Chapter 8
Timing-Error Tolerant NoC Design

With technology scaling, the device characteristics fluctuate to a large extent due to process variations and can cause significant variations in wire delay [122]. Wire delay is also affected by other forms of interference such as supply bounce, transmission line effects, etc. [123, 124]. As such delay variations can affect multiple bits simultaneously, special mechanisms are needed to handle timing errors. In this chapter, we present *T-error*, a timing-error tolerant mechanism to make the interconnect resilient against timing errors arising due to such delay variations on wires.[1]

Current NoC 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* [113, 114], 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.

The proposed *T-error* methods are used 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 large performance benefits can be obtained. Finally, the switches, 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 [55]. 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

[1] We would to acknowledge the contributions of Rutuparna Tamhankar, Stergios Stergiou, Antonio Pullini, Dr. Federico Angiolini, Prof. Luca Benini, and Prof. Giovanni De Micheli.

S. Murali, *Designing Reliable and Efficient Networks on Chips,*
Lecture Notes in Electrical Engineering 34,

requires fast execution, the NoC can be over-clocked 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 [113], where the system's error rate is constantly monitored to tune the voltage or frequency, we envision that the *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 *over-clocked mode*. In *normal mode*, the NoC operates at frequencies less than or equal to the frequency of a conservative design. Under *over-clocked* mode, the frequency of operation can be higher than that of the traditional design. The NoC under the *over-clocked* mode incurs some penalty for error resiliency, even when there are no errors in the system (this is explained in detail in Section 8.4.2). 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 *over-clocked* modes of operation at the application level.

The *T-error* scheme for a NoC link is presented in [137]. In this work, 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. 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.

8.1 The Double Sampling Technique

In most NoC realizations, when errors are detected, corrupted packets are retransmitted. Unfortunately, retransmissions incur significant performance penal-

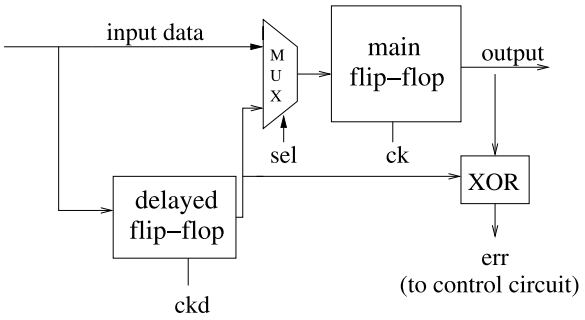


Fig. 8.1 Double data sampling technique

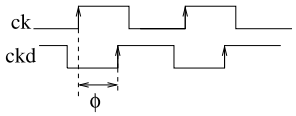


Fig. 8.2 Phase shift between clocks

ties [130]. 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 [130].

To recover from timing errors in a digital system, *double data sampling techniques* have been proposed and used by several researchers [113–120]. 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 Figure 8.1. Both the *main* and the *delayed 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 Figure 8.2.

Thus, data sampled by the *delayed flip-flop* has 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 that 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, data from the *delayed flip-flop* is assumed to be correct and is 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 a 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 *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 Figure 8.2) 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 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 *over-clocked* mode the errors introduced due to over-clocking 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 FIFOs 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 work, we present methods that address only the timing delay variations on the NoC that are introduced due to over-clocking. Coping with other kind of errors (such as soft errors, capacitive coupling based cross-talk, data upsets, etc.) is assumed to be done by means of existing techniques (such as [126–135]). 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 over-clocked frequency of operation.

8.2 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 [94]. 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 [94], 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 [94]. To support

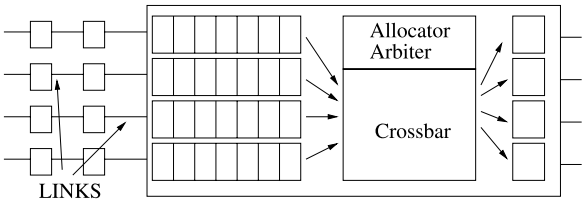


Fig. 8.3 Input queued switch

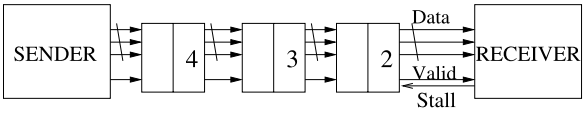


Fig. 8.4 Modified link design with 3 stages

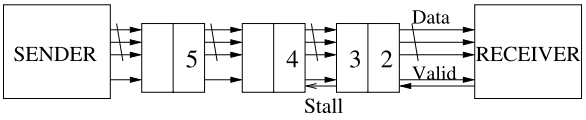


Fig. 8.5 Entry 3 buffered in *secondary flip-flop*

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 (Figure 8.3).

