [10], [11], an aggressive better-than-worst-case design approach is presented for processor pipelines. In this paper, *double sampling* of data is used to control the supply voltage (and hence power consumption) by monitoring the error rate. Favalli and Metra [13] assume an encoded data signal which is checked by a small decoder present at the input of each flip-flop. In case of an error, the clock is delayed for 1 cycle until the correct value of data settles. *Mousetrap* [14] is a high-speed asynchronous pipeline which ensures correct data availability to consecutive stages. The *Iroc* [15] design uses a latch with delayed clock to detect transient faults due to soft errors.

IV. Using Links as a Storage Medium

Flow control is needed in networks to support full throughput operation. Specifically, it is needed to ensure that enough buffering is available at each switch to store the incoming data, and the available buffers are utilized efficiently. In traditional designs, queuing buffers are either located at the inputs (*input-queued* switches) or at the outputs (*output-queued* switches). In some switches, the buffers can be located at both the inputs and the outputs to improve the performance of the NoC [34]. A *credit-based* or *on/off flow-control* mechanism is typically used to manage the input buffers of the switch. In such designs, for maximum network throughput, the number of queuing buffers needed at each input of the switch should be at least $2N + 1$ flits [34], where N is the number of cycles needed to cross the link between adjacent switches. This is because, in credit-based flow control, it takes at least 1 cycle to generate a credit, N cycles for the credit to reach the preceding switch, and N cycles for a flit to reach the switch from the preceding switch [34]. To support link pipelining, there need to be $N - 1$ pipeline buffers on each bit line of the link connecting the switches. Thus, effectively, we need $3N$ flit buffers for each input of the switch/link (Fig. 4).

In [8] and [9], the use of relay stations and link-level flow control has been presented. In such a scheme, each pipeline flip-flop on the link is replaced by a two-entry FIFO, and a link-level flow control is used to ensure full throughput operation. We utilize such links for our NoC architecture. In our NoC architecture, the switch input buffers are also replaced by a two-entry FIFO. Fig. 5 shows a three-stage link pipeline using

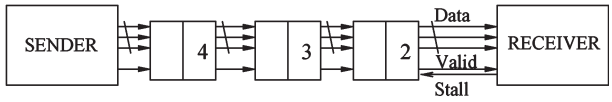

Fig. 5. Modified link design with three stages.

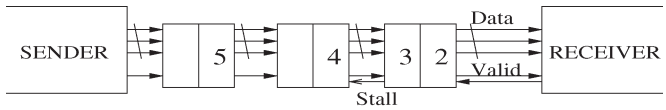

Fig. 6. Entry 3 buffered in *secondary flip-flop*.

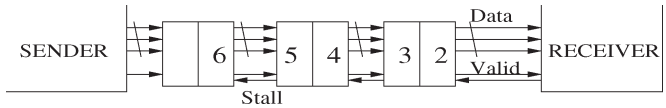

Fig. 7. *Stall* signal propagated to previous stage.

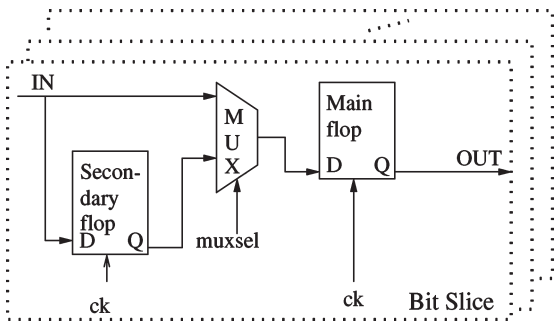

Fig. 8. Two-entry FIFO. The control circuit is common for all the bit lines.

two-entry FIFO at each pipeline stage ($N = 4$, as it takes 1 more cycle to reach the receiver from the last pipeline stage of the link). The scheme has two control signals (*stall* and *valid*) transmitted between sender, receiver, and the link pipeline stages. The *stall* signal is sent by the receiver and flows in the opposite direction to that of the data, while the *valid* signal is driven by the sender, and it flows in the same direction as that of the data. The sender or receiver may be a switch or an NI. The receiver generates a *stall* signal when its storage capacity is full or if it receives a stall request from the following stage. The *valid* signal informs that the data which were received in the previous cycle (at the previous rising edge of clock *ck*) is valid. During normal operation (i.e., when there is no stall request), only one of the flip-flops in the two-entry FIFO is used, as shown in Fig. 5. When a *stall* signal is received by the two-entry FIFO (shown in Fig. 6), the data on the output of the *main flip-flop* are stalled, and new data are received by the secondary flip-flop. The *stall* signal is propagated to the previous stage, as shown in Fig. 7. The schematic of the two-entry FIFO is shown in Fig. 8.

This flow-control mechanism ensures full throughput operation with performance similar to that of the *input-queued* switches with credit-based or ON/OFF flow control. As previously shown, in traditional *input-queued* schemes (Fig. 4), the total number of buffers needed for maximum throughput is $3N$, as compared to only $2N$ buffers [$2 \times (N - 1)$ along the link and two at the switch input] in this scheme (Fig. 9). The traditional *input-queued* design has one flip-flop at each link

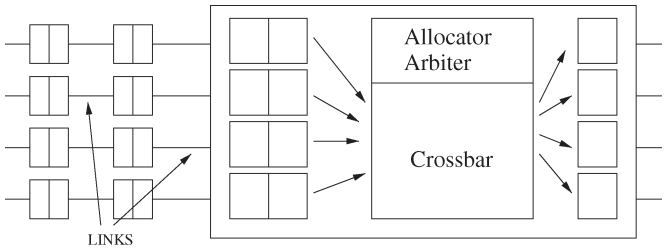

Fig. 9. Modified link and switch design.

pipeline stage. In the *stall/valid* protocol, it takes 1 clock cycle for the *stall* signal to reach the preceding pipeline stage. During this time, the data which are in transit from the preceding pipeline stage cannot be stored when it reaches the current pipeline stage. Thus, for full throughput operation in such a scheme, the link flip-flops are not used for queuing data and, instead, the data are queued at the input of the next switch. By augmenting the link pipeline stage with one more flip-flop, the full throughput operation is achieved. As we also utilize the pipeline flip-flops, the scheme leads to reduced buffering requirements. As the link buffering scheme can be viewed as merely spreading the FIFO buffers of the switch inputs onto the links, it maintains the same deadlock and livelock properties of a design with *input-queued* switches. Moreover, as all the inputs of a switch have the same buffer count in the link-buffer scheme, the switch design becomes more modular when compared to the traditional switch design. Note that the control circuit used at a link pipeline stage in this scheme is common for all the w data bits in a flit of the NoC, and thus the overall cost of the control circuit is negligible.

V. *T-Error* Link Designs

In this section, we present two link designs to support timing-error-tolerant operation needed for overclocking the links. The first design reuses the link FIFO for error recovery with very little hardware overhead (the overhead is only for the control circuitry). This scheme, in the worst case, can incur a 1-cycle penalty for each error occurrence at a pipeline stage. In the second link design scheme, the two-entry FIFOs are augmented with an additional flip-flop. The resulting design is a high-performance link that incurs a 1-cycle penalty only for the first occurrence of an error for a continuous stream of data at each pipeline stage. The design is such that all subsequent errors are automatically resolved.

A. Scheme 1: Low Overhead T-Error Links

In the *T-error* scheme, the two-entry FIFOs along the links are modified to support timing-error-tolerant operation. The modified FIFO structure is shown in Fig. 10. The second flip-flop of the FIFO is clocked at a delayed clock (*ckd*) compared to the clock *ck* of the *main flip-flop*. *ckd* and *ck*, however, feature the same period. The phase shift among them is configured after proper delay analysis, as will be discussed later.

