TERROR: RELIABLE AND EFFICIENT LINK DESIGN FOR NETWORK ON CHIPS

A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF ENGINEER

Rutuparna Tamhankar
February 2005

Approved for the department.

Giovanni De Micheli (Advisor)

Approved for the University Committee on Graduate Studies.

Abstract

Due to shrinking feature sizes and increasing transistor densities, the number of processor/memory cores on a chip and their speed of operation is increasing. In future *Systems on Chips (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. To effectively design future SoCs, *Networks on Chips (NoCs)*, a communication centric design paradigm that considers the delay and reliability issues of wires has been proposed. Wires are becoming increasingly susceptible to delay variation caused due to cross-talk, coupling capacitance, PVT (Process, Voltage, Temperature) variations, ground bounce, inductive interference etc. 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 is required where the communicating link is resilient to timing errors due to delay variations in wire.

This work presents *Terror* - timing-error tolerant link design methodology for aggressively designing the NoC links. Three different *robust* link design schemes are explored that provide various levels of reliability and latency by trading-off hardware complexity and link area. These schemes incorporate distributed buffering where the links are used not only as a *communicating* medium but also as a *storage* medium. Depending on the application needs, NoC characteristics and communication requirements, the three schemes can be selectively used and interchanged in a *plug and play* fashion. Experimental data and simulations on several benchmark applications show large performance improvement (up to 1.5×) for the proposed system when compared to traditional design methodology.

Acknowledgments

I wish to express my sincere thank you to Prof. Giovanni De Micheli for giving me a wonderful research opportunity to work under his guidance. Through various meetings and discussions, he provided the right direction, motivation and useful technical insights that helped me to improve upon the basic ideas. I would also like to thank Mr. Srinivasan Murali whose suggestions, in depth views, detailed comments and valuable contribution substantially helped to improve the research. I would like to express my sincere gratitude to Stergiou Stergios, Kresimir Mihic and other members of the group for their constructive suggestions and assistance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Traditionally, on-chip global communication has been addressed by shared-bus structures and ad-hoc direct interconnections. Such bus architectures are not scalable on large System on Chip *SoC* designs [1] [28] [2]. Hence a new SoC paradigm, *Network on Chips*, has evolved which addresses the distinctive challenges of providing an efficient, functionally correct, reliable operation of interacting system-on-chip components [3]. Faster clock cycles and shrinking feature sizes in the DSM technology have resulted in using a latency insensitive design approach for designing the communication links [8]. With continued scaling of transistors the wire delay as a fraction of the total delay is increasing [4]. The signal propagation delay of an uninterrupted wire grows quadratically with its length hence after some length the wire needs to be partitioned into smaller segments with buffers in between [27] such that delay of each segment is equal to the clock cycle. For example, in a 50nm technology a highly optimized link will need six to ten clock cycles to cross the chip [4]. The growing need of having high performance at lower cost has lead to optimizing the utilization of wire. As clock frequencies increase, scaled wire become relatively slower and on-chip communication delay becomes the limiting factor of future chips [27]. Ensuring reliability of data transfer becomes all the more important in such situation. Error-correcting link design techniques have been proposed which use schemes such as encoding and decoding the data [13] to detect errors, using asynchronous handshake (acknowledge, request) protocol [14] and using extra elements such as latches to capture double-capture data [15] [12].

In DSM technology, wires are becoming thicker and taller but their widths are not increasing proportionally. Due to this the coupling capacitance to adjacent wires

is increasing leading to large propagation delays due to capacitive crosstalk and increased susceptibility to errors due to DSM noise [21]. The worst case delay of a line is $(1 + 4\lambda)\tau$ where τ is the delay of line without any capacitive coupling and λ is the ratio of the coupling capacitance to bulk capacitance [22]. For $\lambda = 2$, the delay increases by nine fold. Delay penalty due to coupling capacitance is maximum when adjacent buses transition in opposite directions. In a triple RC-coupled line the signal propagation delay of the center line may be 50% off from that of a single isolated line with similar structure [23]. Many bus encoding techniques such as [24] , [25], [21] have been proposed that decrease crosstalk between wires and avoid adversarial switching patterns on the data bus. Most of the schemes require additional wires, encoders and decoders. The use of routing algorithms that decrease the probability of faults have been explored in [36], [39]. The use of coding techniques for detecting and correcting on-chip communication errors have been analyzed in [37], [41], [42]. In most of these coding schemes, once an error is detected, retransmission of data is required. Coding schemes that are used to correct errors at the receiver have higher hardware complexity due to the use of complex encoders and decoders [37]. Moreover, correcting multiple errors by such coding schemes is practically infeasible. 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. The method uses encoding and decoding techniques at sender and receiver ends to detect errors. Li et al., [40] monitor the data bus to detect adverse switching patterns (that increase the wire delay) and change the clock frequency dynamically to avoid timing errors on the bus. This requires a bus monitoring system which may not scale with bus width. Also clock control may not be easily feasible in a heterogenous NoC system. Wire delay can be affected by other forms of interference such as supply bounce [35], electromagnetic interference and alpha particle radiation, which eventually lead to transient errors in data transmission. Device characteristics fluctuate to a large extent and also cause variations in delay. For a typical 8 inch wafer a 3 sigma speed shift across the chip [26] is observed. An approximate formula as given by [26] for frequency shift due to process variation in $0.13u$ process is shown below.

$$\Delta F(L, Vt, Tox) \approx -300\Delta L/L - 800\Delta Vt \tag{1.1}$$

It is observed that $20mV$ shift in Vt causes a change of $16Mhz$. For $5\%\triangle L/L$ of $\triangle L/L$ shift, a frequency deviation of $65Mhz$ is obtained.

Thus the predicted delays during design time may be quite different from the actual delays on silicon. Worst case or conservative design approach will give lesser performance benefits as technology further advances since these undesired effects may consume substantial fraction of the total clock cycle and energy of the chip. Reliable link design approaches are needed where the link is tolerant to the variability in the delay due to unpredictability in the wire characteristics. This work introduces a Timing-Error tolerant (*Terror*) design approach towards NoC link design which ensures reliability, correctness and has minimum impact on system performance. This system corrects transient timing delay errors caused due to crosstalk or adversarial data patterns on adjacent bits of a bus. Errors caused due to manufacturing defects (such as broken link) or permanent failures (such as oxide breakdown) are not addressed by the proposed system.

There have been several approaches in the design space to detect and correct timing errors. Unlike *Terror* scheme, the error penalty of these schemes increases with the error rate. Teatime [11], tracks the logic delay variation and dynamically adjusts the clock frequency to accommodate the changes in logic delay. It tries to avoid timing errors but requires complex analog frequency controller (to change the clock frequency) and entails an overhead of the delay tracking logic. Also, the correction efficiency depends upon the time taken between the detection of actual delay variation and the corresponding change in clock frequency. Razor [12] based system has the same basic principle as *Terror* and is used to control power (supply voltage) by monitoring the error rate. It detects and corrects error but has one cycle penalty per error occurrence. Also, it gates the global clock to delay the whole pipeline. Clock control may not be always possible, especially in heterogeneous systems. Favalli et al. [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, clock is delayed for one cycle, till the correct value of data settles. This results in one cycle penalty for a single timing error. Mousetrap [14] is a high speed asynchronous pipeline which ensures correct data availability to consecutive stages, but has a substantial overhead of communication signals (acknowledge and request signals). Iroc [15] suggests to include a latch with delayed clock to detect transient faults due to soft errors. This technique is similar but gives error penalty per occurrence of soft error and involves clock control circuitry. There are many coding techniques for correcting errors such as the Hamming code, but they require extra wires, decoding and encoding circuitry at sender and receiver

ends and do not scale well with bus widths [37]. Moreover correcting multiple errors in the data is difficult in such coding schemes. All of these techniques introduce large latency overheads depending on the error rate and have substantial overhead for large bus widths. *Terror*-enabled systems scale well with bus width and give a bounded penalty that is only dependent on the number of buffer stages between the sender and the receiver and not dependent on the data error rate.

Terror-enabled systems address only transient timing errors. Static errors such as stuck-at faults or cosmic radiation induced faults are assumed to be detected and corrected by other means such as parity encoding, cyclic redundancy checks etc. *Terror*-based systems are suitable for designs having high error rate. Such high rates are possible in a aggressive design approach, where the clock frequency in higher than conservative design or the physical inter-buffer spacing on a link is increased beyond conservative limits. High error rates can be also present in scenarios where the system voltage is reduced for power savings. *Terror*-enabled systems may not be the suitable for low error rate conditions or low congestion scenarios where simpler schemes can be more useful. In this work, we assume that sufficient routing resources are available to route the extra control signals used in the system.

This thesis is organized as follows. Chapter 2 introduces *Terror* design technique, communication protocol and analysis of its latency benefits. Chapter 3 presents a NoC design flow *Netchip* which is used to design application-specific synthesizable NoC architectures. Chapter 4 describes the implementation and benefits of a *Terror*-enabled NoC that explores three different *robust link design methodologies* based on *Terror* design principle. Chapter 5 presents simulation and experimental results for proposed error-resilient NoC link designs on *Netchip* architecture, using different benchmark suites. Chapter 6 summarizes the conclusion.

Chapter 2

Terror: Timing ERROR Tolerant Link Design

2.1 Timing errors

Timing errors occur when the correct data does not arrive at the capturing (rising) edge of the clock (assuming positive edge triggered flops). This happens when there is a relative deviation in the arrival times of the clock and the correct data. Figure 2.1 shows how a relative shift between clock and data arrival times causes a timing error.

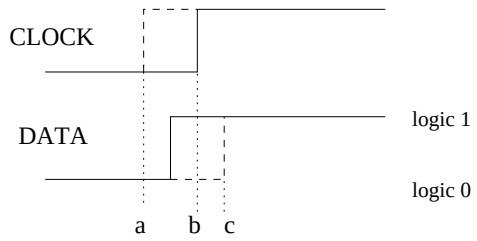

Figure 2.1: Variation in clock and data arrival times

In figure 2.1 the correct value (*logic* 1) of data comes at time instant b and is captured by the rising edge of the clock. If the data arrives late at time instant c or if the clock arrives early at a then a wrong data is captured and we get a timing error. Such timing errors can occur due to a number of reasons such as clock skew, clock jitter, variation in logic delay computation, crosstalk induced delay variation [23] [22], PVT (Process, Temperature, Voltage) variations [26], delay changes due to ground bounce [35] etc. Such effects result in temporary variations in data value, such that

the correct value of data settles down after the rising edge of clock. Figure 2.2 shows various scenarios in which an error can occur. In figure 2.2 correct data is available at time instant t_1 and not t_0. A timing error can be detected by comparing the data captured at t_0 and t_1.

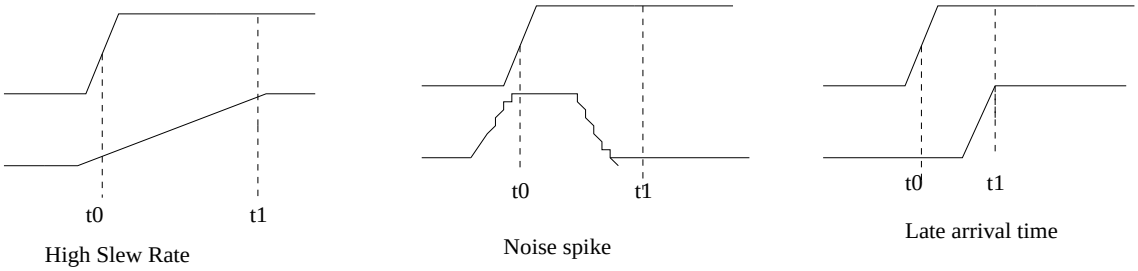

Figure 2.2: Causes of timing error

Faster clock cycles and shrinking features sizes have resulted in smaller logic delays but wire delay has remained relatively constant. A long wire crossing the chip cannot be traversed in one clock cycle. Hence the wire is divided into smaller segments separated by buffers (flops) [45] [44]. The wire delay can now be expressed in integral multiples of clock cycles. Figure 2.3 shows a typical buffered link.