In [70], 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 2-entry FIFO and a link-level flow control is used to ensure full throughput operation. We utilize such links for the NoC architecture. In the NoC architecture, the switch input buffers are also replaced by a 2-entry FIFO. Figure 8.4 shows a 3-stage link pipeline using 2-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 a network interface. 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 was 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 2-entry FIFO is used, as shown in Figure 8.4. When a *stall* signal is received by the 2-entry FIFO (shown in Figure 8.5), the data on output of the *main flip-flop* is stalled and the new data is received by the *secondary flip-flop*. The *stall* signal is propagated to the previous stage, as shown in Figure 8.6. The schematic of the 2-entry FIFO is shown in Figure 8.7.

This flow control mechanism ensures full throughput operation with performance similar to that of *input-queued* switches with credit-based or on/off flow control. As previously shown, in traditional *input-queued* schemes (Figure 8.3), the total number of buffers needed for maximum throughput is $3N$, as compared to only $2N$

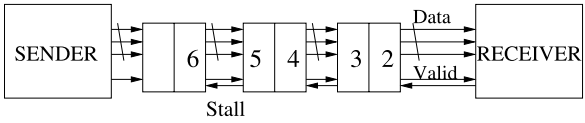


Fig. 8.6 The *stall* signal propagated to previous stage

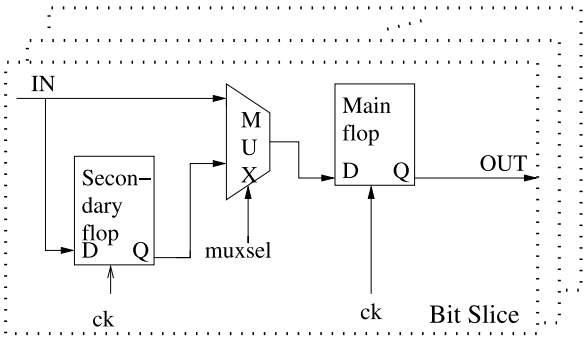


Fig. 8.7 A 2-entry FIFO. The control circuit is common for all the bit lines

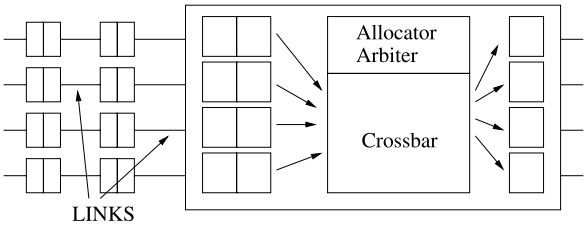


Fig. 8.8 Modified link and switch design

buffers ($2 \times (N - 1)$ along the link and 2 at the switch input) in this scheme (Figure 8.8). The traditional *input-queued* design has one flip-flop at each link pipeline stage. In the *stall/valid* protocol, it takes one clock cycle for the *stall* signal to reach the preceding pipeline stage. During this time, the data which is 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 data is 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 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.

8.3 *T-error* Link Designs

In this section, we present two link designs to support timing error tolerant operation needed for over-clocking 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 2-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.

8.3.1 Scheme 1: Low overhead T-error Links

In the *T-error* scheme, the 2-entry FIFOs along the links are modified to support timing error tolerant operation. The modified FIFO structure is shown in Figure 8.9. 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 is sampled twice, once by the *main flip-flop* (at time instant t_0 in Figure 8.11) and then by the *delayed flip-flop* (at time instant t_1). There are two modes of operation at 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 Figure 8.9, 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.