The incoming data are sampled twice: once by the *main flip-flop* (at time instant t_0 in Fig. 11) and then by the *delayed flip-flop* (at time instant t_1). There are two modes of operation at

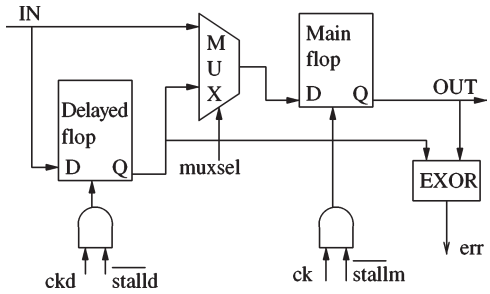

Fig. 10. Low overhead T-error buffer.

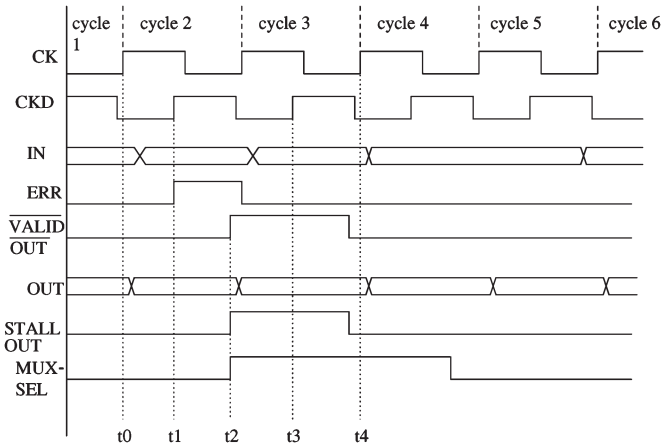

Fig. 11. Waveforms for scheme 1.

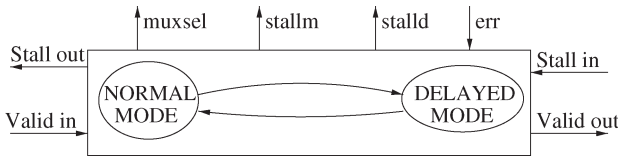

Fig. 12. Control circuit for scheme 1 (presented in Fig. 10).

each pipeline stage of the link: *main mode* and *delayed mode*. Initially, all the pipeline FIFOs are set to the main mode, and data transmission begins. In every cycle, at the clock edge *ck*, the *main flip-flop* captures and transmits the incoming data. At clock edge *ckd*, the *delayed flip-flop* captures the incoming data, and the error-detection control circuit checks whether there is any difference between the main and the *delayed flip-flop* values. As shown in Fig. 10, an EXOR gate is connected to the outputs of the *main flip-flop* and *delayed flip-flop* to detect a timing error. The err signals of all w bits of the flit (vertically across the width of the link) at a pipeline stage are ORed and fed as an input to the control circuit. Thus, a timing error in any bit of the flit causes the entire flit to be resampled at the pipeline stage. The control circuit at each pipeline stage, which is common for all the bit lines of the link, is presented in Fig. 12.

If there is an error in the data sampled by the *main flip-flop*, the data that were transmitted at clock edge CK are incorrect. The correct data from the *delayed flip-flop* are sent at the next clock edge (at time instant t_2). Whenever a timing error occurs (i.e., *err* signal is set to one), a *stall* signal is sent to the previous stage such that the previous stage is stalled for 1 cycle. Also, a

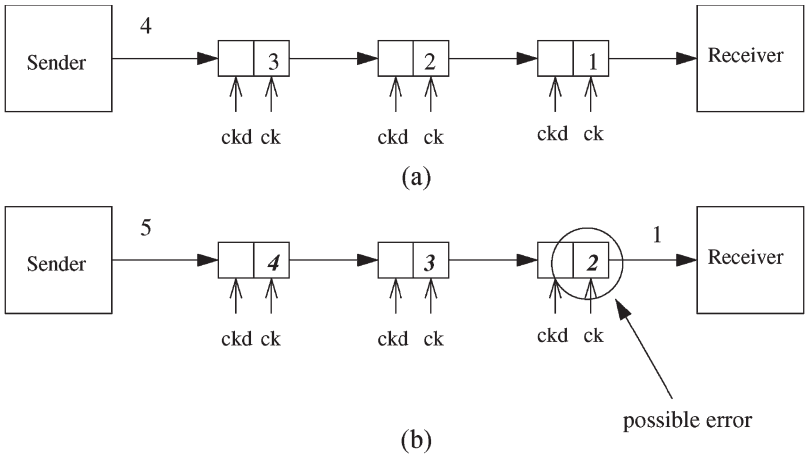

Fig. 13. Network operation without congestion. The data in the FIFOs at time instances t and $(t+1)$ are presented in (a) and (b).

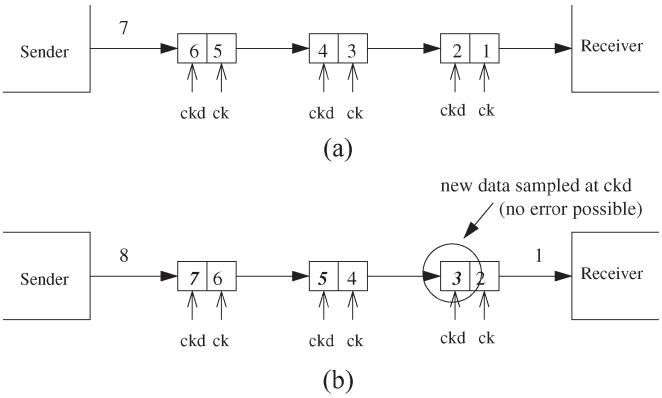

Fig. 14. Network operation under congestion. The data in the FIFOs at time instances t and $(t+1)$ are presented in (a) and (b).

valid signal is sent to the following stage, informing that the data sent in the previous cycle were nonvalid.

A FIFO at a pipeline stage of the link enters the *delayed mode* when a *stall* signal from the next stage causes queuing of data at the FIFO. The *stall* signal can be issued to handle regular congestion, that is as a flow-control wire, or to let the downstream stage sort out an error condition. When a FIFO is in a *delayed mode*, all timing errors are automatically avoided, as the incoming data are always sampled through the *delayed flip-flop*. Thus, in networks with severe congestion, most timing errors are automatically avoided. Examples of operation of the FIFOs for a network with no congestion and with congestion are presented in Figs. 13 and 14. In the network with no congestion, at each pipeline stage, the data are always directly sampled by the *main flip-flop* and sent out by it. In the network with congestion, the data from the preceding pipeline stage is always captured by the *delayed flip-flop* at the current pipeline stage and later sent out by the *main flip-flop*. Since the data are always sent at *ck* from the preceding stage and sampled at *ckd* in the current stage, the wire transitions have more than one clock period to settle and, thus, timing errors are automatically avoided. In the worst case, if the FIFO always operates in the *main mode*, each timing error occurrence will incur one clock cycle penalty for recovery.

However, in the worst case, when there is no congestion and the FIFO always tries to operate in *main mode*, each timing error occurrence incurs one clock cycle penalty for recovery. The link stage switches from *main mode* to *delayed mode* and back for each faulty piece of data. Detailed performance analysis of this scheme and comparison with the next link design scheme for several benchmark applications are presented in Section VIII-F.