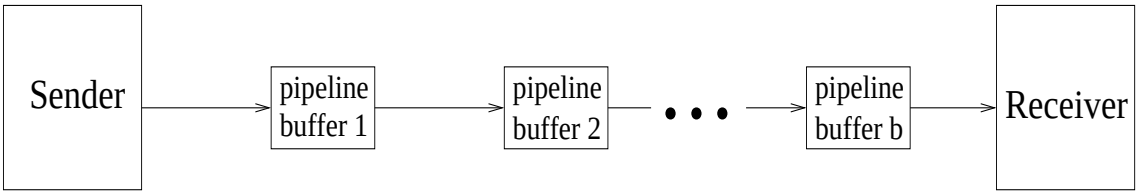

Figure 2.3: Typical Buffered Link

Each buffer is a single flop which captures data at the rising edge of clock. In a Terror enabled link, this buffer is modified and an error correcting circuit is also added, as shown in figure 2.4.

2.2 Design Principle

In a Terror system, the buffer (henceforth called *Terror flop*) has two flops - *main flop* and *delayed flop*. As shown in figure 2.5, the *main flop* is clocked by *ck* and the *delayed flop* is clocked by *ckd*, which is delayed with respect to clock *ck*. The incoming data is sampled twice, once by the *main flop* (at rising edge of clock *ck*) and then by the *delayed flop* (at rising edge of clock *ckd*). The clock-delay between

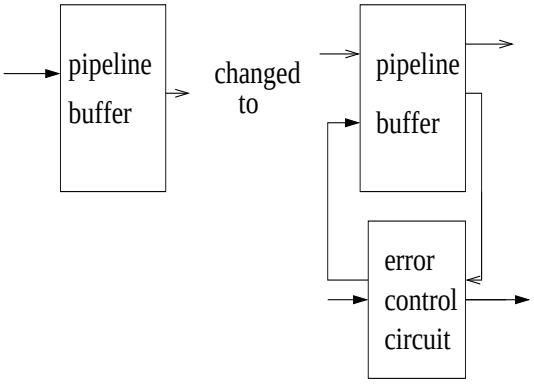

Figure 2.4: Modified Buffer

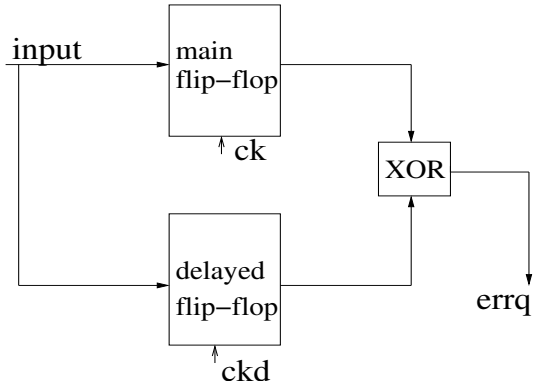

Figure 2.5: Basic Idea

the *main* and the *delayed flops* (i.e. the delay between the clock edges *ck* and *ckd*) is designed such that even in the presence of unpredictability in the wire delay, the correct data bit is obtained at clock edge *ckd*. Thus the *delayed flop* is designed to operate in an error-free manner. The `EXOR` gate is used to compare the data captured by the *main flop* and *delayed flop*. Signal *errq* is high when there is a timing error i.e. when the outputs of the two flops are different. There are two modes of operation of the Terror flop: *normal mode* and *delayed mode*. Initially all the Terror flops on the link are set to the *normal mode* and data transmission begins. In every cycle, at the clock edge *ck*, the *main flop* captures and transmits the incoming data. At clock edge *ckd*, the *delayed flop* captures the incoming data and the error detection circuit checks whether there is any difference between the *main flop* and the *delayed flop* values. If there is a difference, there is an error in the *main flop* value and the data that was transmitted at *ck* is incorrect. The correct data from the *delayed flop* is sent at the next clock edge *ck* and the Terror flop enters the *delayed mode*. Suitable control signals for the downstream buffers/receiver to recover from the error are also generated and sent. A wrong data is sent for one cycle and thus there is one cycle penalty for occurrence of an error.

Once the Terror flop has entered the *delayed mode*, all subsequent data is captured by the *delayed flop* and sent by the *main flop* in the next cycle. Once the Terror flop goes into *delayed mode* it always operates in an error-free manner. This is because the data gets more time to settle to the correct value since it is captured by the delayed flop at rising edge of clock *ckd*. Thus after a single cycle penalty, there is no additional penalty for the rest of the data transmitted through the Terror flop. The same argument applies to all the pipeline buffers of the link. Thus in case of errors,

the maximum (worst-case) overhead in sending data through the Terror-based link is just the number of pipeline buffers in that link, independent of the data size and error-rate. When the data transmission is completed, the Terror flops that are in the *delayed mode* return back to the *normal mode*.

2.3 Logic level Implementation

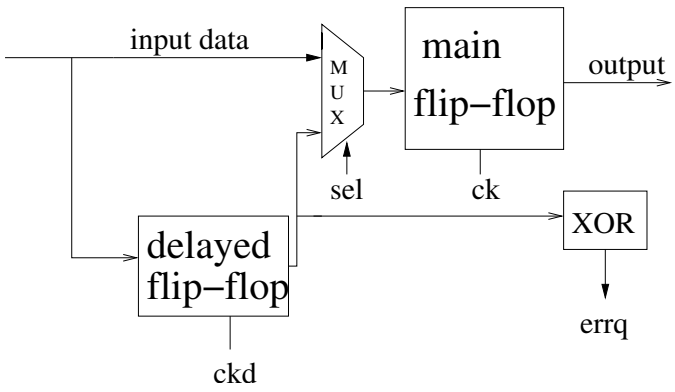

Figure 2.6: Logic level implementation of Terror

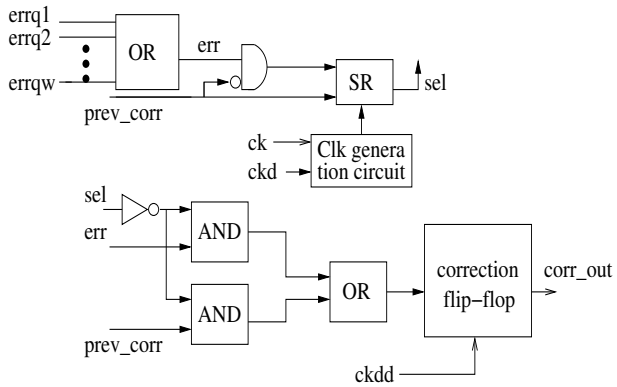

Figure 2.7: Error control circuit

In the Terror-based communication scheme, each pipeline flop is replaced by the Terror flop, which consists of two flops (*main flop* and *delayed flop*), a 2:1 multiplexer (`MUX`) and an `XOR` gate (refer Figure 2.6). The delayed clock *ckd* for the *delayed flop* is derived from the main clock *ck* locally at each buffer by using a *delay chain* (a chain of inverters). At each pipeline stage, an error control circuit (Figure 2.7) is added for generating suitable control signals when an error is detected.

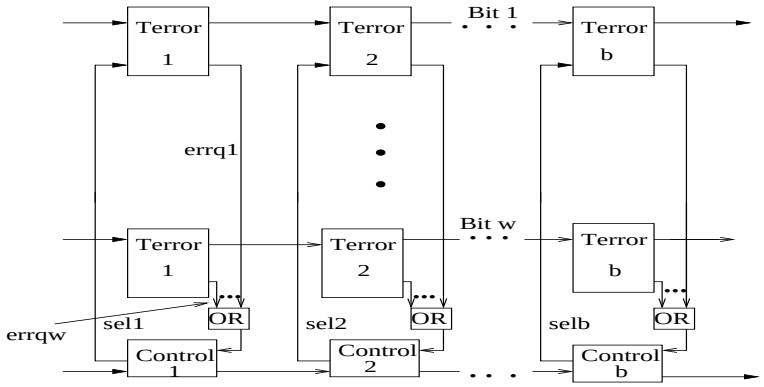

Figure 2.8: Link Design using Terror

Let the number of bit-lines in the link be w lines. The `XOR` outputs (*errq* signals) generated at all the w bit lines, at each pipeline stage of the link are `ORed` and fed as an input to the error control circuit. The error control circuit consists of a `SR` latch, `AND`, `OR` gates and a *correction flop*. For proper operation, the *correction flop* is clocked by *ckdd*, which is delayed from clock *ckd* and locally generated at each pipeline stage. A Terror-based link with w bit-lines (width of the link) and b pipeline buffers (on each bit line) is shown in Figure 2.8. Note that only one error-correction circuit is used at each pipeline stage for all the w bit-lines of the link, so that all the bit-lines of the link are in synchronization with each other. Moreover, the overhead of the error correction circuitry is also reduced by this design.

When an error is detected by any of the Terror pipeline stages, the *err* signal, which is an input to the `SR` latch is set. The `SR` latch output, *sel* is then set to 1, so that the 2:1 `MUX` starts sampling the *delayed flop* output and the Terror flops at this pipeline stage enter the *delayed mode*. Control signal *corr_out* is set to 1 and sent to the next pipeline stage/receiver to indicate that the previously sent data was incorrect (this *corr_out* signal is received as the *prev_corr* signal by the next pipeline stage/receiver). Once the Terror buffer enters *delayed mode*, no more errors occur as the data sampling is through the *delayed flop* (refer Figure 2.6). After all the data is transmitted, the `MUX` control signal is reset to 0 and the Terror buffer returns to *normal mode*. In Figure 2.9, we show an example where an error is corrected in cycle 2 and the Terror buffer operation transitions from *normal mode* to *delayed mode*.

In the above scheme, we note that if the current pipeline stage had an error at clock cycle t and any previous pipeline stage has an error at clock cycle $t+k$ ($k > 0$), then if the current pipeline receives a *prev_corr* signal, it switches back to the *normal*

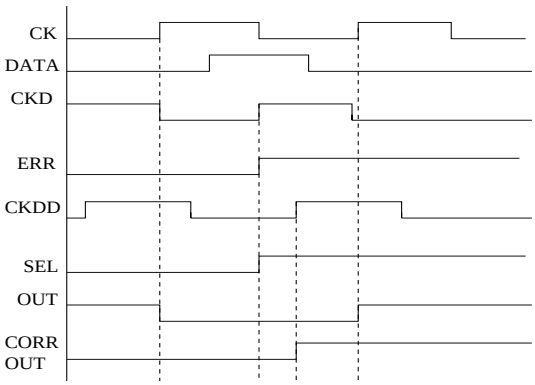

Figure 2.9: Normal to delayed mode

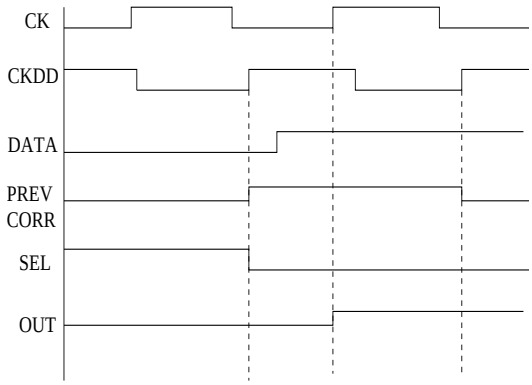

Figure 2.10: Delayed to normal mode

mode and sends the new data coming from the previous pipeline stage. There is no need to resend the data at the current pipeline stage in the next cycle as anyway it is incorrect (because of the error in the previous stage, as indicated by the *prev_corr* signal). This is shown in Figure 2.10. To implement this scheme, the `SR` latch is modified, such that the output is 0 when *prev_corr* is 1 (meaning that the previously received data is wrong and the current data is correct) and the output is 1 when *errq* is one. When a Terror flop returns from *delayed mode* to *normal mode*, the input *prev_corr* is not propagated to the next stage. This is because Terror flop is not sending an incorrect data. For proper Terror operation, the *prev_corr* line used for signaling the occurrence of an error should be error-free. The *prev_corr* line can be made error free by various means such as shielding the line from other bit lines, routing the line in higher metal layer so that it propagates faster, providing parity checker to detect an error in the line, etc. As only a single *prev_corr* line is added to the link that typically has multiple bit-lines, the overhead in shielding or conservatively designing the *prev_corr* line is low.