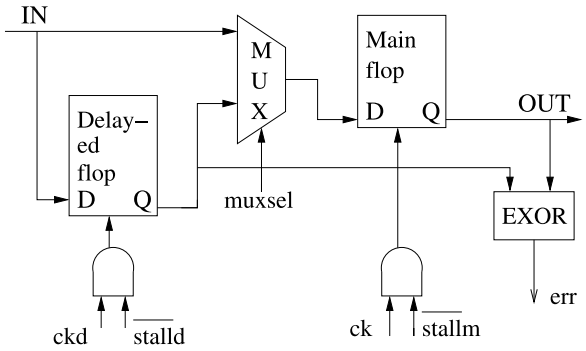


Fig. 8.9 Low overhead *T-error* buffer

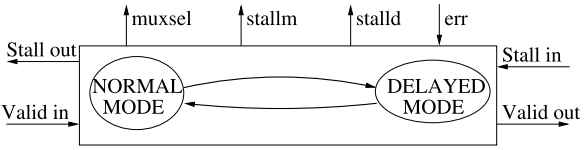


Fig. 8.10 Control circuit for scheme 1

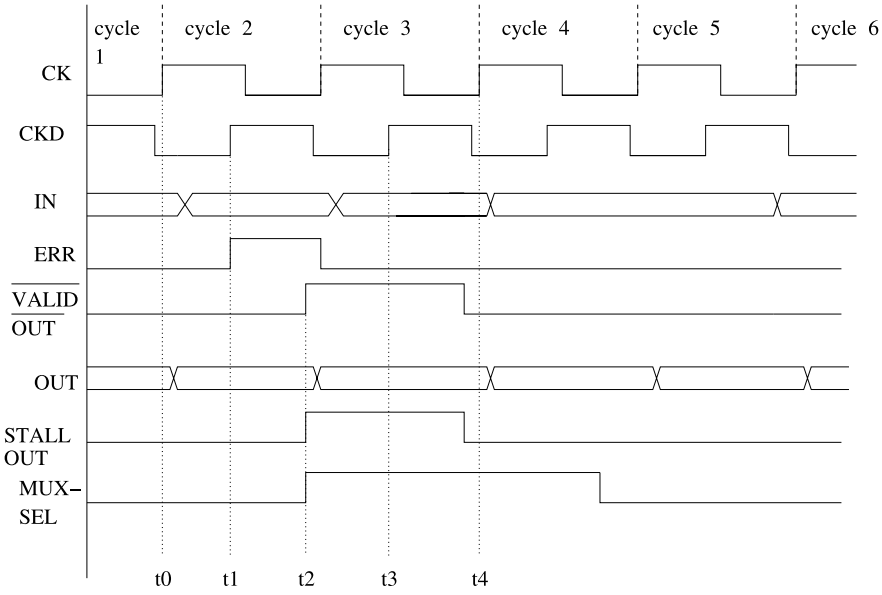


Fig. 8.11 Waveforms for scheme 1

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 Figure 8.10.

If there is an error in the data sampled by the *main flip-flop*, the data that was transmitted at clock edge *ck* is incorrect. The correct data from the *delayed flip-flop* is 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 one cycle. Also, a $\overline{valid}$ signal is sent to the following stage, informing that the data sent in the previous cycle was 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 *delayed mode*, all timing errors are automatically avoided, as the incoming data is 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 Figures 8.12 and 8.13. In the network with no congestion, at each pipeline stage, data is always directly

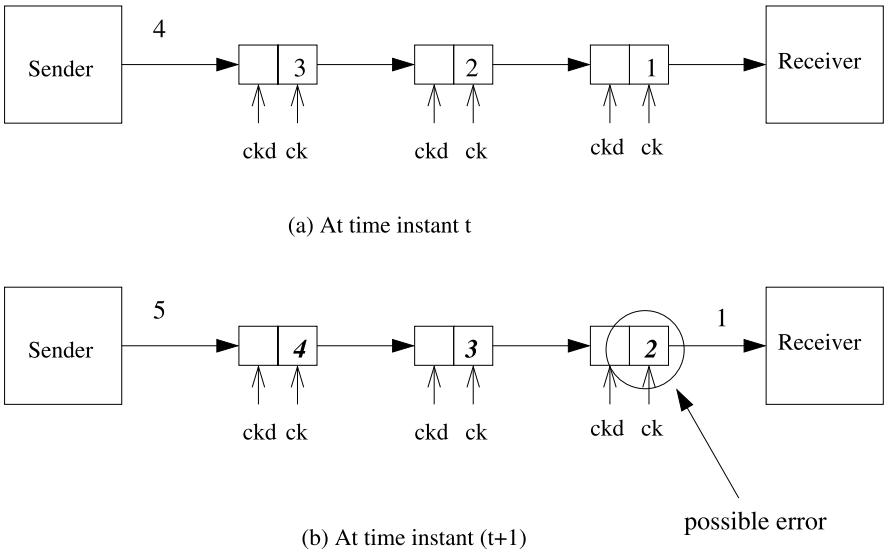


Fig. 8.12 Network operation without congestion. The data in the FIFOs at time instances t and $(t + 1)$ are presented in (**a**) and (**b**)

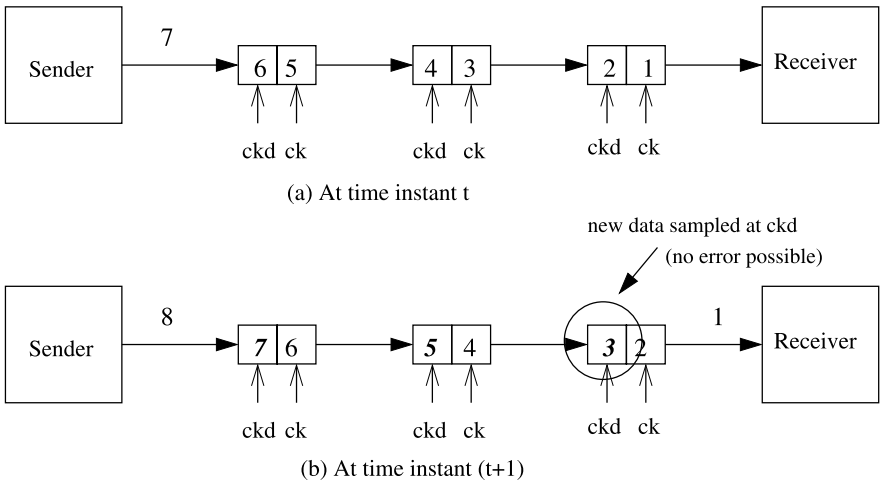


Fig. 8.13 Network operation under congestion. The data in the FIFOs at time instances t and $(t + 1)$ are presented in (**a**) and (**b**)

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 data is 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 1 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 is presented in Section 8.6.6.