The amount of timing delay that is tolerated by the *T-error* design depends on the phase shift between the clocks of the *main* and the *delayed flip-flops*. This shift should be as large as possible, so that the *delayed flip-flop* is guaranteed to sample the right data and to provide correct system operation. However, the maximum shift is constrained by internal repeater delays (the error-detection logic must operate between a *ckd* edge and the following *ck* edge). Detailed timing analysis and SPICE simulations (for a link size of 32 bs) showed that clock *ckd* can be delayed by 53.3% of the clock period with respect to *ck*. In this paper, we assume that a maximum delay of 50% of the clock is tolerable with a *T-error*-enabled system. Thus, the delayed clock *ckd* is just the inverted value of the main clock, and the delay chains are not needed to generate it. At the same time, the maximum delay which is tolerated on a wire is 150% of the clock period, providing ample margin for timing error correction. In the *T-error* scheme, metastability conditions may occur and are corrected using efficient transistor-level implementation of the FIFO circuit, which are presented in [33]. The control lines (*stall* and *valid*) that need to have error-free operation can be made robust using a variety of methods (such as using wider metal lines and shielding). We refer the interested reader to [33] for transistor-level implementation details, timing analysis, and SPICE simulation results of the *T-error* scheme.

B. Scheme 2: High-Performance T-Error Links

The performance of the above link design can be improved by having an additional flip-flop to store incoming data whenever a stall is encountered. A three-entry FIFO, instead of the two-entry FIFO previously described, is used in this scheme (refer to Figs. 15 and 16). The third flip-flop, called *auxiliary*

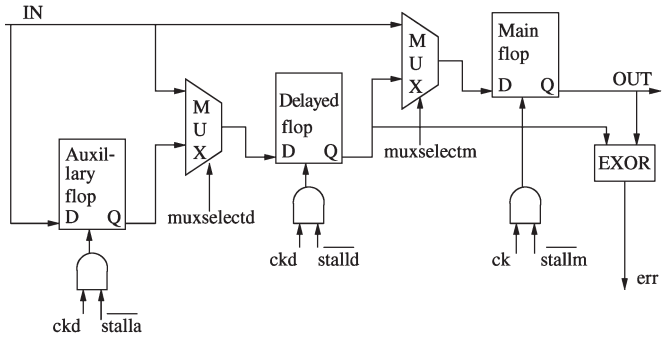

Fig. 15. Schematic for scheme 2.

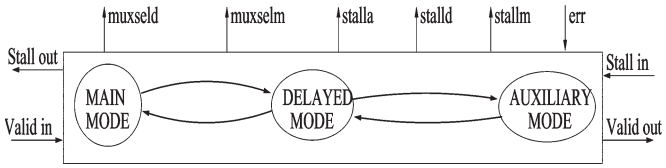

Fig. 16. Control circuit for scheme 2 (presented in Fig. 15).

flip-flop, is added in series to the *delayed* and *main flip-flops*; it also samples data on rising edges of the delayed clock *ckd*. The operation is similar to the above design, except that for a continuous stream of data, even if all incoming pieces of data were to be corrupted, only a single 1-cycle penalty would be incurred to correct timing errors at a pipeline stage. This is because the FIFO enters the *delayed mode* upon the first error occurrence; once in this mode, all subsequent pieces of data are sampled through the *delayed* and *auxiliary flip-flops*, making them automatically error free. The presence of the *auxiliary flip-flop* lets the link stage continue operating even upon fault occurrences; the sender does not perceive any interruption in the data flow. Only at the end of the whole data stream, the stage empties and switches back to *main mode*. An example is presented in Fig. 17. Note that even in the absence of timing errors, the *auxiliary flip-flops* can still improve general system performance, as they also behave as queuing buffers to minimize congestion-related penalties.

VI. Aggressive Switch/NI Design

We assume that the basic design of the switches and NIs is based on the $\times pipes$ building blocks [19]. In this section, we describe the changes needed in the basic architecture to support the *overclocked* mode of operation. The $\times$pipes NI are composed of two modules: a front-end interface with the cores and a back-end interface with the switches and links. The NI back-end is the only part that needs to support NoC overclocking. Since its architecture is similar to that of the switches, we describe only the changes required in the switches.

There are two changes required in the switches to support NoC overclocking. The first is that the switches should also be able to operate at higher frequencies to utilize the faster links. The other is that the switches should be able to handle the data from the links that may have timing errors. An NoC switch, as shown in Fig. 9, consists of input buffers, allocator/arbiter, crossbar, and output buffers. In our link-based flow-control, there is a two-entry FIFO at the input of the switch, which can be made timing-error tolerant, similar to the link FIFO *T-error* schemes presented in the previous section. The switch design changes will now be presented.

A. Output Buffer Changes

In an *input-queued* switch, normally, a single register is used at each output to store data before sending the data onto the links. Note that in some designs, the output buffer can be taken to be part of the link design, depending on the targeted operating frequency of the switch. In some other cases, more than one buffer may be used at each output, so that the performance of the NoC can be improved. In the $\times$pipes architecture, the number of buffers at the output is a parameter that can be configured by the user according to his or her application needs.

As a starting point, the architecture of a $\times$pipes switch with a single output buffer is shown in Fig. 9. The $\times$pipes switch already supports distributed buffering along the links. In this architecture, the switch has a latency of 2 cycles for data transfers. There are two sets of flip-flops in the switch that may cause timing violations when overclocked: output buffers and flip-flops that are used to maintain the allocator/arbiter states. From synthesis of the $\times$pipes architecture, we found the operating frequency of the original switch to be 1 GHz. The path from the input of the switch to the state flip-flops was 0.4 ns, while the critical path was from the input to the output (which also samples the arbiter/allocator states). With overclocking, we target a 1.5$\times$ increase in frequency (i.e., 1.5-GHz operating frequency) of the switches. Therefore, we found that the state flip-flops are *safe* even under overclocking, since the available cycle time is 0.66 ns, and that only the output buffers need to be made timing-error tolerant. Note that in other switch architectures, if the state flip-flops are not safe when overclocked, they should be *T-error* enabled as well. Otherwise, the amount of overclocking will be limited by them. Also, if the switch has more pipeline stages, the *T-error* principle needs to be applied to each pipeline stage.

In order to overclock the switch, we apply the *T-error* design to the head flip-flop of the output FIFO, and the other flip-flops in the output FIFO are made to sample data at *ckd*. Fig. 18 shows the changes in the output buffer of the switch. Note that errors can occur only when the data are sampled through the head of the FIFO and when the NoC operates in the *overclocked* mode.

B. Input Buffer Changes

When timing errors occur at a link pipeline stage, wrong data can reach the switch input before the correct data are received. If the switch samples wrong data, several complications can arise. As an example, timing errors on the routing fields of the header flit may result in misrouting a packet. In order for the switch to handle data errors, there are several cases to be considered, and recovering the switch state from such cases requires complex hardware and control circuits [34]. Another way to detect wrong data at the switch input is to use some error-detecting code (such as cyclic redundancy check) for each

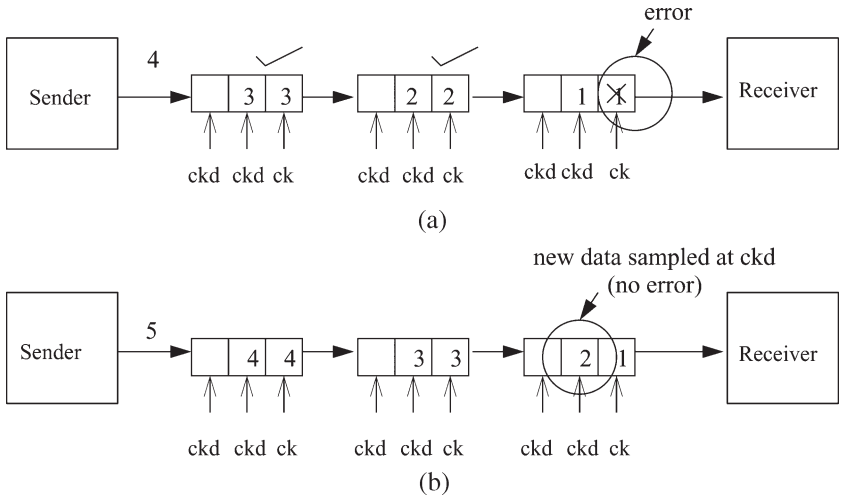

Fig. 17. Example of three-entry FIFO operation, where for a continuous stream of data, an error occurrence at a pipeline stage causes further errors to be automatically avoided at that stage. (a) At time instant t, clock edge ckd. (b) At time instant $(t+1)$, clock edge ckd.