Figure 2.11 plots the latency for transmitting 1000 bits of data on the link as a function of the delay between the clocks *ck* and *ckd* of the *main* and *delayed* flip flops for an on-chip bus length of $1.2mm$ operating at 1 GHz. For a conservative design the number of pipeline stages on the bus is 6 (assuming $0.2mm$ spacing between adjacent buffers). In a *Terror*-enabled system, since the clock *ckd* can be delayed by one cycle (ideally) from clock *ck* the maximum spacing between consecutive Terror buffers can be twice the clock cycle (as the delayed flop can capture data one cycle late). Thus for an aggressively design link, the number of pipeline stages are halved to 3. The plot in figure 2.11 shows that, as the delay between the clocks, *ck* and *ckd* increases,

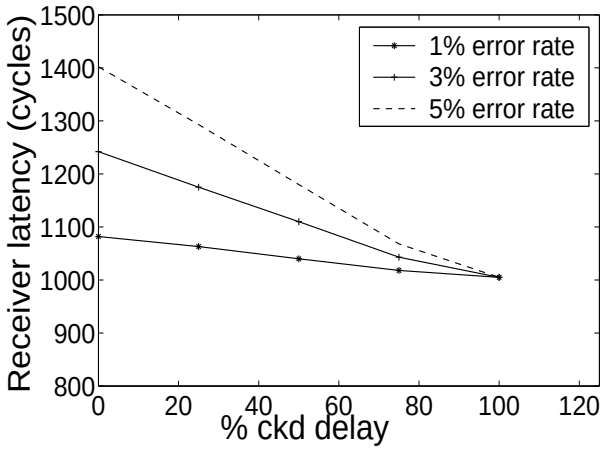

Figure 2.11: Receiver latency variation with delay between clocks *ck* and *ckd* for ideal case

the number of errors detected and corrected by the Terror scheme starts to increase as the *delayed flip-flop* gets a larger time window to sample the incoming data. When the difference between the clocks is less than one cycle, it is assumed that the data bit errors that are not corrected by Terror buffers are retransmitted using an end-to-end flow control mechanism. In most network designs, a *Go Back-N* retransmission strategy is used, where all the data bits following the data with error will be resent [10]. In this case, the latency penalty is much higher. As seen from the plot, there is a significant reduction in the latency as the delay between the clocks *ck* and *ckd* increases.

2.4 Comparison with Alternative Schemes

In this section we compare the performance (latency savings) of a Terror system with a typical error correcting scheme such as Razor [12]. Razor based system has same basic principle as Terror. In a Razor based system, the supply voltage is adjusted by monitoring the error rate. Error recovery is done by delaying the pipeline by one cycle, which results in one cycle penalty per error occurrence. We evaluate the benefits of Razor and Terror systems, in terms of consumption of clock cycles to send data.

For example, consider a Razor based system, in which an error occurs at each cycle. For transmitting N cycles of data, we will need $2N$ cycles. This is because, each error occurrence results in a single cycle overhead. Thus the percentage of useful

cycles is just $N/2N$=50% and the whole pipeline is delayed by N cycles. Now consider a Terror-based system, where the maximum error penalty is limited by the number of Terror elements in the communication link. This is because one cycle penalty for error correction is incurred only for the first occurrence of error at the input of a Terror element. The operation is such that subsequent data transmissions are error-free. Thus, if the total number of Terror elements between the sender and receiver is b, then the percentage of useful cycles is $N/(N + b)$ and the pipeline is delayed by only b cycles. For $b < N$, there are substantial benefits.

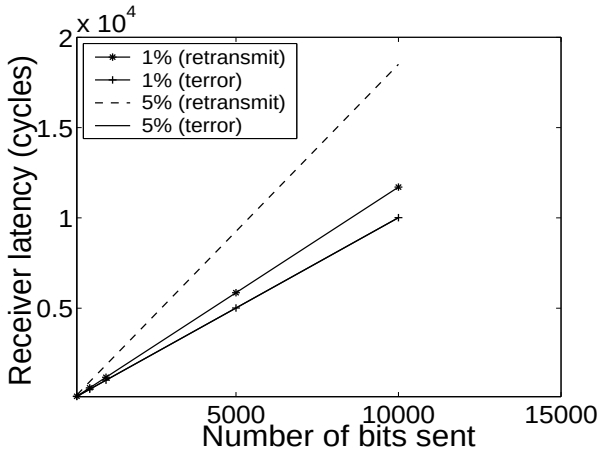

Figure 2.12: Comparison of latency for the retransmission and Terror scheme

Figure 2.12 shows the comparison of receiver latency for a Terror-based scheme and a traditional retransmission based scheme. The latency for data transmission for two different error rates (1% and 5%) is plotted for various data sizes for the two schemes. As seen from the figure, the latency for the Terror system for various error rates is almost equal to the latency for an ideal case when there are no timing errors. For a chosen error rate, as the size of data transferred increases, there is significant latency savings in the Terror system when compared to the traditional scheme of retransmission. Moreover, as the error rate starts to increase, there is much larger savings in latency for the Terror based system. For the data size of 1000 bits and error rate of 5%, there is a 35% reduction in latency in the Terror-based system when compared to the retransmission scheme.

2.5 Timing Analysis

In a Terror-based system, the reduction in latency when compared to a traditional design approach depends on the delay between the clock edges *ck* and *ckd*. Ideally, the clock *ckd* can be delayed by one cycle from *ck*, so that the number of pipeline stages in the link (and hence the latency) is reduced by 50%. This is because delay variations upto 1 clock cycle can be tolerated. Thus the physical distance between adjacent buffers can be doubled and hence the number of buffers can be halved. In practice, the delay between the clock edges *ck* and *ckd* is much lower than one cycle as it is bounded by the delay and timing requirements (setup time, hold time) of the logic elements. Note that the effect of the logic delay can be decreased significantly by optimizing the transistor level implementation of the design.

As shown in figure 2.6, the main flop (clocked by *ck*) has 2:1 `MUX` at its input. This introduces additional delay in the data path. Now the data has to arrive earlier at the input pin as compared to a flop without `MUX` . This increases the setup time of the data input to the Terror element. This increase in setup time is given by t_{mux} (which is the `MUX` delay). Similarly, we have an `AND` gate and an `OR` gate at the input of the *correction flop*, which are in the path of *prev_corr* signal. This increases the setup time requirement of the *prev_corr* signal over the normal set-up time ($t_{setup(nominal)}$). The new set-up time for the *correction flop* is given by:

$$t_{setup} = t_{and} + t_{or} + t_{setup(nominal)}. \tag{2.1}$$

The minimum spacing required between rising edges (assuming all flops are rising edge triggered) of *ckd* and *ckdd* is determined by the total path delay of the *err* signal. The *err* signal has to satisfy the setup time of the *correction flop*. The *err* signal path delay starts from the clock *ckd* to *q* delay of the *delayed flop* , through the `XOR` and `OR` gate (ORs *errq* signals) and then through the AND and OR gates. The clock *ckdd* to the *correction -flop* should arrive such that it captures the correct value of the *err* signal. This correct value of *err* signal is only available sometime after the rising edge of *ckd* and this time delay is the summation of all the delays in the *err* path and the nominal setup time of the *correction flop*. Thus *ckdd* should be spaced from *ckd* to accommodate the *err* path delay. The minimum spacing t_{ckd} between rising edges of *ckd* and *ckdd* is given by the equation.

$$t_{ckd} = t_{ckq} + t_{xor} + t_{or-tree} + t_{and} + t_{or} + t_{setup(nominal)} \tag{2.2}$$

In the case of a bus, the *errq* signals of all bit lines in a link are `ORed`. This ensures that all the bit lines in the link are in synchronization with each other, simplifying the receiver design. The `OR` tree can be implemented as a domino gate to reduce delay. This domino gate can precharge when clock *ckd* is low and evaluate when *ckd* is high.

There is some delay for Terror to go from *delayed* to *normal mode*. Terror goes from *delayed* to *normal mode* when *prev_corr* is set to one. This resets the `SR` latch output (*sel* signal). This *sel* signal is an input to the 2:1 `MUXs`. For correct functionality, the *sel* signal should change value before the rising edge of *ck*. The minimum spacing between rising edges of clocks *ckdd* and *ck* is determined by the total path delay of the *sel* signal. The *prev_corr* signal satisfies the setup time of the *correction flop* and hence comes sometime before the rising edge of *ckdd*. Path delay of *sel* signal starts when *prev_corr* arrives, then it goes through the `SR` latch and then the 2:1 `MUX` . Minimum spacing (t_{ckdd}) between *ckdd* and *ck* is the total path delay minus the setup time of the *prev_corr* signal.

$$t_{ckdd} = t_{SRlatch} + t_{mux} - t_{setup(correctionflop)} \tag{2.3}$$

In the error correcting circuit, the *err* signal (which is the output of the `OR` of the *errq* signals) asserts the *sel* signal. The *sel* and *err* signals are also fed as inputs to the *correction flop* of the error control circuitry. Thus *sel* should not change before t_{hold} of the *correction flop*. It should satisfy below hold-time condition:

$$t_{hold} < t_{SRlatch} + t_{and} + t_{or} \tag{2.4}$$

The timing delays and overheads can be minimized by transistor level optimizations such as including the input `MUX` into the flop, using a domino OR gate instead of a static OR tree, using a simplified latch design, combining `AND-OR` logic into the *correction flop*. Figure 2.13 shows the optimized transistor-level circuit diagram where flip-flops with embedded logic have been used.

The clock (*ckd*) to the *delayed flop* could ideally be delayed by one cycle, such that even if the data arrives one cycle late it is captured and sent the next cycle. But due to timing overheads this window is decreased by ($t_{ckdd} + t_{ckd}$). This gives an upper bound on the frequency by which the clock cycle can be decreased beyond

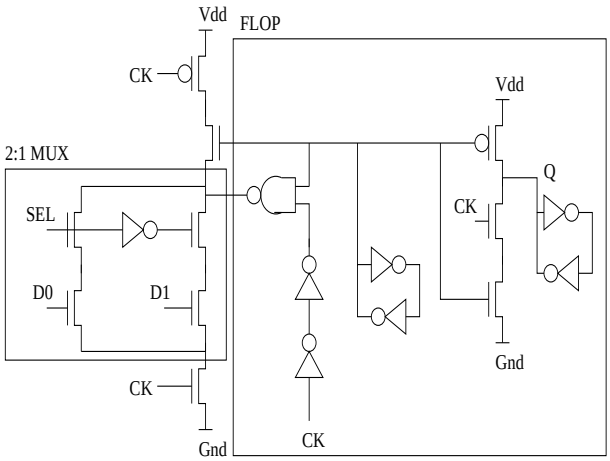

Figure 2.13: Semi-Dynamic Flip Flop Design

Table 2.1: Timing Overheads

Parameter	% Overhead
Hold Time	10
32-bit OR delay	8.6
t_{ckdd}	9.7
t_{ckd}	27.0
ckd delay	36.7

specifications.

2.6 Transistor Level Simulation

A transistor level Terror element was designed for a 32 bit bus in 100 nm technology targeted for 1GHz operating frequency, operating at 1.2 V. From SPICE simulations, we obtained values for timing overheads, presented in Table 2.1, where the overheads are expressed as a percentage of cycle. This table is an estimate of the timing constraints and the timing overheads can be reduced substantially by using better transistor sizing techniques, process technology and commercially available CAD tools.

The table 2.1 shows that practically clock *ckd* cannot be delayed beyond 63.3% of cycle time. This is because there is a minimum spacing requirement $t_{ckd} + t_{ckdd} =$ 36.7% which should be satisfied. Hence only 100-36.7 = 63.3% of cycle is available for *ckd* delay. Also, to satisfy hold time requirement (10%) of the *main flop*, minimum spacing between *ck* and *ckd* should be 10% of the cycle. Due to this, range for

variation of *ckd* is limited to 53.3% of the clock cycle. To simplify calculations, we use the range for variation of *ckd* as 50.0% of the clock cycle.