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 bits) showed that clock *ckd* can be delayed by 53.3% of the clock period with respect to *ck*. In this work, 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 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 [137]. The control lines (*stall, valid*) that need to have error-free operation can be made robust using a variety of methods (such as using wider metal lines, shielding). We refer the interested reader to [137] for transistor-level implementation details, timing analysis, and SPICE simulation results of the *T-error* scheme.

8.3.2 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 3-entry FIFO, instead of the 2-entry FIFO previously described, is used in this scheme (refer to Figures 8.14 and 8.15). The third flip-flop, called *auxiliary 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 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 data flow.

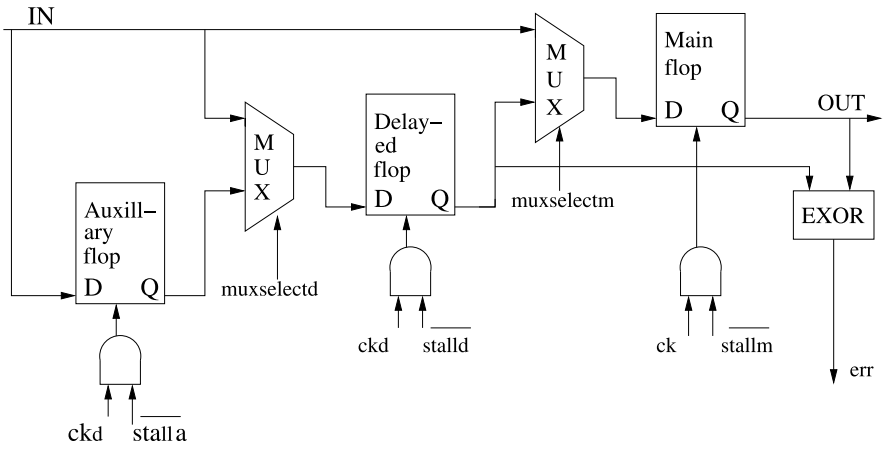


Fig. 8.14 Schematic for scheme 2

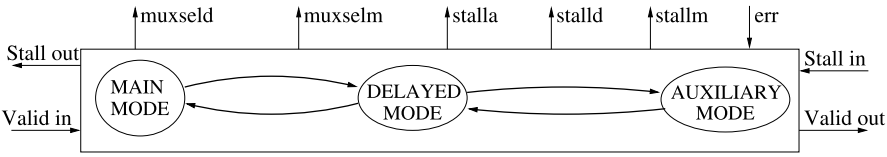
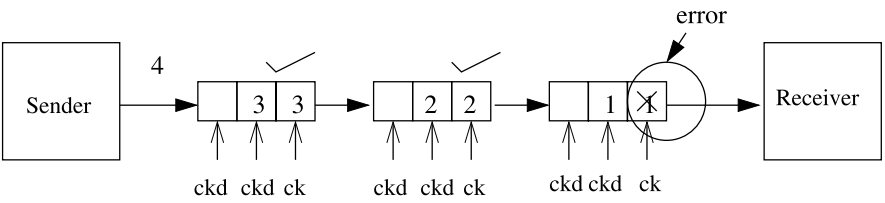
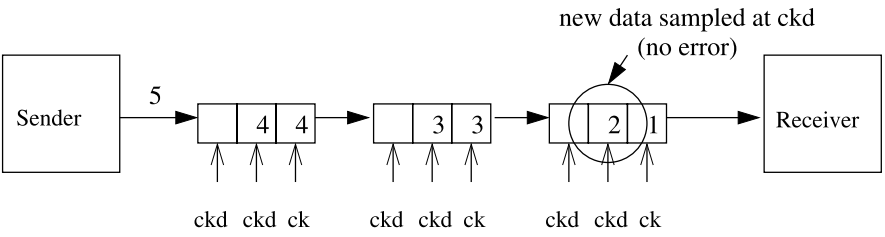


Fig. 8.15 Control circuit for scheme 2



(a) At time instant t, clock edge ckd



(b) At time instant (t+1), clock edge ckd

Fig. 8.16 Example of 3-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

Only at the end of the whole data stream, the stage empties and switches back to *main mode*. An example is presented in Figure 8.16. Note that even in 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.

8.4 Aggressive Switch/NI Design

In this section, we describe the changes needed in the basic architecture to support the *over-clocked* mode of operation. The ×pipes NI is 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 over-clocking. 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 over-clocking. 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. A NoC switch, as shown in Figure 8.8, consists of input buffers, allocator/arbiter, crossbar and output buffers. In the 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.