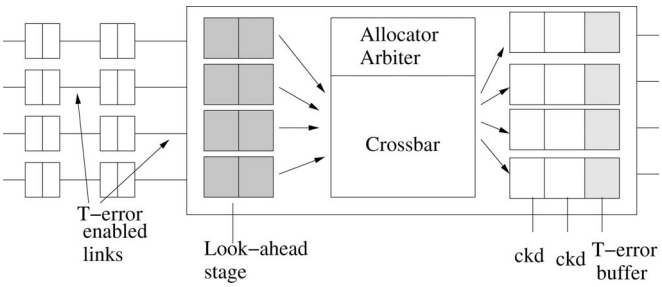

Fig. 18. Overclocked switch design with output and input buffer changes.

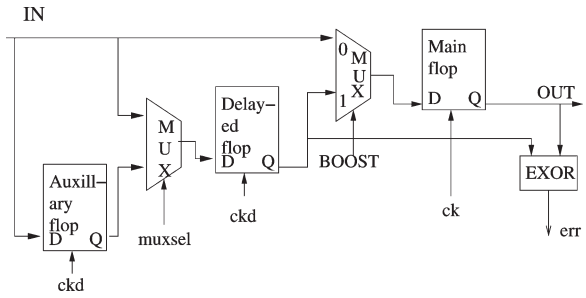

Fig. 19. *Look-ahead stage* at the switch input.

flit of the packet. However, in the *overclocked* mode, all the bits of the data could encounter timing errors, and such schemes may be inefficient. Thus, to simplify the switch hardware, we use a look-ahead stage at the input of the switch that ensures that correct data are always fed to the internal switch logic (see again Fig. 18). The look-ahead stage stores an incoming flit for 1 clock cycle, i.e., until the *valid* line indicates whether the received data were correct or not. In case of a correct reception, data are fed to the switch arbiter/allocator. Otherwise, it is discarded by the look-ahead stage. Note that even when there are no errors occurring in the system, a latency penalty could arise from insertion of the look-ahead buffers, unless properly tackled, as explained in the next section.

VII. Dynamic Configuration of the NoC

When the frequency of the NoC is varied based upon DFS/DPM techniques, the NoC may operate at frequencies lower than or equal to the conservative design frequency. In such a *normal operating mode*, the error resiliency penalty due to *T-error* needs to be completely hidden. The *T-error* mechanism at the link FIFO and the switch/NI output buffers incur error resilience penalty only when an error occurs. Thus, they dynamically adjust to the errors happening in the system. However, the look-ahead stage at the input of the switch incurs a 1-cycle penalty even under the *normal* operating mode. To avoid this 1-cycle penalty in the *normal mode*, we use a global *BOOST* signal that is issued at the application level by the (one or more) processing cores. A value of $BOOST = 1$ indicates that the NoC is in *overclocked* mode, while $BOOST = 0$ indicates *normal mode* of operation. The *BOOST* signal may take several clock cycles (tens of cycles) to spread to all the switches and NIs in the NoC. The actual transition between the *normal* and *overclocked* modes occurs after the *BOOST* signal is completely spread around the NoC.

The input buffer control logic is modified such that the *look-ahead stage* is used only when $BOOST = 1$, as shown in Fig. 19. The transition from the *normal mode* to *overclocked* mode is smooth in the design, as the look-ahead is started when the *BOOST* signal is spread. However, transition from the *overclocked* mode to the *normal mode* requires special care, as there may be some residual errors in the NoC. To make a smooth transition dynamically (i.e., without flushing all the data in the network), we use the following design change. In the *T-error* NoC, all residual errors are maintained on the links between the switches, as the switches always receive the right data due to the look-ahead mechanism. When a transition to the *normal mode* occurs, the look-ahead stage is bypassed only when there are no incoming data from the link. Thus, any data from the output buffer of the switch or the link that may have residual errors goes through the look-ahead stage, which ensures that the right data are fed to the switch inputs. As the transitions between *normal* and *overclocked* modes occur at the application level (which may occur every tens of thousands of

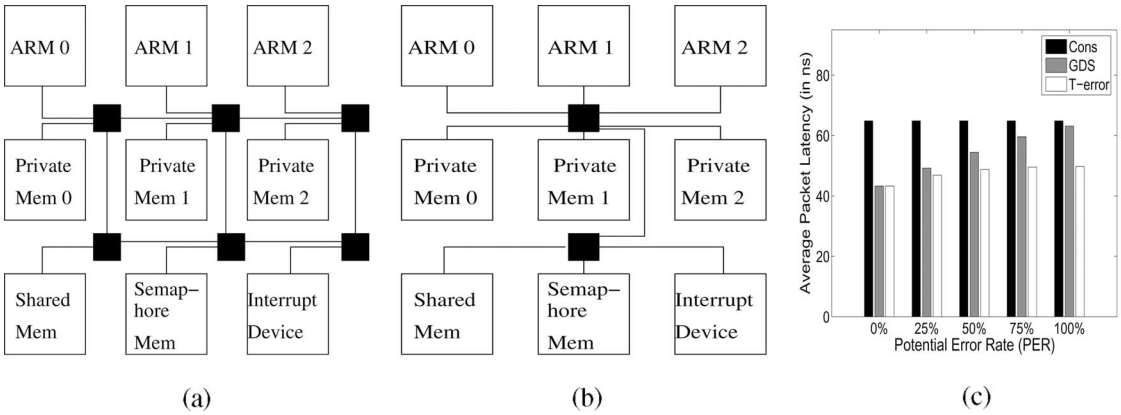

Fig. 20. Mesh and custom-topology mappings and comparison of traditional schemes with *T-error*. (a) Mesh topology. (b) Custom topology. (c) Mesh topology results.

cycles), the performance overhead incurred due to this dynamic configuration is negligible.

VIII. Experimental Results

In this section, we present the simulation case studies for the *T-error* designs.

A. Simulation Platform

The simulation platform consists of cycle-accurate SystemC models of the *T-error* designs for the switches, links, and NIs incorporated on the ×pipes architecture. We use the MPARM simulation environment [40], which allows several interconnect structures (such as *advanced microcontroller bus architecture*, *STBus*, and ×*pipes*) to be utilized to connect processor/memory cores and which has support for a variety of benchmark applications. Functional SystemC simulations were carried out on a variety of application benchmarks.