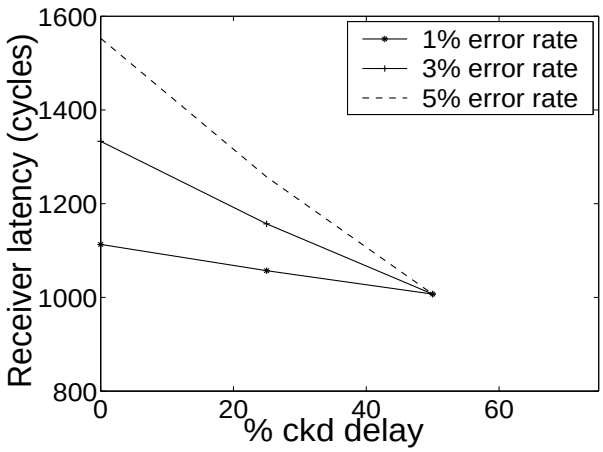

Figure 2.14: Receiver latency for practical case

Figure 2.14 shows the latency for transmitting 1000 bits of data for various error rates as a function of the delay between the clocks *ck* and *ckd* for the practical case. Using conservative design approach, the number of buffers on the link is 6. Ideally the delay between clocks *ck* and *ckd* can be one cycle which reduces the number of buffers on the link by 50% (3 buffers on the link). Practically, due to timing overheads associated with the flip-flops and the logic elements, the delay between clocks *ck* and *ckd* can be increased only by half the clock cycle, so that the on-chip link is pipelined with 4 stages. Thus in figure 2.14 the percentage *ckd* delay stops at 50%.

Errors due to meta-stability of data, as discussed in [12] can occur at the input of the *delayed flop*. These can not be completely eliminated but can be minimized. Since we use a *delayed flop* instead of a delayed latch used in [12] we significantly minimize short-path constraint problem.

2.7 Analysis of Penalty

The maximum latency penalty in the Terror system is bounded by the number of pipeline stages in the link and is independent of the amount of data sent and the error rate. This is because, a single cycle latency penalty is incurred at a pipeline stage only for the first detection and correction of an error. Once an error occurs at a pipeline stage, the pipeline stage enters the *delayed mode*, so that subsequent data transmission at this pipeline stage is guaranteed to be error free.

The actual latency penalty is a function of temporal and spatial probability of error occurrence. For a Terror link with b pipeline stages, we get

$$1 \quad \leq \quad Penalty \quad \leq \quad b \tag{2.5}$$

The magnitude of penalty is a function of when and where the timing error occurs in the pipeline. In Figure 2.3, if a timing error occurs first at the pipeline stage b, followed by an error at pipeline stage $(b-1)$ and so on up to the pipeline stage 1, then the total penalty for error correction is just one cycle. This is because the error in pipeline stage $(b-1)$ is absorbed by Terror at pipeline stage b (since Terror b goes from *delayed* to *normal mode*, incoming error is not propagated by Terror b).

On the other hand, if a timing error occurs first at the pipeline stage 1, followed by an error at pipeline stage 2 and so on up to Terror b, then penalty for error correction is b cycles. This is because at each pipeline stage, a single cycle penalty is incurred for error detection and correction.

Thus, the maximum penalty in the Terror scheme is b cycles, while the actual penalty lies between 1 and b cycles. Also, we note that maximum penalty is independent of the total amount of data sent. This makes Terror design very attractive for high bandwidth data communication of current and future SoCs.

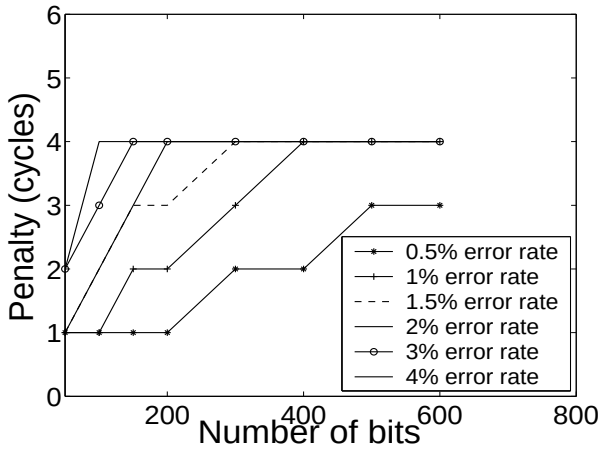

Figure 2.15: Terror penalty for different data sizes

Figure 2.15 shows the variation of maximum penalty for error correction in the Terror system, with respect to the total number of bits sent on the link. With increase in error rate and size of data transmitted, the penalty increases until it reaches 4. After the penalty of 4 cycles, which is the number of pipeline stages in the link, the

penalty is constant with increasing data size and error rate. Note that a typical error correction mechanism would degrade at higher error rates and large data size. But in Terror enabled system, the latency overhead does not increase with error rate or data size, making it suitable for large bandwidth data transmission of SoCs. Moreover, as explained in [12], the system can be operated at lower voltage levels when errors are permitted to occur in the system. In such a design, significant power savings can be achieved as the error-rate increases and Terror-based communication mechanism supports such a design.

Chapter 3

Network on Chips Design Methodology

This chapter describes Network-on-Chips Design Flow, which is the background information required for next chapter, to describe Terror enabled NoC links. This information is given here only for the sake of completeness.

The growing complexity of customizable single-chip multi-processors has necessitated the use of a highly-scalable communication infrastructure. The system architecture is becoming more communication centric than computation centric and hence efforts are made to design efficient and reliable communication networks. Growing number of *Network-on-Chip* (NoC) architectures have been proposed recently for *System-on-Chip* (SoC) integration. Depending upon the application requirements, a NoC design framework can derive an optimized configuration with respect to different design objectives and instantiate the selected application specific on-chip micronetwork. The NoC design flow `NetChip` partitions the development work into major steps (topology mapping, selection and generation) and provides proper tools for their automatic execution (`SUNMAP` [32], ×`pipesCompiler` [33]).

3.1 Introduction

A typical SoC is a highly complex system consisting of building blocks from multiple sources (either in-house made or externally supplied), such as general-purpose fully programmable processors, co-processors, DSPs, dedicated hardware accelerators, memory blocks, I/O blocks, etc. A scalable communication architecture that supports

the trend of SoC integration consists of an on-chip packet-switched micro-network of interconnects, generally known as Network-on-Chip [3, 27, 28]. Interface protocols such as *Virtual Component Interface* (VCI) [29] and *Open Core Protocol* (OCP) [30] are used for integrating domain-specific computation resources. In the NoC design methodology, various computing tasks at the application level are captured into models of computation. With the help of a mapping tool, area and power estimates are made and feasibility of different topologies is explored. An instantiation tool builds an architecture of network components depending upon the mapping information.

3.2 NoC Design Flow

The design flow of `NetChip` is presented in Figure 3.1. `NetChip` assumes that the application has already been mapped onto cores by using pre-existing tools (such as [31]) during the hardware/software co-design phase. The resulting cores together with their communication requirements represent the inputs to our NoC synthesis flow. The average rate of data transfer between the cores is determined by static analysis or simulation. A graph called *core graph* is made which represents the resulting cores and communication demands.

`NetChip` has three phases of operation: *topology mapping phase*, *topology selection phase* and *topology generation phase*. `NetChip` in-turn has two tools built into it: `SUNMAP` [32] which performs the topology mapping and selection phases and the ×`pipesCompiler` [33] which generates the selected topology.

In the *topology mapping phase*, `NetChip` takes as inputs:

- the core graph with communication among cores annotated as edge weights
- the design objective function that needs to be optimized
- the design constraints that are to be satisfied by the mapping.

Netchip has a *Graphical User Interface (GUI)* designed in `TCL/TK` for entering the inputs. The input core graph is then mapped onto various standard topologies (*mesh*, *torus*, *hypercube*, *Clos* and *butterfly*) defined in the topology library. `Netchip` explores various design objectives such as minimizing average hop delay, area and power dissipation. In the *topology selection phase*, the various topologies (with mappings produced from the mapping phase) are evaluated for several design objectives

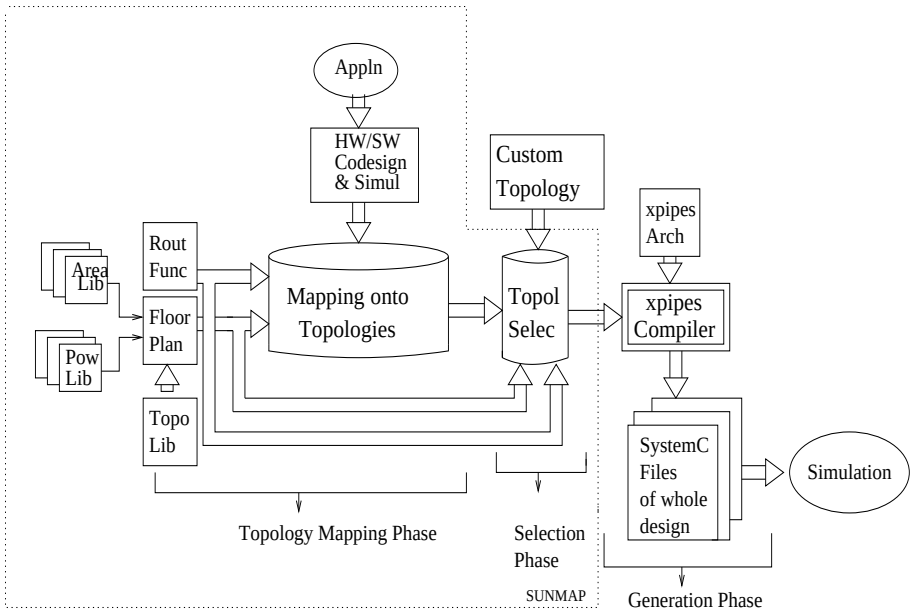

Figure 3.1: NoC Design Flow

and the best topology for the application is chosen. The design file describing the selected topology and routing files describing the routes (or paths) to be taken (which depends on the chosen routing function) are automatically generated.

In the *topology generation* phase, `NetChip` reads the design and routing files and generates SystemC description of network components for the selected topology using `×pipesCompiler`. The network components generated are optimized for that particular network and support reliable, latency- insensitive operation.

3.3 NoC Design tools

3.3.1 SUNMAP: Topology mapping tool

`SUNMAP` produces a mapping of cores onto various NoC topologies that are defined in a topology library. The mappings are optimized for the chosen design objective (such as minimizing area, power or hop delay) and satisfy the design constraints (such as area or bandwidth constraints). `SUNMAP` uses floorplanning information early in the mapping process to determine the area-power estimates of a mapping and to produce feasible mappings (satisfying the design constraints). The tool supports various routing functions (dimension ordered, minimum-path, traffic splitting across minimum-paths, traffic splitting across all paths) and chooses the mapping onto the

best topology from the library of available ones.