8.4.1 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 ×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 ×pipes switch with a single output buffer is shown in Figure 8.8. The ×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 over-clocked: output buffers and flip-flops that are used to maintain the allocator/arbiter states. From synthesis of the ×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 over-clocking, we target a 1.5× increase in frequency (i.e., 1.5 GHz operating frequency) of the

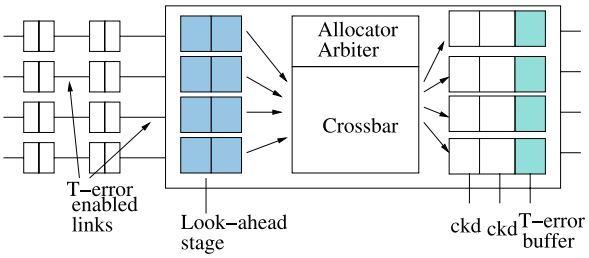


Fig. 8.17 Over-clocked switch design with output and input buffer changes

switches. Therefore, we found that the state flip-flops are *safe* even under over-clocking, 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 over-clocked, they should be *T-error* enabled as well. Otherwise, the amount of over-clocking 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 over-clock 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*. Figure 8.17 shows the changes in the output buffer of the switch. Note that errors can occur only when the data is sampled through the head of the FIFO and when the NoC operates in the *over-clocked* mode.

8.4.2 Input Buffer Changes

When timing errors occur at a link pipeline stage, wrong data can reach the switch input before the correct data is 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 require complex hardware and control circuits [94]. Another way to detect wrong data at the switch input is to use some error detecting code (such as cyclic redundancy check) for each flit of the packet. However, in the *over-clocked* 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 is always fed to the internal switch logic (see again Figure 8.17). The look-ahead stage stores an incoming flit for one clock cycle, i.e., until the *valid* line indicates whether the received data was correct or not. In case of correct reception, data is 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.

8.5 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 (one or more) processing cores. A value of $BOOST = 1$ indicates that the NoC is in *over-clocked* 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 *over-clocked* modes occur 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 Figure 8.18. The transition from the *normal mode* to *over-clocked* mode is smooth in the design, as the look-ahead is started when the *BOOST* signal is spread. However, transition from the *over-clocked* 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 is 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 is fed to the switch inputs. As the transitions between *normal* and *over-clocked* modes occur at the application level (which may occur every tens of thousands of cycles), the performance overhead incurred due to this dynamic configuration is negligible.

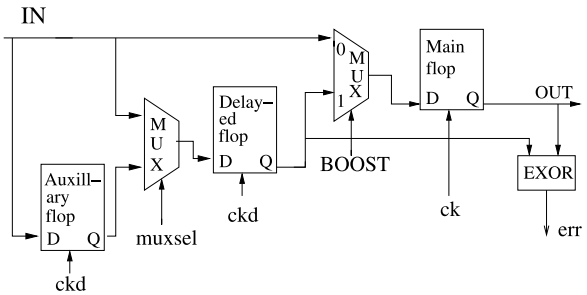


Fig. 8.18 The *look ahead stage* at the switch input

8.6 Experimental Results

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

8.6.1 Simulation Platform

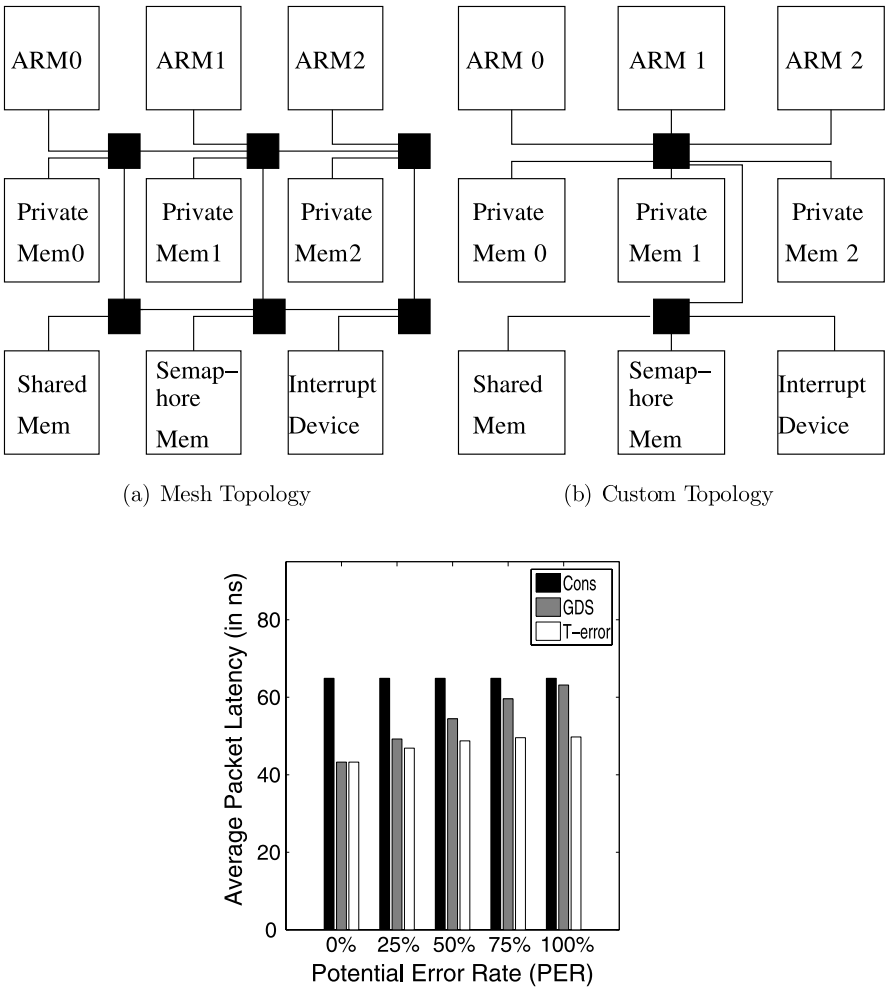
The simulation platform consists of cycle-accurate SystemC models of the *T-error* designs for the switches, links, and NIs, incorporated on the ×pipes architecture. Functional SystemC simulations were carried out on a variety of application benchmarks.