B. Experiments on a Multimedia Benchmark

We plugged three ARM7 processors, three private memories (one for each processor), and three shared memories for interprocessor communication on the MPARM platform. We ran functional benchmarks modeling multimedia processing on the general-purpose cores. The benchmarks include heavy synchronization activity through the shared memories, since they model producer/consumer pipelines of multimedia processing. The benchmarks create a large number of connections (around 30) between the various cores. We hand-mapped the application onto two topologies [Fig. 20(a) and (b)]: a 3×2 mesh topology with the processors connected to their private memories using a single switch and a custom topology with two switches. The mappings were performed such that the most demanding traffic flows traverse fewer switches in the NoC.

We assume the size of each predesigned processor and memory core to be 2×2 mm, typical of today's small processors and on-chip memories. From the approximate floorplans of the topologies, we conservatively assume that the links of the mesh topology have one pipeline stage, while those of the custom topology have two pipeline stages.

We perform experiments on three schemes: a traditional *conservative (CONS)* design approach, a *general double-sampling (GDS)* scheme that is not integrated with the network flow control (such as those presented in the earlier works in Section III), and the *T-error* scheme with three-stage FIFO presented in this paper. From synthesis of the original ×pipes architecture, the conservative NoC's maximum operating frequency is found to be 1 GHz. With 50% delay between the clocks to the main and *delayed flip-flops*, the *GDS* and *T-error* designs' maximum frequency (under *overclocked mode*) is assumed to be 1.5 GHz. To evaluate the designs, we define a new metric: *potential error rate (PER)*. The *PER* represents the percentage chance that a flit reaching a FIFO incurs one or more timing errors if sampled directly on a *ck* edge. Note that even if the *PER* is 100%, the actual errors happening at the *T-error* FIFO can be very few, as most of the errors after the first are automatically avoided by the design. This is because, in most scenarios, the data are sampled first by the *delayed flip-flop* and only afterward sent out by the *main flip-flop*, avoiding all potential errors. For an *overclocked* system, the *PER* value depends on how much the system is *overclocked*, the actual operating conditions of the system (such as effect of process variations on the FIFO, operating temperature, and other noise effects), actual data patterns on the link, etc. As an example, if the bus-encoding techniques are not used to reduce the effects of capacitive crosstalk, the conservative design is capable of operating with the worst case data patterns on the links. In such a case, even at the highest frequency in the *overclocked* mode, if the adversarial switching patterns do not occur on the link, the *PER* can be 0%. The *T-error* design dynamically adapts to all these effects and operates under the entire range of *PER* values. For simulations, we vary the *PER* values, and we inject potential errors at each *T-error* FIFO randomly based on the chosen *PER* value.

The average packet latency for the mesh and custom topologies for the various schemes for different PER values are presented in Figs. 20(c) and 21. As we *overclock* only the communication architecture, we compare the schemes based on the average packet latency for communication, instead of comparing the total application run time. When compared to

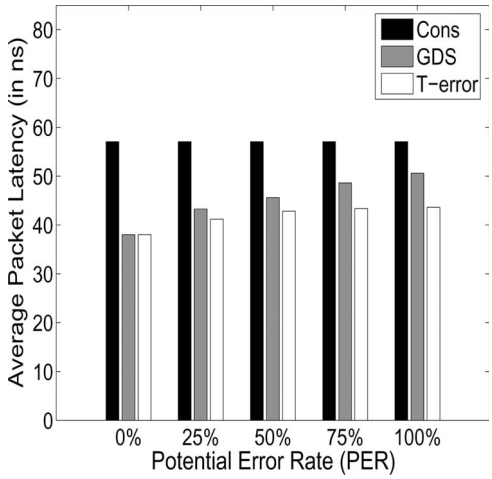

Fig. 21. Custom-topology results.

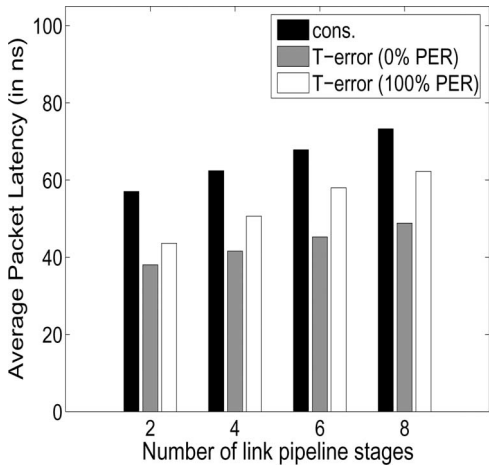

Fig. 22. Effect of pipeline depth.

the traditional CONS design, the *T-error* design results in significant performance improvements. Latency is reduced by 33.33% in the best case (0% *PER*) and by 23.42% in the worst case (100% *PER*). When compared to the general GDS scheme, the *T-error* scheme still shows up to 21.2% reduction in latency, as much of the error recover penalty is hidden under the network operation. When compared to the GDS technique applied to input-queued switches, the *T-error* scheme (with three-stage FIFOs at the links) also results in 30% reduction in the number of queuing buffers used. In fact, the three-entry *T-error* FIFO scheme utilizes $3 \times (N-1)$ buffers on each link (where N is the number of cycles needed to traverse the link) and two buffers at the switch input, while the input queued switches with the general double sampling technique need $2N+1$ buffers at the input of the switch and $2 \times (N-1)$ buffers on the links (refer to Section IV, where results for the two-entry FIFOs are presented).

To see the impact of the length of the links on the *T-error* scheme, we simulated the design mapped onto the custom topology with varying number of pipeline stages on the links. As shown in Fig. 22, even on significantly long links, the *T-error* scheme gives a large improvement in performance when compared to the conservative design approach.

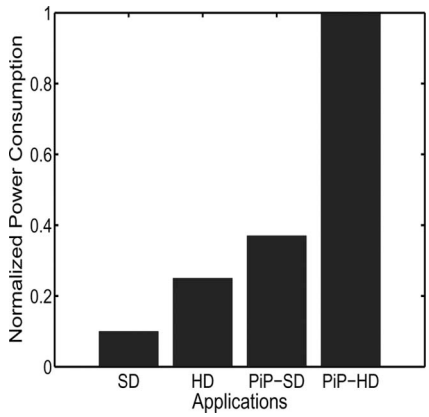

Fig. 23. Effect of DPM policies.

C. Effect of Application-Level Power Management

We conducted experiments on our multimedia benchmarks to show the usefulness of the application-level DPM policies. We model four different application scenarios in the platform: *standard-definition (SD)* video decoding and display, *high-definition (HD)* video decoding and display, *picture-in-picture standard definition (PiP-SD)*, and *picture-in-picture high definition (PiP-HD)*. The voltage and frequency of operation of the network was tuned individually for each application. The power consumption of the network for the various applications when the DPM policies are used, normalized with respect to that of the base system (where no DPM policy is used), is presented in Fig. 23. The use of application-level DPM policies results in an average of 57% reduction in power consumption of the NoC. The *T-error* scheme can be integrated into the work presented in [41], where several application modes are defined for industrial SoC designs, and where the NoC operating frequency and voltage are tuned individually for each application mode in the design.

D. Experiments on Other Benchmarks

We performed experiments on the conservative and *T-error* designs on several other benchmarks which are as follows:

1) matrix multiplication benchmark suite without shared memory (MAT1);
2) matrix multiplication benchmark suite with shared memory (MAT2);
3) fast Fourier transform benchmark suite using fixed point arithmetic (FFT); and
4) quick sort benchmark suite (*Qsort*).

Many of these benchmarks are application kernels that can be used to inject different traffic rates onto the NoC and test various aspects of the NoC. We assume the delay to traverse the links in the NoC to be 2 cycles, i.e., the links have two pipeline stages. We conducted experiments varying the number of processor/memory cores used by the applications (application partitioning) and topologies of the NoC. For all the experiments, except for those presented in Section VIII-F, we use the three-entry *T-error* FIFO design. In Section VIII-F, we compare the performance of the two *T-error* link designs.