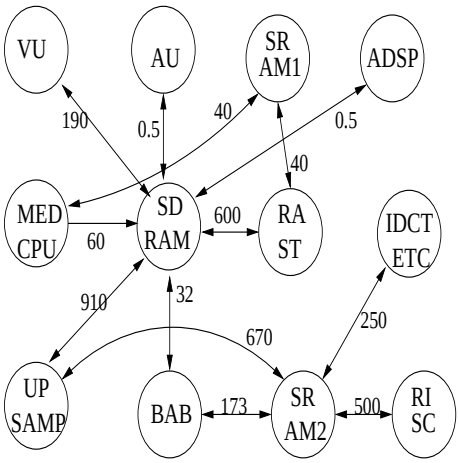

Figure 3.2: Video Object Plane Decoder Core Graph

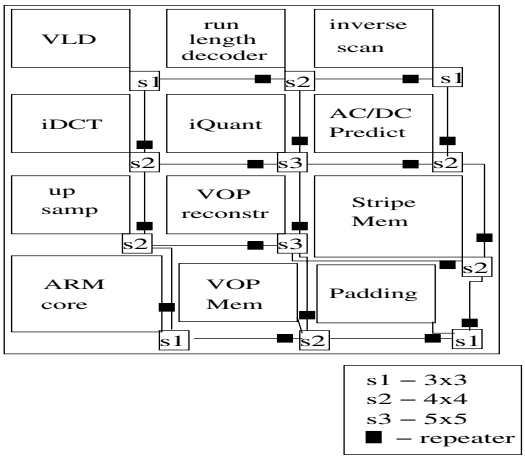

Figure 3.3: Mapping of VOPD on a mesh

Figure 3.2 shows the the core graph of Video Object Plan decoder and figure 3.3 shows the mapping on a Mesh topology. In the VOPD, about half the cores communicate to more than a single core. This motivates the configuration of this custom NoC, having less than half the number of switches than the mesh NoC. Figure 3.4 shows custom-mapping of the VOPD. By using the area-power models built into `NetChip`, the area and power consumption of the network components of the custom NoC were automatically obtained. Significant area and power improvements are obtained with the custom NoC, as fewer number of switches are used and the switches have smaller size than the mesh switches as shown in Table 3.5

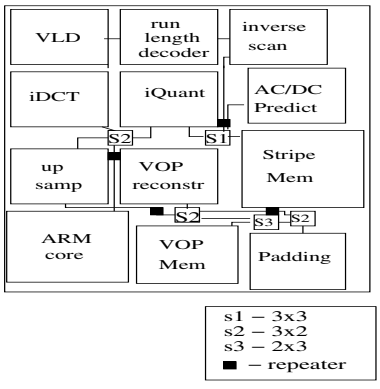

Figure 3.4: Custom mapping of VOPD

Figure 3.5: Custom and Mesh Mappings of VOPD

Param	Mesh	Custom	Ratio
Area (mm^2)	1.26	0.22	5.73
Power (mW)	108.74	40.08	2.71

Table 3.1: Switch Area and Energy

Size (in×out)	Area (mm^2)	Energy (pJ/bit)
1×1	0.018	7.08
2×2	0.037	21.94
3×3	0.08	45.96
4×4	0.10	79.08
8×8	0.74	313.04

Area-power models are obtained through tools such as ORION [34] for 0.1μ technology node. Wiring parameters from [4] can be used to estimate link power dissipation. Area and power consumption for some example switch configurations are given in Table 3.1.

3.3.2 Xpipes Compiler

×`pipesCompiler` is used to generate a customized NoC configuration. ×`pipesCompiler` uses the ×`pipes` library, which consists of highly parameterized network components that can be tailored to the communication needs of the selected architecture.

×`pipesCompiler` offer high degree of parameterization in which both global network-specific parameters and local block-specific parameters can be changed as required. These parameters include flit size, degree of redundancy of error control logic, address space of the cores, maximum number of hops between any two nodes, maximum number of bits allocated within a packet for end-to-end flow control etc. Also parameters related to network interface such as the type of interface (master, slave or both), flit buffer size at the output port and other interface parameters to the cores such as number of address/data lines, maximum burst length, etc can also be changed. In the NoC, data is transmitted in the form of packets. A packet is divided into flits. Flits can be of various types such as head flit, tail flit or payload flit as shown in figure 3.6

For the VOPD application described in previous section, cycle-accurate simulations of the NoCs were performed. Two-state Markov Models as stochastic traffic generators were used to model the bursty nature of the application traffic, with average communication bandwidth matching the applications' average communication bandwidth. Snapshots of SystemC simulations of mesh and custom NoCs for some of the cores of VOPD are shown in Figure 3.7. The time between transmission of a flit

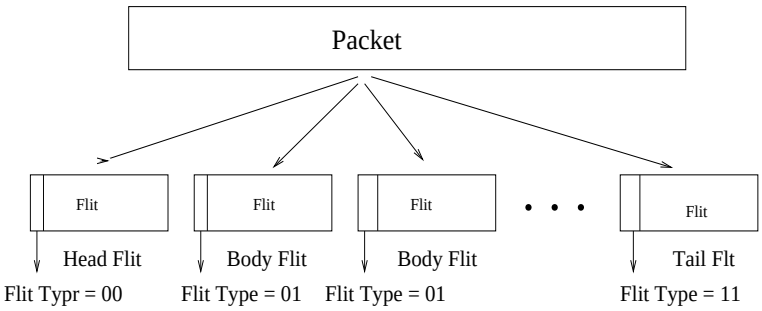

Figure 3.6: Packet Partitioning into Flits

and its reception, which includes the switch delay, link delay and contention delay, is marked in the Figure 3.7 . The variation of average packet latency (for 64 byte packets, 32 bit flits and 7 cycle switch delay) with link bandwidth is shown in Figure 3.8 Application-specific NoCs have lower packet latency as the average number of switch and link traversals is lower. Moreover, the latency increases more rapidly for the mesh NoCs with decrease in bandwidth. With the custom NoC, we achieve an average of 25% savings in latency (see Figure 3.8).

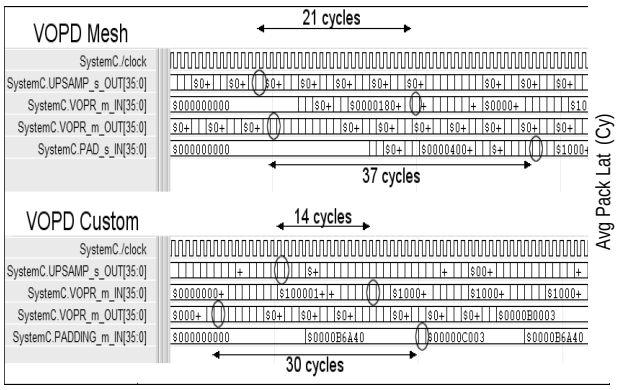

Figure 3.7: SystemC output for VOPD simulation

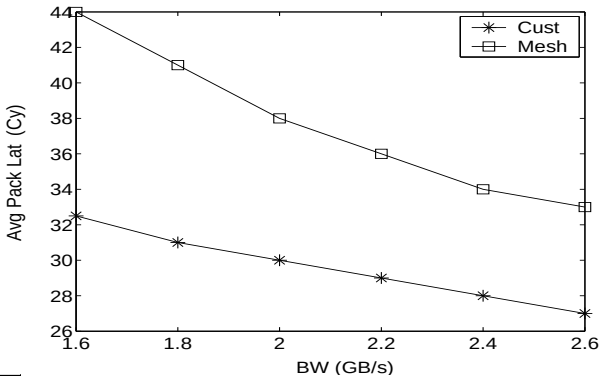

Figure 3.8: VOPD average latency

3.4 NoC Design components

NoC Link

Switch-to-switch links are subdivided into basic segments whose length guarantees that the desired clock frequency (up to the maximum speed achievable by a certain technology) can be used and that the system operating frequency is not bound by the delay of the longest link. Depending on the specific link length, a certain number

of clock cycles is needed by a flit to cross that interconnect. As shown in figure 3.9 typically, multiple outstanding flits propagate across the link during the same clock cycle. When flits are correctly received at the destination switch, an ACK is propagated back to the source, and after N clock cycles (where N is the length of the link expressed as number of repeater stages) the flit will be discarded from the buffer of the source switch. On the contrary, a corrupted flit is NACKed and will be retransmitted in due time.

By means of a proper buffering policy, network switches have been designed in such a way that their functional correctness depends on the flit arriving order and not on their exact timing, so that input links of the switches can be different and of any length.

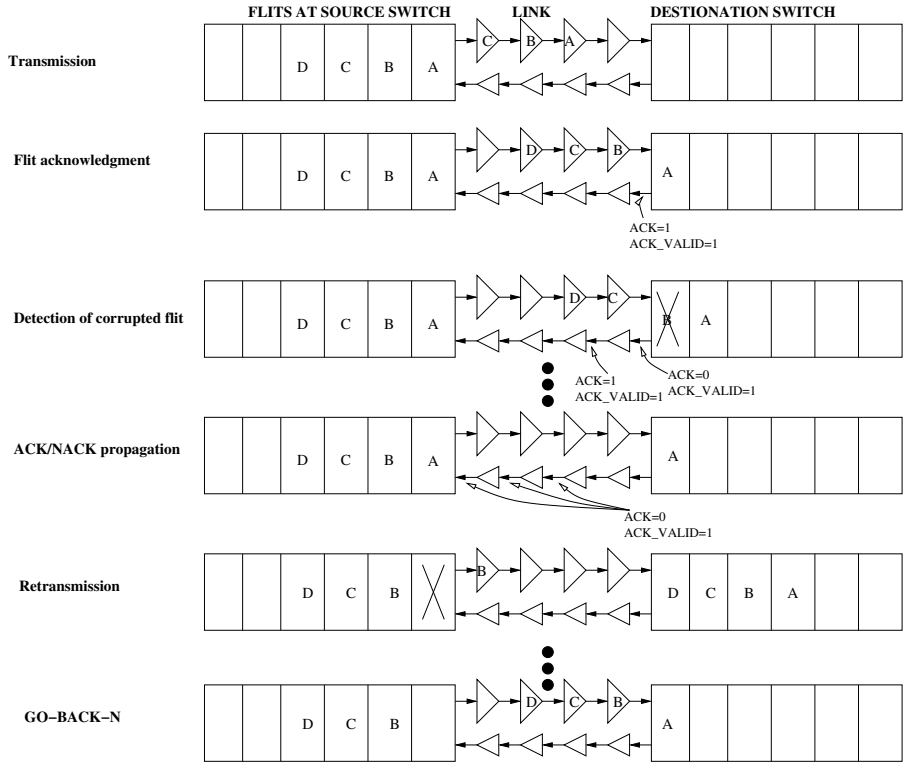

Figure 3.9: Pipelined Link Model and latency insensitive link level error control

Noc Switch

NoC switch uses output buffering and is pipelined to maximize the operating clock frequency of the switch. All switch inputs are connected to the inputs of each output module. Flow-control signals generated by each module (such as ACK and NACK

for incoming flits) are collected by a centralized switch unit, that directs them back to the proper source switch. The CRC decoders for error detection work in parallel with the switch operation, thereby hiding their latency.

For latency insensitive operation, the switch has virtual channel registers to store $2N+M$ flits, where N is the link length (expressed as number of basic repeater stages) and M is a switch architecture related contribution (12 cycles in this design). The reason is that each transmitted flit has to be acknowledged before being discarded from the buffer. Before an ACK is received, the flit has to travel across the link N cycles, an ACK/NACK decision has to be taken at the destination switch (a portion of M cycles), the ACK/NACK signal has to be propagated back (N cycles) and recognized by the source switch (remaining portion of M cycles). During this time, other $2N + M$ flits have been transmitted but not yet ACKed. Figure 3.10 shows a typical switch configuration with two virtual channels.

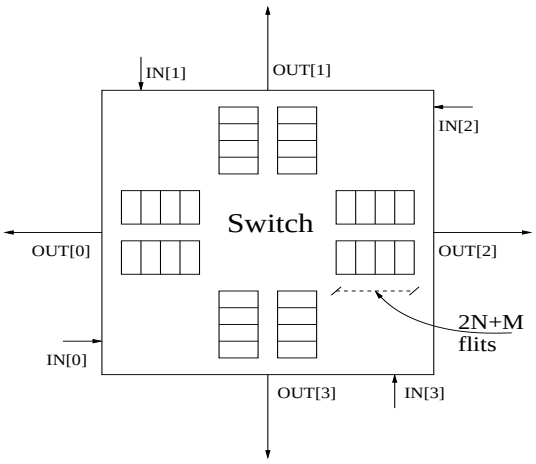

Figure 3.10: Example of 4×4 switch with two virtual channels

NoC Network Interface

×`pipes` Network interface communicates with cores using OCP and performs protocol conversion to adapt to network protocol. The NoC architecture uses wormhole switching and static routing. Routes are obtained by the network interface by accessing a look-up table based on the destination address. Each route is represented by a set of direction bits. Each switch directs the flits belonging to a certain packet to the particular output port, based on the direction bits.