8.6.2 Experiments on a Multi-Media Benchmark

We plugged 3 ARM7 processors, 3 private memories (one for each processor), and 3 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 (Figures 8.19(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 2 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 1 pipeline stage, while those of the custom topology have 2 pipeline stages.

We perform experiments on 3 schemes: a traditional *CONServative* (*CONS*) design approach, a *General Double-Sampling* (*GDS*) scheme that is not integrated with the network flow control (such as presented in the earlier works [113] and the *T-error* scheme with 3-stage FIFO presented in this work. 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 *over-clocked 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



(a) Mesh Topology

(b) Custom Topology

(c) Mesh Topology Results

Fig. 8.19 Mesh and custom topology mappings and comparison of traditional schemes with *T-error*

the design. This is because in most scenarios, data is sampled first by the *delayed flip-flop* and only afterward sent out by the *main flip-flop*, avoiding all potential errors. For an *over-clocked* system, the *PER* value depends on how much the system is *over-clocked*, the actual operating conditions of the system (such as effect of process variations on the FIFO, operating temperature, other noise effects), actual data patterns on the link, etc. As an example, if bus encoding techniques are not used to reduce the effects of capacitive cross-talk, 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 *over-clocked* mode, if the adversarial switching patterns do

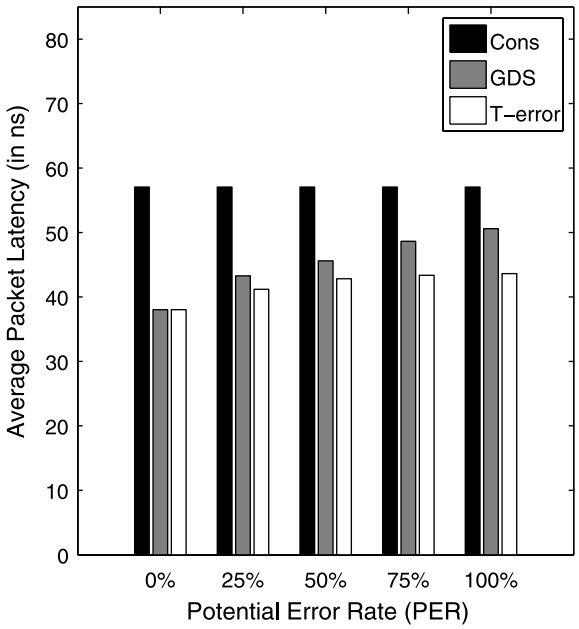


Fig. 8.20 Custom topology results

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 Figures 8.19(c) and 8.20. As we *over-clock* 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 the traditional conservative design (CONS), 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 double sampling scheme (GDS), 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 3-stage FIFOs at the links) also results in 30% reduction in the number of queuing buffers used. In fact, the 3-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 2 buffers at the switch input, while the input queued switches with the general double sampling technique needs $2N+1$ buffers at the input of the switch and $2 \times (N-1)$ buffers on the links (refer to Section 8.2, where results for 2-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 seen from Figure 8.21, even on significantly long links, the

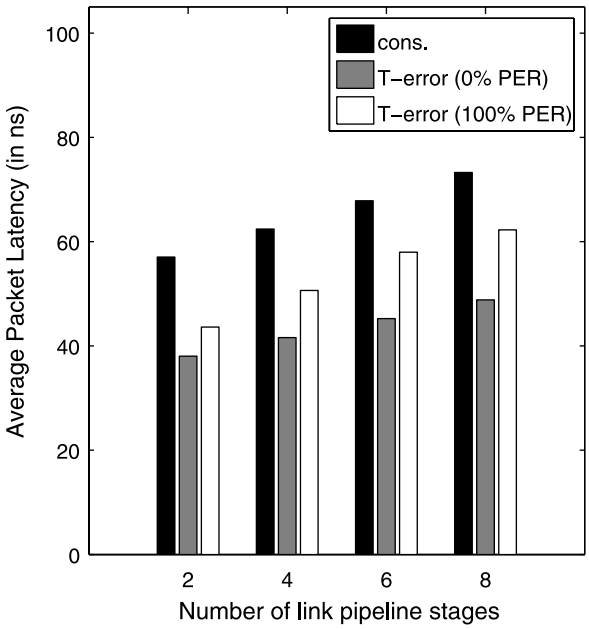


Fig. 8.21 Effect of pipeline depth

T-error scheme gives a large improvement in performance when compared to the conservative design approach.