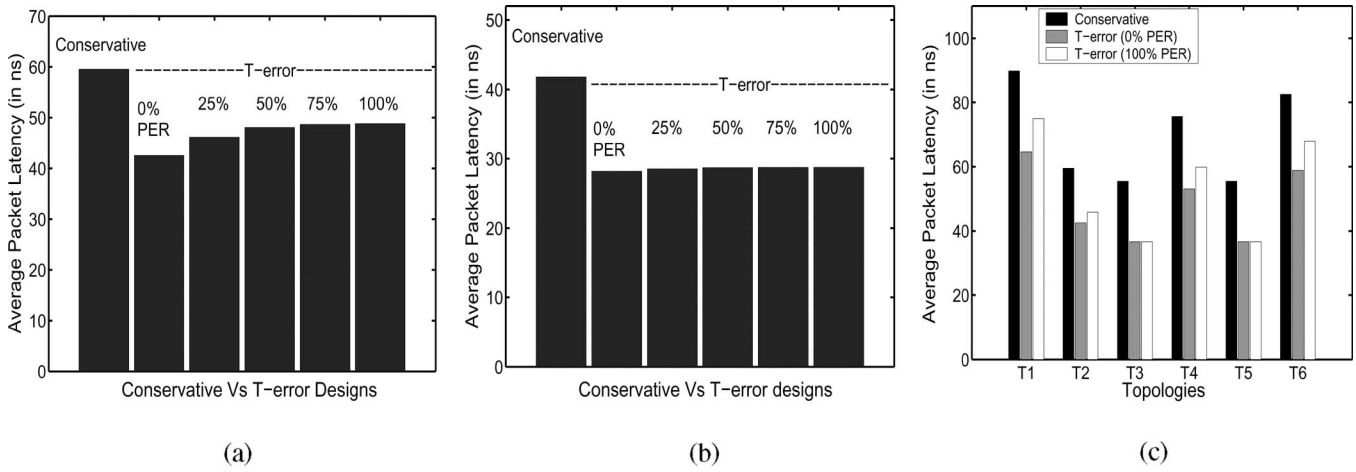

Fig. 24. Performance comparison of conservative and *T-error* designs for different *PER* values for read and write transactions. (a) MAT2: read transactions. (b) MAT2: write transactions. (c) Topology effects: read transactions.

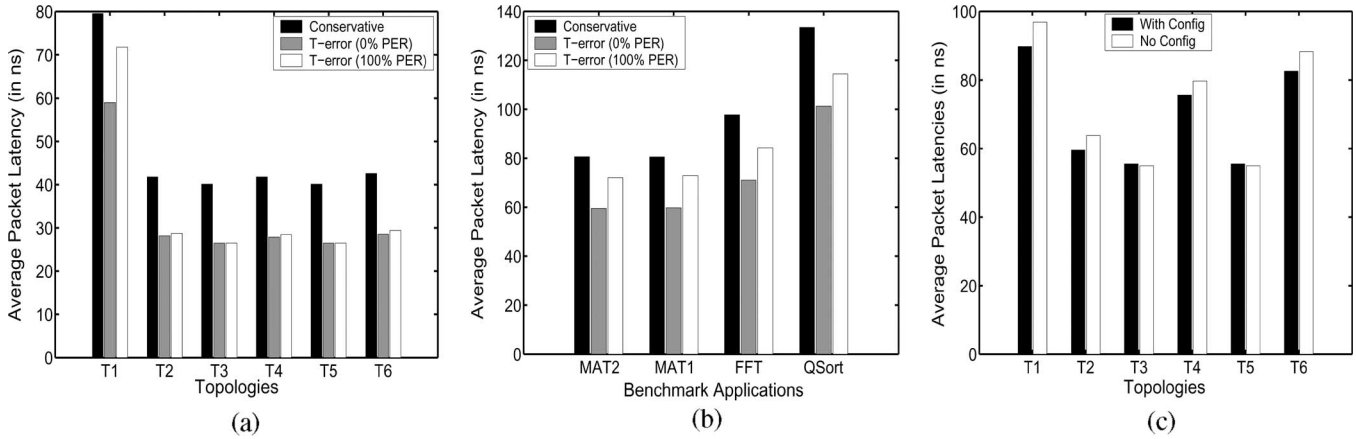

Fig. 25. (a) and (b) Performance comparison for various topologies and benchmarks. (c) Effect of dynamic NoC configuration.

In Fig. 24(a) and (b), the average packet latency (in nanosecond) observed for the conservative and *T-error* design for the *MAT2* benchmark for read [Fig. 24(a)] and write transactions [Fig. 24(b)] is presented. The read transactions require two-way data transfer on the network: a request is sent by the processor, and a response with the data item is sent back by the memory. The write transactions require only one-way data transfer: The processor sends the data to be written to the memory. We denote the entire transaction latency for each data word by the average packet latency metric. Thus, the read transactions incur a higher latency for communication. As shown in the figures, for the *MAT2* benchmark, the *T-error* design results in a significant performance improvement, with the best case of 28.5% reduction in read latency (for 0% *PER*) and worst case of 19.6% (for 100% *PER*). For the write transactions, the average reduction in latency for the *T-error* designs varies from 32.5% (for 0% *PER*) to 31.1% (for 100% *PER*). Note that the increase in latency due to the higher PER values is not overly significant, showing that the *T-error* scheme effectively hides much of the error recovery penalty under the network operation.

The performances of the *T-error* system for various topologies for the *MAT2* benchmark for read and write transactions are presented in Figs. 24(c) and 25(a). The designs compared vary from small seven-core NoCs to 51-core NoCs with different application partitioning. The topologies vary from regular (like *mesh*) to custom, which are manually developed ones. As shown in the figures, for all the topologies for both read and write transactions, the *T-error* design results in significant performance improvement over the conservative design. In Fig. 25(b), we present the average packet latencies (averaged across both read and write transactions) for the designs for several benchmark applications. The average reduction in the latency for the benchmarks for the *T-error* designs varies from 25.7% (for 0% *PER*) to 12.7% (for 100% *PER*).

E. Effect of NoC Configuration

Dynamic configuration of the NoC is designed to avoid any latency penalty for the switch look-ahead mechanism under the *normal mode*, where the frequency of operation is ≤ 1 GHz. In Figs. 25(c) and 26(a), we present the packet latencies for the NoC with and without the configuration mechanism for various topologies and benchmarks. The configuration mechanism results in significant reduction in the packet latency (up to 13.8%) for the applications. This reduction is attributed to two reasons: One is the reduction in the pipeline depth of the

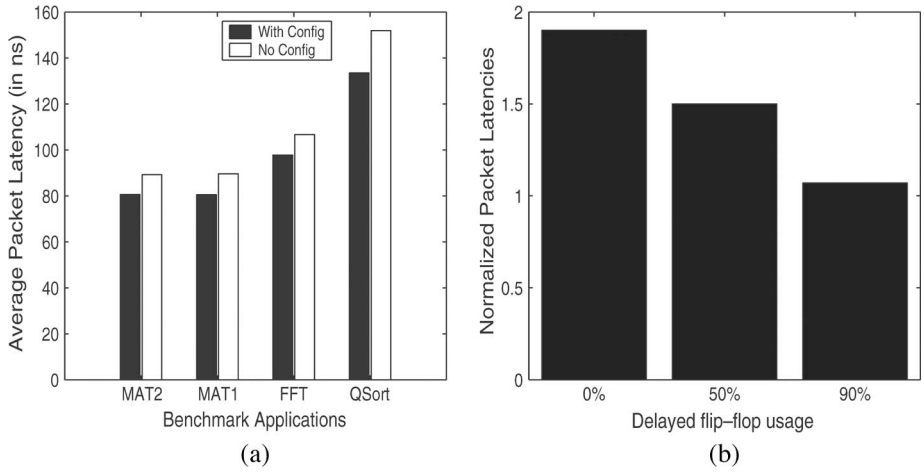

Fig. 26. (a) NoC configuration. (b) Choice of link design schemes.