The ×pipesCompiler

A high-level description of NoC as created by ×pipesCompiler consists of the definition of the cores, network interfaces, switches, links and their interconnections. The number of pipeline stages of each link is also dictated, based on link length estimations by the floorplanner and target clock speed. The ×pipes library of SystemC soft macros is used by ×pipesCompiler to generate the network components for the chosen topology. The output of ×pipesCompiler is a SystemC hierarchical description of all the switches, links, network nodes and interfaces that specifies their topological connectivity. The structure of the SystemC output can be optimized either for simulation or synthesis. The final description can be compiled and simulated at the cycle-accurate and signal-accurate level, and can be synthesized by back-end RTL synthesis tools for silicon implementation.

Chapter 4

Terror enabled NoC link designs

As devices shrink and interconnect wire delay becomes more susceptible to cross-talk, coupling, process variations and other noisy interferences, reliable link design approaches are needed where the NoC link is tolerant to variations in delay caused by the noisy environment. This chapter describes three *robust* link design methodologies (based on Terror) that provide different levels of reliability for data transferred on the links with different performance and hardware complexity. The link buffers (flip-flops) are used to store data, so that the link is used as a *storage medium*, rather than just a *communication medium*. By distributing the input buffers of a NoC switch on to the links, the number of buffers used in the switches of the NoC reduces considerably (up to 33%) when compared to traditional input-queued switches, while the performance of the NoC remains unaffected. These enhanced link designs use an aggressive design approach, where the links are designed for normal (ignoring data-dependent delay variations) operating conditions, instead of worst-case operating conditions. Terror (Timing Error-Tolerant) design methodology is extended in order to cope with the timing errors that may occur in the aggressive design approach. All the three designs have the same terminal interfaces such that the three designs can be interchanged in a *plug* and *play* fashion.

4.1 Storing data on the link, not on the switch

This section explains the distributed buffering scheme where the link is used as a *storage medium* rather than just a *communication medium*. The buffer (flip-flop) on a NoC link is replaced by a two-entry FIFO, which is similar to the Terror flop

as shown in figure 2.6 (without the $EXOR$ gate). Figure 4.1 shows a 3-stage link pipeline using 2-entry FIFO at each pipe-line stage. The scheme has two control inputs - *stall* and *valid* signals. *Stall* signal is sent by the receiver and flows in the opposite direction to that of the data, while *valid* signal is sent by the sender and it flows in the same direction as that of the data.

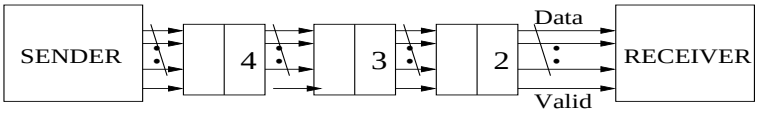

Figure 4.1: Modified link design with 3 stages

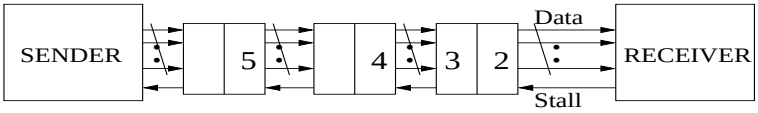

Figure 4.2: Entry 3 buffered in *secondary flip-flop*

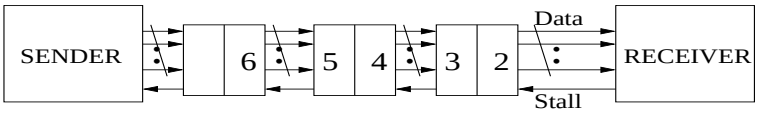

Figure 4.3: *stall* signal propagated to previous stage

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 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 4.1. When a *stall* signal is received by the 2-entry FIFO (shown in Figure 4.2), then data on output of the *main flip-flop* is stalled and new data is received in the *secondary flip-flop*. The *stall* is propagated to the previous stage as shown in Figure 4.3. The schematic of the 2-entry FIFO is shown in Figure 4.4.

As shown in Figure 4.4, when a *stall* is received, the *main flip-flop* is stalled, and the new data is captured by the *secondary flip-flop*. The control circuit generates the *stall* signals for the *main flip-flop* and *secondary flip-flop* and the *muxselect* signal. The control circuit has two inputs - *stall* signal and *valid* signal and five outputs - *stall* and *valid* signals for the previous and next stages respectively, and three control

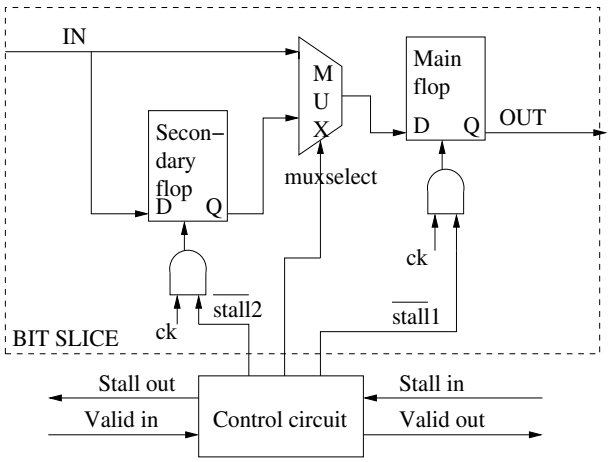

Figure 4.4: Two-entry FIFO

signals for controlling the operation of the two-entry FIFO (shown in Figure 4.4). In the traditional input-queued scheme, the number of buffers needed at each input of the switch and the link is $3b$ ($2b$ at the switch input and b on link) [46] as compared to the $2b+2$ buffers ($2b$ at the link and 2 at the switch input) in this scheme. Moreover, as all the inputs of a switch have same buffer count, the switch design becomes more modular, when compared to the traditional switch design. The control circuit used at link pipeline stage in this scheme is common for all the w data bits of the NoC link, and thus the overall cost of the control circuit is negligible.

4.2 Reliable buffered link design

This sections presents three *robust link design* methodologies, based on *Terror* to detect and correct the timing errors that can occur in a NoC link, without substantially affecting the performance of the NoC. In the first scheme, timing errors are only detected and are not corrected. Error correction is assumed to be done by retransmission. The second scheme can detect and correct timing errors. Error penalty in the second scheme is only dependent on the number of buffers on the link and does not depend upon the error rate. The third scheme can also detect and correct timing errors but its penalty increases with the number of errors. In all these approaches the semantics (timing relation and logical interpretation) of the control signals, namely, *stall* and *valid* signals remain same. Thus the switch-to-link interface for the schemes remains the same and hence the different link design schemes can be interchanged in a *plug and play* fashion.

4.2.1 Scheme 1: Terror Detection

To make the design more resilient to timing errors, delayed clock *ckd* is used to trigger the *secondary flip-flop* (henceforth called *delayed flip-flop*) of the two-entry FIFO. The delay between the clock edges *ck* and *ckd* is designed such that, even in the presence of unpredictability in the wire characteristics, there is sufficient time for the data to reach the *delayed flip-flop* by clock edge *ckd*. Thus the data captured by the *delayed flip-flop* is always timing-error free. The *main flip-flop* is clocked aggressively such that the data captured by the *main flip-flop* can have a timing error. The delayed clock *ckd* is derived locally from clock *ck* by using a delay chain (chain of inverters).

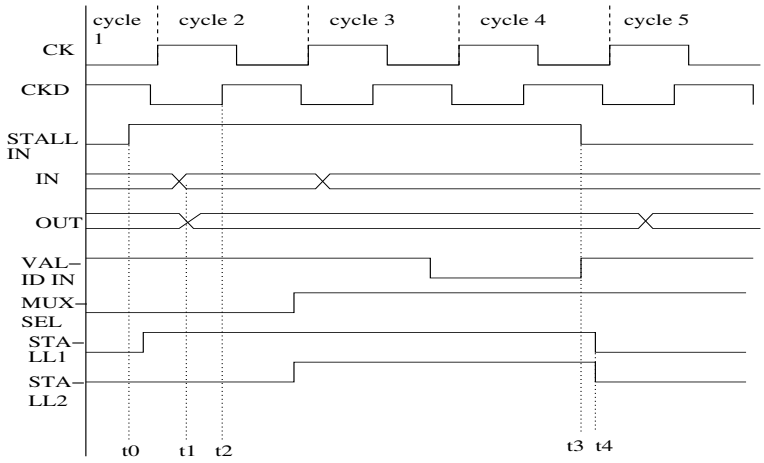

Figure 4.5: Waveforms for reliable scheme

Figure 4.5 explains the operation of this scheme. The *stall* signal arrives at time instant t_0 which causes the *main flip-flop* to stall. Suppose due to coupling noise or other interferences, the delay of the wire segment increased and hence the correct value of input data is available at time instant t_1. Since the clock *ckd* to the *delayed flip-flop* was delayed, it captured the correct value of data at time instant t_2. When the *stall* signal becomes low (at t_3) the *stall1* and *stall2* signals are set to zero by the control circuit (at t_4) and the *main flip-flop* sends the data that was captured by the *delayed flop-flop*. When the *main flip-flop* gets stalled, the input *stall* signal is propagated to the previous stage. The *main flip-flop* or the *delayed flip-flop* is not stalled if the data captured is not valid. Since the *delayed flip-flop* always captures data at a delayed clock *ckd*, the input data always has more time to settle to the correct value. Thus timing errors do not occur when the data is captured by the *delayed flip-flop*. Penalty for timing error occurs when data is captured by the *main*

flip-flop and when there is no *stall* signal.

The control circuit that generates the *muxselect*, *stall1* and *stall2* signals is shown in Figure 4.6. The control circuit is a FSM having two states - delayed state and normal state. In normal state, data is captured by the *main flip-flop*, while in delayed state data is captured by the *delayed flip-flop*.

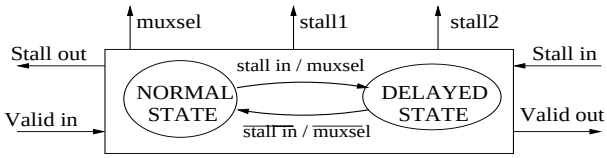

Figure 4.6: Control circuit FSM for scheme 1

Additional signals and some flow control mechanism (either switch-to-switch or end-to-end retransmission) is used to detect and correct the timing errors that occur when the data is captured by the *main flip-flop*. The use of the delayed clock adds negligible hardware overhead to the link buffer scheme, but avoids many of the timing errors. An interesting fact about the scheme is that, when the the network has significant congestion most of the timing errors are avoided. This is because, most of the data in the congested network passes through the *delayed flip-flop*, as the *main flip-flop* stalls more often due to congestion. Effect of retransmitting data has more impact on performance of a heavily loaded network, as latency penalty for retransmission is much higher on a congested network than on a lightly loaded network. Moreover, retransmission also increases the congestion on the network, thereby indirectly increasing the latency for the packets that don't incur timing errors. Thus, this simple scheme for avoiding timing errors is in fact sufficient for many systems.

4.2.2 Scheme 2: Terror Detection and Correction

The previous scheme detects timing errors but corrects them, only when the data is captured by the *delayed flip-flop*. When data is captured by the *main flip-flop*, error correction is done by retransmission. We enhance scheme 1 such that all timing errors are detected and corrected, irrespective of which flip-flop captures the data. The semantics for *stall* and *valid* signals remain same for both the schemes.