8.6.3 Effect of Application-Level Power Management

We conducted experiments on the multimedia benchmarks to show the usefulness of the application-level DPM policies. We model 4 different application scenarios in the platform: *Standard Definition* video decoding and display (*SD*), *High Definition* video decoding and display (*HD*), *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 Figure 8.22. The use of application level DPM policies results in an average of 57% reduction in power consumption of the NoC.

8.6.4 Experiments on Other Benchmarks

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

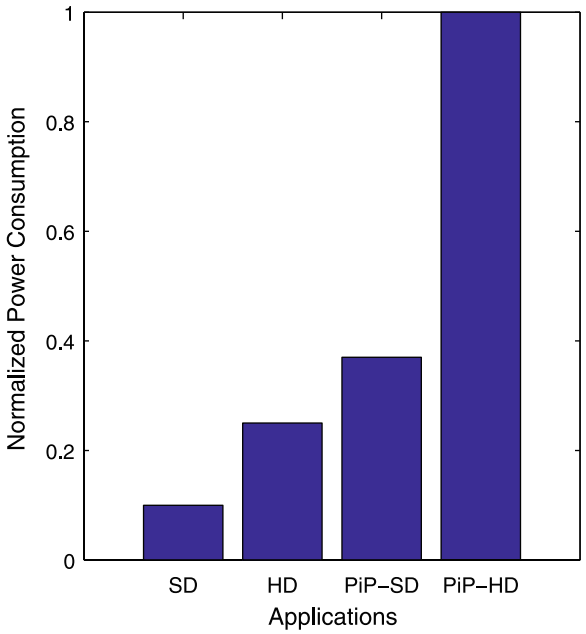
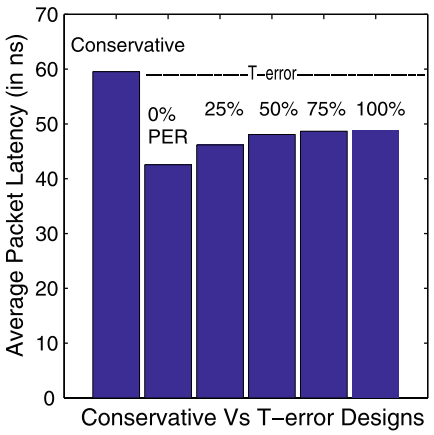


Fig. 8.22 Effect of DPM policies

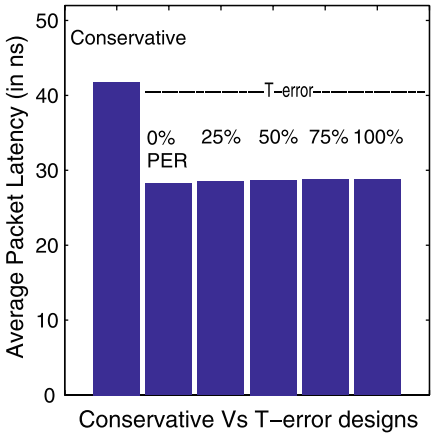
- Matrix multiplication benchmark suite without shared memory (*MAT1*)
- Matrix multiplication benchmark suite with shared memory (*MAT2*)
- Fast Fourier transform benchmark suite using fixed point arithmetic (*FFT*)
- Quick sort benchmark suite (*Qsort*)

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 2 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 8.6.6, we use the 3-entry *T-error* FIFO design. In Section 8.6.6, we compare the performance of the two *T-error* link designs.

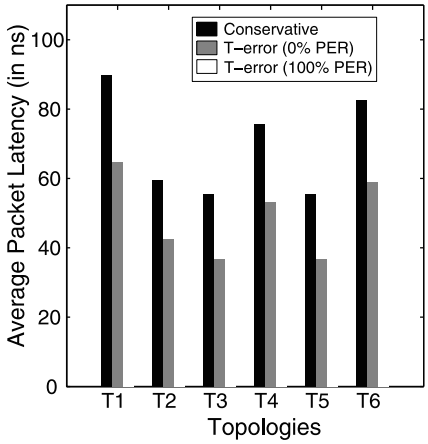
In Figures 8.23(a) and (b), the average packet latency (in ns) observed for the conservative and *T-error* design for the *MAT2* benchmark for read (Figure 8.23(a)) and write transactions (Figure 8.23(b)) is presented. The read transactions require two way data transfer on the network: a request is sent 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 seen from 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*).