NoC (i.e., reduction in the number of cycles needed to transfer a packet under zero load conditions), and the other is the fact that congestion in the NoC reduces as the packets spend less time in the network.

F. Choice of Link Design Schemes

In Section V, we presented two link design schemes with *scheme 1* having very little hardware overhead and *scheme 2* having higher performance. The efficiency of the schemes depends on the congestion levels in the NoC and the application's traffic patterns. For heavily congested NoCs, most of the traffic would be sampled through the *delayed flip-flops* in both schemes, resulting in similar performance. For uncongested networks supporting bursty application traffic, *scheme 2* has much higher performance than the *scheme 1* design. These effects are illustrated in Fig. 26(b), where the average packet latencies in a mesh network using *scheme 1* design are presented. The latency values are normalized with respect to the latency incurred by the *scheme 2* design for an uncongested NoC. The traffic pattern is such that each core injects bursty traffic onto the NoC. For such a bursty traffic pattern, *scheme 2* design has minimum overhead for all congestion levels, while the performance of the *scheme 1* design depends on the particular congestion level. We varied the congestion in the network, which is represented in Fig. 26(b), by the percentage of time data sampled by the *delayed flip-flop*. As shown, as the congestion in the network starts to increase, the performance of *scheme 1* design approaches that of the *scheme 2* design. The different link design schemes can be used in different parts of the same NoC if needed, as they have the same interface to the switches/NIs. Thus, particular links that need higher performance can be designed using *scheme 2*.

G. Synthesis Results

Using Synopsys Design Compiler, we synthesized the *T-error* schemes to get area estimates of the proposed schemes. For synthesis, we use a United Microelectronics Corporation 0.13μ technology library, a base NoC operating frequency of 1 GHz, and an operating voltage of 1.2 V. Table I shows the area overhead for the different *T-error* schemes for 32-b flit-size for a 5×5 mesh NoC. The base NoC area is the sum of the areas of switches, links, and NIs without the *T-error* design changes. As shown in Table I, the schemes incur only a modest increase in area (around 4% increase in the base NoC area).

TABLE I
AREA OVERHEAD

Design	Area (mm^2)
Base NoC	4.9
T-error Scheme 1 NoC	4.95
T-error Scheme 2 NoC	5.1

IX. CONCLUSION

The use of conservative methods to design NoCs that target *safe* operation under all conditions leads to suboptimal system performance. In this paper, we have presented aggressive *timing-error-tolerant (T-error)* design methodologies for designing the switches, links, and NIs of NoCs. The NoC in the *T-error* system is designed aggressively to operate at frequencies higher than conservative designs and to recover from the resulting timing errors in an efficient manner. The error recovery mechanism is integrated with a new link-based flow-control mechanism, so that most of the error recovery penalty is hidden under the network operation. Experiments show large performance improvements (up to $1.5\times$) for the communication architecture in the proposed system when compared to traditional conservative designs. In the future, we plan to analyze the power consumption of the *T-error* methods and to extend them for aggressively designing processor architectures.

REFERENCES

[1] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vencentelli, "Addressing the system-on-a-chip interconnect woes through communication-based design," in *Proc. DAC*, Jun. 2001, pp. 667–672.

[2] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J. Soininen, M. Forsell, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. ISVLSI*, Apr. 2002, pp. 117–122.

[3] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. DATE*, Mar. 2000, pp. 250–256.

[4] K. Goossens, J. Dielissen, and A. Radulescu, "The Aethereal network on chip: Concepts, architectures, and implementations," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 21–31, Sep./Oct. 2005.
[5] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *Computers*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[6] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. DAC*, Jun. 2001, pp. 684–689.
[7] R. Ho, K. Mai, and M. Horowitz, "The future of wires," *Proc. IEEE*, vol. 89, no. 4, pp. 490–504, Apr. 2001.
[8] L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.
[9] L. Carloni and A. Sangiovanni-Vincentelli, "Coping with latency in SoC design," *IEEE Micro*, vol. 22, no. 5, pp. 24–35, Sep./Oct. 2002.
[10] D. Ernst, N. S. Kim, S. Pant, S. Das, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. Int. Symp. Microarchitecture*, Dec. 2003, pp. 7–18.
[11] T. Austin, D. Blaauw, T. Mudge, and K. Flautner, "Making typical silicon matter with razor," *Computer*, vol. 37, no. 3, pp. 57–65, Mar. 2004.
[12] A. Uht, "Going beyond worst-case specs with TEAtime," *Computer*, vol. 37, no. 3, pp. 51–56, Mar. 2004.
[13] M. Favalli and C. Metra, "Low-level error recovery mechanism for self-checking sequential circuit," in *Proc. DFT*, Oct. 1997, pp. 234–242.
[14] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition signaling asynchronous pipelines," in *Proc. ICCD*, Sep. 2001, pp. 9–17.
[15] E. Dupont, M. Nicolaidis, and P. Rohr, "Embedded robustness IPs for transient error free ICs," *IEEE Des. Test Comput.*, vol. 19, no. 3, pp. 56–70, May 2002.
[16] W. J. Dally and J. W. Poulton, *Digital Systems Engineering*. Cambridge, U.K.: Cambridge Univ. Press, 1998.
[17] Y. Eo, S. Shin, W. Eisenstadt, and J. Shim, "A decoupling technique for efficient timing analysis of VLSI interconnects with dynamic current switching," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 9, pp. 1321–1337, Sep. 2004.
[18] D. Wang and W. McNall, "A statistical model based ASIC skew selection method," in *Proc. IEEE Workshop Microelectron. and Electron Devices*, 2004, pp. 64–66.
[19] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. De Micheli, "Xpipes lite: A synthesis oriented design library for networks on chips," in *Proc. DATE*, Mar. 2005, pp. 1188–1193.
[20] L. Chen, M. Marek-Sadowska, and F. Brewer, "Coping with buffer delay change due to power and ground noise," in *Proc. DAC*, Jun. 2002, pp. 860–865.
[21] P. J. Restle, K. A. Jenkins, A. Deutsch, and P. W. Cook, "Measurement and modeling of on-chip transmission line effects in a 400 MHz microprocessor," *IEEE J. Solid-State Circuits*, vol. 33, no. 4, pp. 662–665, Apr. 1998.
[22] R. Hegde and N. R. Shanbhag, "Toward achieving energy efficiency in presence of deep submicron noise," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 4, pp. 379–391, Aug. 2000.
[23] D. Bertozzi, L. Benini, and G. De Micheli, "Low power error-resilient encoding for on-chip data buses," in *Proc. DATE*, Mar. 2002, pp. 102–109.
[24] P. Vellanki, N. Banerjee, and K. Chatha, "Quality-of-service and error control techniques for network on chip architectures," in *Proc. GLSVLSI*, Apr. 2004, pp. 45–50.
[25] H. Zimmer and A. Jantsch, "A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip," in *Proc. ISSS/CODES*, Sep. 2003, pp. 188–193.
[26] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networks-on-chips," *IEEE Des. Test Comput.*, vol. 22, no. 5, pp. 434–442, Sep./Oct. 2005.
[27] M. R. Stan and W. P. Burleson, "Bus-invert coding for lowpower I/O," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 1, pp. 49–58, Mar. 1995.
[28] L. Li, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "A cross-talk aware interconnect with variable cycle transmission," in *Proc. DATE*, Feb. 2004, pp. 1012–1017.
[29] K. Patel and I. Markov, "Error-correction and cross-talk avoidance in DSM busses," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 12, no. 10, pp. 1076–1080, Oct. 2004.
[30] S. Srinivas and N. Shanbag, "Coding for system-on-chip networks: A unified framework," in *Proc. DAC*, Jun. 2004, pp. 103–106.
[31] K. Hirose and H. Yasuura, "A bus delay reduction technique considering cross-talk," in *Proc. DATE*, Mar. 2000, pp. 441–445.
[32] P. Sotiriadis, "Interconnect modeling and optimization in deep sub-micron technologies," Ph.D. dissertation, Massachusetts Inst. Technol., Cambridge, MA, 2002.
[33] R. Tamhankar, S. Murali, and G. De Micheli, "Performance driven reliable link design for networks on chips," in *Proc. ASPDAC*, Jan. 2005, pp. 749–754.
[34] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Mateo, CA: Morgan Kaufmann, Dec. 2003.
[35] M. Mizuno, W. J. Dally, and H. Onishi, "Elastic interconnects: Repeater-inserted long wiring capable of compressing and decompressing data," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2001, pp. 346–347.
[36] V. Chandra, H. Schmit, A. Xu, and L. Pileggi, "A power aware system level interconnect design methodology for latency insensitive systems," in *Proc. ICCAD*, Nov. 2004, pp. 275–282.
[37] R. Marculescu, "Networks-on-chip: The quest for on-chip fault-tolerant communication," in *Proc. IEEE ISVLSI*, Feb. 2003, pp. 8–12.
[38] F. Worm, P. Ienne, P. Thiran, and G. De Micheli, "A robust self-calibrating transmission scheme for on-chip networks," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 1, pp. 126–139, Jan. 2005.
[39] M. Pirretti, G. Link, R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Fault tolerant algorithms for network-on-chip interconnect," in *Proc. ISVLSI*, Feb. 2004, pp. 46–51.
[40] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon, "Analyzing on-chip communication in a MPSoC environment," in *Proc. DATE*, Feb. 2004, pp. 752–757.
[41] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "A methodology for mapping multiple use-cases on to networks on chip," in *Proc. DATE*, Mar. 2006, pp. 1–6.
[42] A. Pullini, F. Angiolini, D. Bertozzi, and L. Benini, "Fault tolerance overhead in network-on-chip flow control schemes," in *Proc. SBCCI*, Sep. 2005, pp. 224–229.