Figure 4.7 shows the schematic for this scheme. A three-entry FIFO (instead of

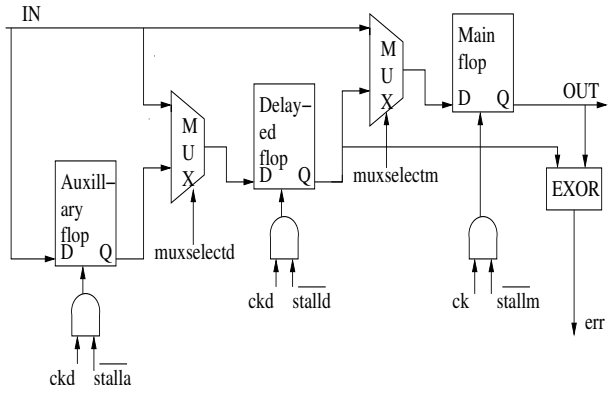

Figure 4.7: Schematic for scheme 2

the two-entry FIFO used in the previous scheme) is used in this scheme. A third flip-flop, called *auxiliary flip-flop* is added in series with the *delayed* and *main flip-flops*. An `EXOR` gate is connected to the outputs of the *main flip-flop* and *delayed flip-flop*. It compares the data captured by the *main flip-flop* (at rising edge of clock *ck*) and the data captured by the *delayed flip-flop* (at rising edge of clock *ckd*) and set its output *err* to 1, if the two values are different i.e., if a timing error has occurred. The *err* signals of all w buffers (vertically across the width of the bus) are `ORed` and fed as an input to the control circuit.

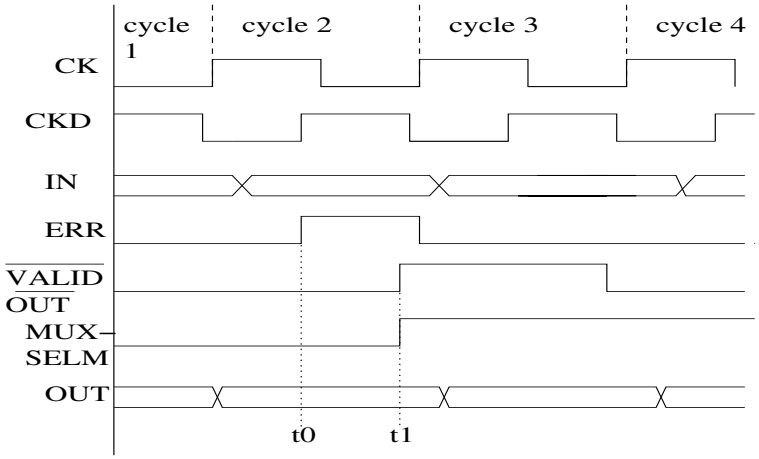

Figure 4.8: Waveforms for Scheme 2

As shown in Figure 4.8, during normal operating condition data is captured and sent by the *main flip-flop*. When a timing error occurs, *err* is set to one at time instant t_0 and *muxselectm* is set to one at t_1 such that the output of *delayed flip-flop* is connected to input of *main flip-flop*. The *main flip-flop* sends the correct data (captured by the *delayed flip-flop*) in next cycle (*cycle* 3). The *muxselectm* is reset

to zero when the input data is not valid. When the data is sampled through the *delayed flip-flop* and a *stall* signal arrives, the incoming data is stored in the *auxiliary flip-flop*. This scheme incurs one cycle penalty for the first occurrence of a timing error, and as long as there is continuous data flow on the link, further timing errors are avoided.

Figure 4.9 shows the control logic for this scheme. It is a FSM with three states - normal state, delayed state and auxiliary state. In normal state, only *main flip-flop* is used. In delayed state, *delayed flip-flop* and *main flip-flop* are connected in series, while in auxiliary state, all the three flops are connected in series. The control circuit determines the current state based on the previous state and the status of the input signals.

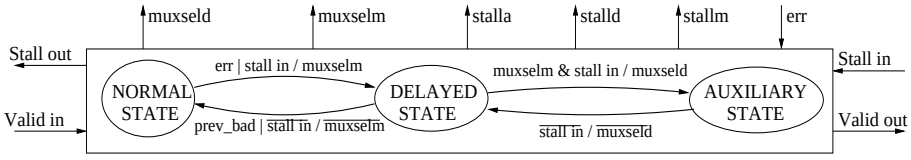

Figure 4.9: Control circuit FSM for scheme 2

4.2.3 Scheme 3: Simplified Terror Correction

Scheme 3 is a simplification of scheme 2, with less area. In scheme 3, the error penalty increases with the error rate while in scheme 2, error penalty does not increase with the error rate, but is bounded by the number of buffers on the link between the sender and the receiver. In scheme 3, one cycle penalty is present for each occurrence of error. Compared to previous scheme, scheme 3 has simpler control logic and uses a two-entry FIFO instead of a three-entry FIFO.

In normal operation, input data is captured and sent by *main flip-flop* on the rising edge of clock *ck*. As shown in Figure 4.10, an `EXOR` gate is connected to the outputs of the *main flip-flop* and *delayed flip-flop* to detect a timing error. 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 on previous cycle was non-valid. The current stage (which had a timing error) sends the correct data on the next cycle. Each occurrence of timing error causes a stall of one cycle.

The operation of this scheme is shown in figure 4.11. The value of data captured

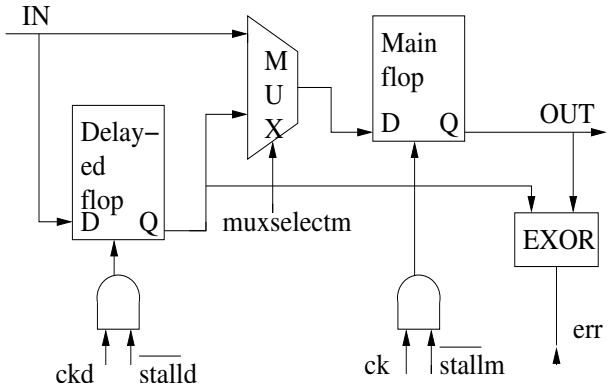

Figure 4.10: Schematic for Scheme 3

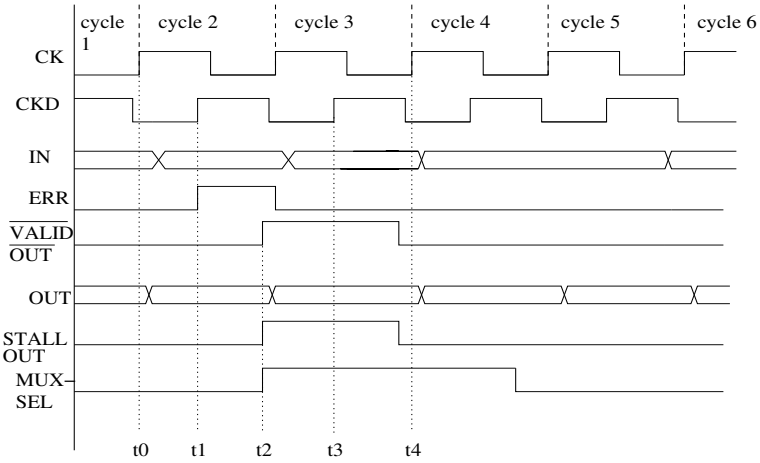

Figure 4.11: Waveforms for Scheme 3

by *main flip-flop* at time instant t_0 is different from the value captured by *delayed flip-flop* at time instant t_1. Hence a timing error has occurred and the *err* signal is set high at t_1. The control logic sets the *muxselectm* signal to one which connects the output of the *delayed flip-flop* to the input of the *main flip-flop*. A *stall* signal is also generated (at t_2) for one cycle which is sent to the previous stage to stall its output. Hence the previous stage does not send a new data on the next clock cycle. A $\overline{valid}$ signal is sent (at t_2) to the next stage informing that the data sent on previous cycle was not valid. The *main flip-flop* sends the correct data (that was captured by the *delayed flip-flop*) in *cycle 3*. At time instant t_3 *delayed flip-flop* captures new data and at time instant t_4 in *cycle* 4 *main flip-flop* sends this captured data. Due to the *stall* signal the previous stage does not send new data in *cycle 4*. The *muxselectm* is reset to zero in *cycle 4* due to which the *main flip-flop* input is connected to the data input (IN). The *main flip-flop* sends a new data (which was stalled by previous

stage in *cycle 4*) in *cycle 5*. Since the previous stage was stalled only for one cycle, it sends a new data in *cycle 5* which is captured and sent by the *main flip-flop* of the current stage at *cycle 6*. If the data captured by the *delayed flip-flop* is not valid, then *muxselectm* is not set to one and a *stall* signal is not sent to the previous stage.

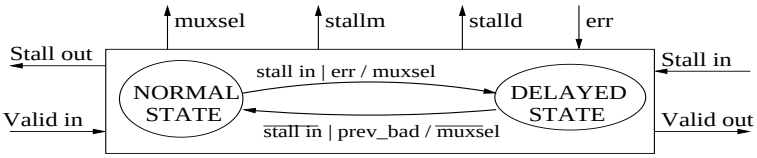

Figure 4.12: Control circuit FSM for scheme 3

Figure 4.12 shows the control logic FSM for this scheme. Similar to scheme 1, the control circuit has two states - normal state and delayed state. In normal state, data is captured by the *main flip-flop*. In delayed state, data is captured by the *delayed flip-flop* and the previous stage is stalled. Since a stall request is generated for each timing error, the error penalty increases with the number of errors.

4.2.4 Towards Modular NoC design

In summary, we observe that the three schemes provide different levels of reliability and error tolerance. Table 4.1 shows comparison between the three schemes.

Table 4.1: Comparison between three schemes

Criterion	Scheme 1	Scheme 2	Scheme 3
Error tolerance	Only detects	Detect and Correct	Detect and Correct
Error Penalty	Depends on correction scheme	Depends on number of link buffers	Increases with error rate
Control logic	Simplest	Complex	Moderate
Area [1]	1	1.72	1.24
Control signals	3	5	4

The sender and receiver interfaces remain same for all the three schemes. In case of buffered links between switches, the switch design becomes less complex, modular, consumes less area and facilitates layout of NoC. We do not need switches having same ports but different requirements for storage (input or output) buffers, since the storage is done on the link. The link is used as a *storage* medium as well as *communicating*

[1]Normalized to scheme 1. Quantitative results are reported in section 5.4

medium. Depending upon the application needs and NoC characteristics different link designs can be chosen and interchanged in a *plug and play* fashion. NoC design and layout becomes more regular, structured and can be easily optimized.

Chapter 5

Simulation Results and Experimental Data

This chapter explores the latency benefits and characteristics of Timing -Error tolerant link design. The simulation platform consists of cycle-accurate SystemC models of the *robust link design schemes* (as described in previous chapter). The number of pipeline stages on each link, which is application and topology dependent, and the choice of the link design scheme are taken as input parameters for the architecture. The choice of the link design scheme depends on the application characteristics and the error-rate of the system. Functional System C simulations were carried out on a variety of application benchmarks using ×`pipes` NoC architecture. Following benchmarks were used for simulations :-

- 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)

Using the ×`pipes` architecture we instantiated the following network topologies to evaluate the relative performance of the three schemes.

1. Two processors, 2×2 mesh (T1)
2. Twenty-four processors, one 11×11 switch and eight 8×8 switches (T2)
3. Four processors, one 11×11 crossbar-like switch (T3)

4. Four processors, three 5 clusters (T4)

5. Eight processors, one 19×19 crossbar-like switch (T5)

6. Eight processors, 3×4 mesh with 6×6 nodes (T6)

Functional error models based on data dependent errors were developed. The wire delay for data transmission on a link depends on the switching activities in the data patterns transmitted. As presented in [23], the wire delay for adversarial switching patterns (such as the 3-bit transitions from 101 to 010) varies by 50% of normal wire delay. To introduce such data dependent timing errors, the link is monitored and checked for various adversarial switching patterns to inject timing errors. Variations in delay due to ground bounce and PVT (Process, Temperature, Voltage) variations are also accounted by using the probability of such error occurrences, as obtained from [26], to introduce timing errors.

5.1 Effect of Aggressive Design

We compare aggressive and conservative design approaches and show that in spite of large error rates, aggressive clocking results in smaller execution time as opposed to conservative clocking mechanism with no errors. Aggressive link design can be achieved in two ways: by either using a higher clock rate for the aggressive design as compared to the conservative design, or by increasing the physical spacing between adjacent link pipeline stage flip-flops, thereby decreasing the number of pipeline stages on a link. As the aggressive design methodology targets *safe* operation only under normal operating conditions, occasionally timing errors may occur in both these schemes. By incorporating the Terror methodology on the aggressive designs, the impact of the timing errors can be reduced and achieve large latency savings compared to conventional design methodologies.