(a) MAT2: read transactions



(b) MAT2: write transactions

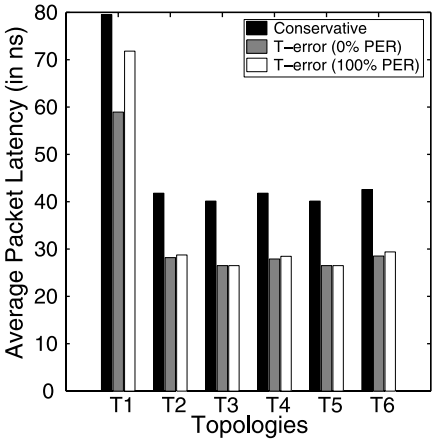


(c) Topology Effects: read transactions

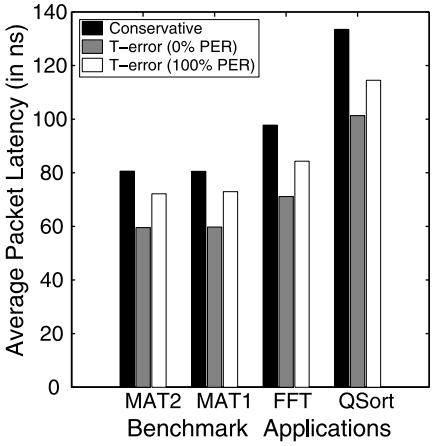
Fig. 8.23 Performance comparison of conservative and *T-error* designs for different *PER* values for read and write transactions

For the write transactions, the average reduction in latency for the *T-error* designs vary 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.

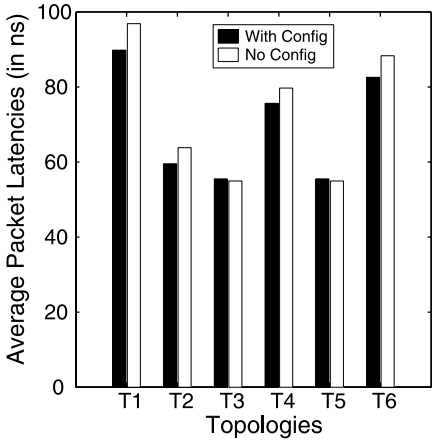
The performance of the *T-error* system for various topologies for the *MAT2* benchmark for read and write transactions are presented in Figures 8.23(a) and 8.24(a). The designs compared vary from small 7-core NoCs to 51-core NoCs with different application partitioning. The topologies vary from regular (like *mesh*)



(a) Topology Effects: write transactions



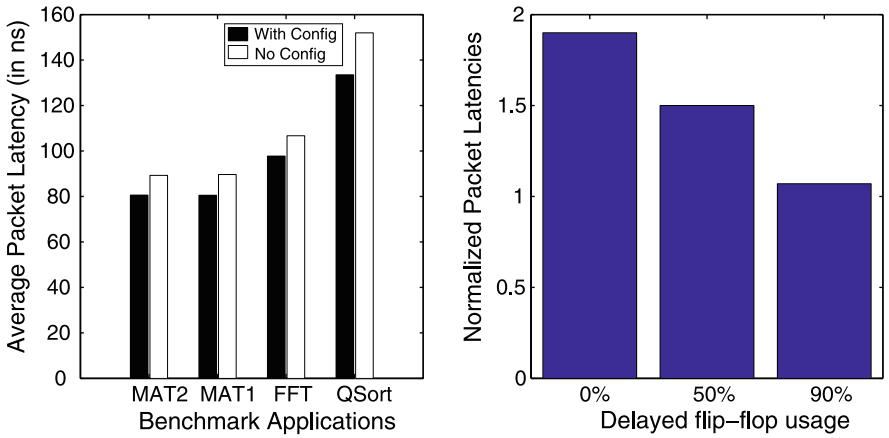
(b) Benchmark Effects: read and write transactions



(c) Topology Effects: NoC Configuration

Fig. 8.24 (**a**) and (**b**) Performance comparison for various topologies, benchmarks, and (**c**) Effect of dynamic NoC configuration

to custom, manually developed ones. As seen from 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 Figure 8.24(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 latency for the benchmarks for the *T-error* designs varies from 25.7% (for 0% *PER*) to 12.7% (for 100% *PER*).



(a) Benchmark Effects: NoC Configuration

(b) Choice of Link Design Schemes

Fig. 8.25 (**a**) NoC configuration and (**b**) Choice of link design schemes

8.6.5 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 $\leq$1 GHz. In Figures 8.24(c) and 8.25(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 packet latency (up to 13.8%) for the applications. This reduction is attributed to two reasons: one is the reduction in pipeline depth of the 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 packets spend less time in the network.

8.6.6 Choice of Link Design Schemes

In Section 8.3, 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 Figure 8.25(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

Table 8.1 Area overhead

Design	Area (mm^2)
Base NoC	4.90
T-error scheme 1 NoC	4.95
T-error scheme 2 NoC	5.10

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 Figure 8.25(b) by the percentage of time data is sampled by the *delayed flip-flop*. As seen, 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*.

8.6.7 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 UMC 0.13 µm technology library, a base NoC operating frequency of 1 GHz and an operating voltage of 1.2 V. Table 8.1 shows the area overhead for the different *T-error* schemes for 32-bit 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 seen from the table, the schemes incur only a modest increase in area (around 4% increase in the base NoC area).

8.7 Summary

The use of conservative methods to design NoCs, that target *safe* operation under all conditions leads to suboptimal system performance. In this chapter, 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. The methods are also applicable to remove timing errors in conservative designs.