Rutuparna Tamhankar (M'06) received the Bachelor of Electronics Engineering degree from the University of Pune, Pune, India, in 1999, the Master of Science degree in electrical engineering from the West Virginia University, Morgantown, in 2001, and the Engineer's degree in electrical engineering from Stanford University, Stanford, CA, in 2005. His thesis at Stanford University was related to performance-driven link design for networks-on-chip.

Currently, he is with the Circuit Design Group, Marvell Semiconductors Inc., Santa Clara, CA.

Srinivasan Murali (S'02) received the Bachelor of Computer Science and Engineering degree (with a gold medal) from the University of Madras, Chennai, India, in 2002. He is currently working toward the Ph.D. degree in electrical engineering at Stanford University, Stanford, CA.

His research interests include reliable and efficient design methods for networks-on-chip and systems-on-chip.

Mr. Murali won a Best Paper Award at the Design, Automation and Test in Europe (DATE) Conference in 2005.

Stergios Stergiou received the B.S. degree from the University of Athens, Athens, Greece, and the M.S. degree in computer science from the University of Patras, Patras, Greece. He is currently working toward the Ph.D. degree in electrical engineering at Stanford University, Stanford, CA.

His research interests comprise all aspects of computer-aided design of digital circuits.

Antonio Pullini received the M.S. degree in electronics engineering from the University of Bologna, Bologna, Italy, in 2005.

He is currently a Research Assistant with the Politecnico di Torino, Turin, Italy. His research interests include low-power digital design and networks-on-chip.

Federico Angiolini received the M.S. degree (*summa cum laude*) in electrical engineering from the University of Bologna, Bologna, Italy, in 2003, where he is currently working toward the Ph.D. degree in the Department of Electronics and Computer Science.

His research is mostly focused on memory hierarchies, multiprocessor-embedded systems, networks-on-chip, and nanotechnologies.

Luca Benini (S'94–M'97–SM'04) received the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1997.

He is a Professor with the University of Bologna, Bologna, Italy. He also holds a Visiting Faculty position with the Ecole Polytecnique Federale de Lausanne. His research interests are in the design of systems for ambient intelligence, from multiprocessor systems-on-chip/networks-on-chip to energy-efficient smart sensors, and sensor networks. He has published more than 250 papers in peer-reviewed international journals and conference proceedings, three books, several book chapters, and two patents.

Dr. Benini has been Program Chair and Vice-Chair of the Design, Automation and Test in Europe Conference. He was a member of the 2003 MEDEA+ EDA roadmap committee 2003. He is a member of the IST Embedded System Technology Platform Initiative (ARTEMIS): working group on design methodologies, a member of the Strategic Management Board of the ARTIST2 Network of Excellence on Embedded System, and a member of the Advisory Group on Computing Systems of the IST Embedded Systems Unit. He has been a member of the technical program committees and organizing committees of several technical conferences, including the Design Automation Conference, International Symposium on Low Power Design, and the Symposium on Hardware 100-Software Codesign. He is Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS and of the *ACM Journal on Emerging Technologies in Computing Systems*.

Giovanni De Micheli (S'82–M'83–SM'89–F'94) received the B.Sc. degree in nuclear engineering from Politecnico di Milano, Milan, Italy, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1980 and 1983, respectively.

He is currently a Professor and Director of the Integrated Systems Center, École Polytechnique Fédérale, Lausanne, Switzerland, and President of the Scientific Committee of Centre Suisse d'Electronique et de Microtechnique (CSEM), Neuchatel, Switzerland. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware/software codesign, and low-power design. He is the author of *Synthesis and Optimization of Digital Circuits* (McGraw-Hill, 1994), and coauthor and/or coeditor of six other books and of over 300 technical articles. He has been a member of the technical advisory board of several companies including Magma Design Automation, Coware, Aplus Design Technologies, Ambit Design Systems, and STMicroelectronics.

Dr. De Micheli was the recipient of the 2003 IEEE Emanuel Piore Award for contributions to computer-aided synthesis of digital systems. He is a Fellow of ACM. He received the Golden Jubilee Medal for outstanding contributions to the IEEE Circuits and Systems (CAS) Society in 2000, the 1987 D. Pederson Award for the best paper in the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, two Best Paper Awards at the Design Automation Conference, in 1983 and 1993, and a Best Paper Award at the DATE Conference in 2005. He was President of the IEEE CAS Society in 2003, and is currently President Elect of the IEEE Council on Electronic Design Automation (EDA) and chairing of the IEEE Product Package Committee. He was Program Chair of the pHealth and VLSI SOC conferences in 2006. He was Editor-in-Chief of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF 1084 INTEGRATED CIRCUITS AND SYSTEMS in 1987–2001, and he was the Program Chair and General Chair of the Design Automation Conference in 1996–1997 and 2000, respectively.