To see the effects of the aggressive design methodology, two sets of experiments are performed: in the first experiment, we over-clock the system, keeping the spacing between adjacent link pipeline stage flip-flops same as that of a conservative design, and in the second experiment, we increase the spacing between the adjacent link pipeline stage flip-flops, keeping the same clock rate. We use the timing error-tolerant methodology, for correcting the timing errors that occur in the aggressive design.

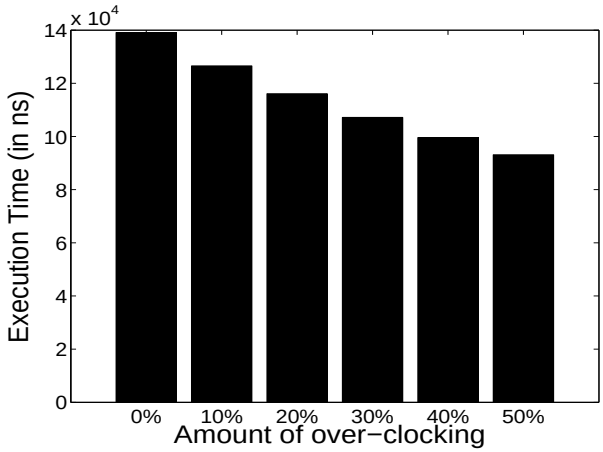

Figure 5.1: Amount of over-clocking

The total execution time (in ns) for the Mat2 benchmark suite for a set of aggressively designed links using Scheme 2 , with over-clocking is shown in Figure 5.1. The execution times plotted (the y-axis), includes the time taken for both computation and communication of data in the system. The base clock frequency of the conservative system is assumed to be 1 Ghz. The x-axis represents the percentage by which we over-clock the system. As the clock frequency increases, the amount of errors that occur at each link starts to increase. As an example, the flit-error rate (the probability that a flit arriving at a link pipeline stage has a timing error) for this benchmark example is 6% when the system is over-clocked by 10% and is around 29%, when the system is over-clocked by 50%. As the system is over-clocked, ideally, the execution time for an application should decrease. However, as the system is over-clocked, the error-rate starts to increase and the penalty for error detection and correction increases. Thus the execution time for the system is dependent on both these effects: of reduced clock delay and increased error detection/correction penalty. As the penalty for error detection and correction for Scheme 2 is very small, there is a large reduction in application execution time for the aggressive design. As seen from Figure 5.1, large performance improvement is achieved by over-clocking the timing-error tolerant system.

Figure 5.2 shows the difference in execution times of the various aggressive designs obtained by increasing the spacing between adjacent flip-flops. As the design becomes more aggressive, the performance of the system increases. Note that in all these plots, the delay of the *secondary flip-flops* stop at 50%, as SPICE simulations (described in Chapter 2) , showed that the maximum delay between the *main* and *delayed flip-flops*

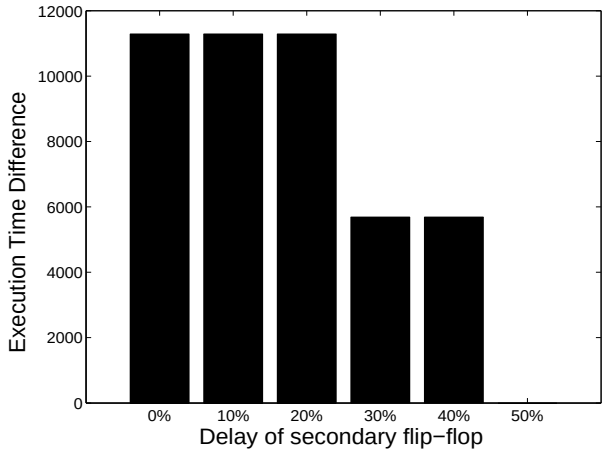

Figure 5.2: Percentage increase in spacing

cannot be increased beyond 50% due to timing overheads of the circuit.

5.2 Effect of Delayed Clock

In this sub-section we explore the performance of Scheme 1 for different congestion levels in the network. In Scheme 1, the timing errors are avoided when the data is captured by the *secondary flip-flop* of the 2-entry FIFO, since it is clocked by delayed clock *ckd*. Figure 5.3 shows the normalized execution times as a function of the percentage of time the *secondary flip-flop* is used, for the FFT benchmark suite.

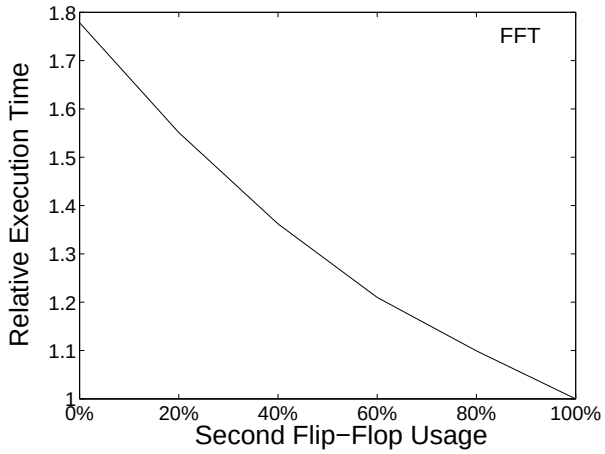

Figure 5.3: Second flip-flop Usage

As the amount of time the *secondary flip-flop* is used increases, the error rate

decreases. For this example, the error-rates vary from 5% (when the *secondary flip-flop* is not used at all) to nearly 0% (when the *secondary flip-flop* is used for almost 100% of the time). The execution times (the y-axis in the plot) are normalized to the ideal application execution time (i.e. when there are no errors). We simulated errors by using error models that were based on [23], [26].

When the network has low congestion, the percentage of time the *secondary flip-flops* are used is small, as the *secondary flip-flops* are used only when there is some congestion, which causes back-pressure to happen in the network. In this case, when errors happen, the retransmission of data doesn't have much impact on the congestion (as the congestion itself is small). For a heavily congested network, most of the data passes through the *secondary flip-flops* and thus, most of the timing errors are avoided. Thus Scheme 1 (using delayed clock for the second flip-flop) is a good choice, when the retransmission penalty for zero-load latency is acceptable for the application (which depends on the error rate in the system and the application characteristics). (Note that the 100% usage point, plotted in Figure 5.3, will never occur in reality as initially flits have to pass through the first flip-flop and only when some congestion starts to happen, the *secondary flip-flops* are used).

5.3 Retransmission Vs Terror Schemes

In this sub-section, we compare the performance of systems that use traditional retransmission mechanism (we assume switch-to-switch retransmission) when timing errors are detected, with systems that use Scheme 2 presented in this paper. In Figure 5.4, we plot the total execution time for the various benchmark applications for these two schemes, for 5% error rate.

For all the benchmarks the total execution cycles for the retransmission scheme is much higher than the proposed scheme (up to 2x for FFT). As the error rate increases, the penalty incurred by the retransmission scheme increases exponentially (refer Figure 5.5 for results on the FFT benchmark), while there is very little change in penalty for the T-error scheme. The reason for the almost constant penalty of this scheme (Scheme 2) is that, once a first timing error occurs at a pipeline stage of a link (which incurs a single cycle penalty for correction), and as long as there is a continuous data flow on the link, further timing errors are avoided at this pipeline stage (refer Section 4.2).

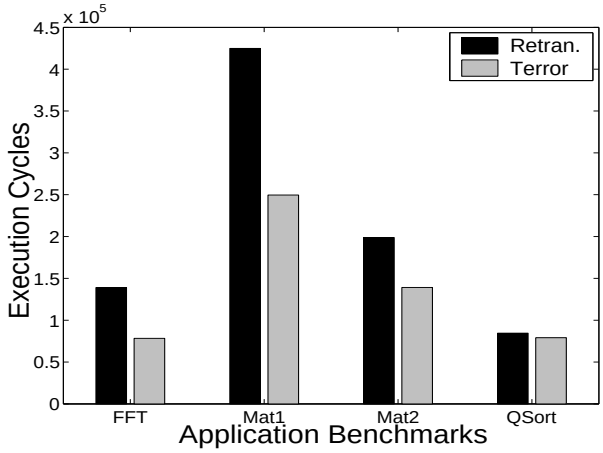

Figure 5.4: Application effects

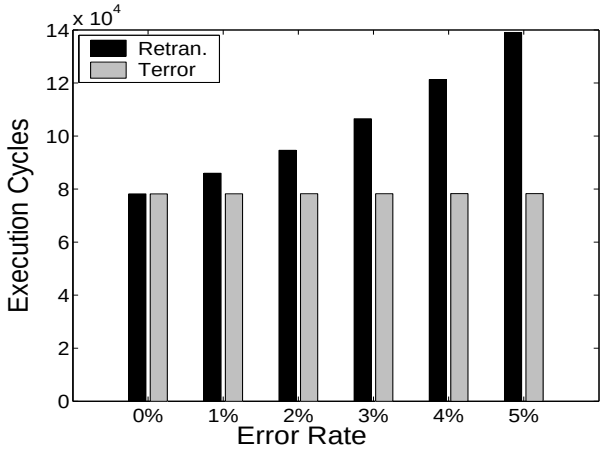

Figure 5.5: Error rate effects

Figure 5.6 shows the effect of retransmission and T-error schemes for different NoC topologies for the MAT2 benchmark. The topologies compared vary from small 7 core NoCs to 51 core NoCs. For all the topologies, Scheme 2 gives substantial performance improvement over the retransmission scheme.

5.4 Choice of Link Design Schemes

In this sub-section we present experimental data that shows the trade-offs between the three schemes for different network conditions and traffic flow patterns. In Figure 5.7 we plot the relative execution times of the Mat2 benchmark for Scheme 1 (use of delayed clock) and Scheme 3 for different amount of congestion in the network, for 4% error rate. The plots are normalized with respect to the execution time of the

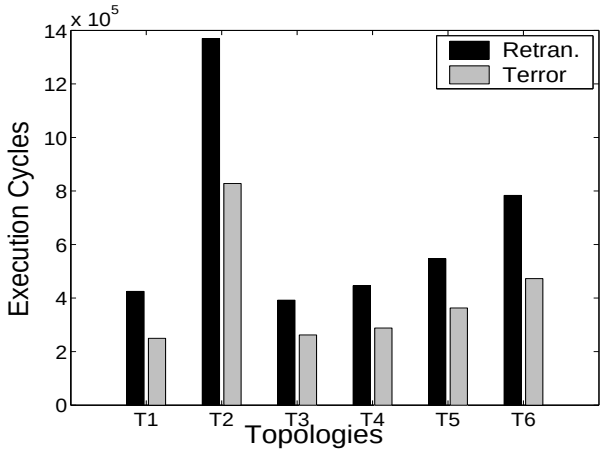

Figure 5.6: Topology effects

ideal situation, where timing errors don't occur.

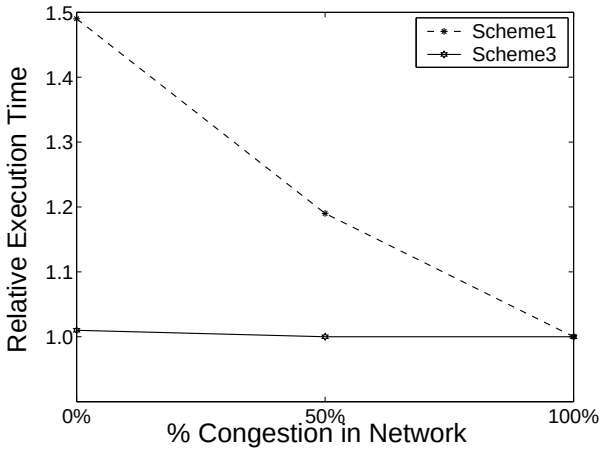

Figure 5.7: Schemes 1 and 3

We refer to the percentage congestion as the amount of time the *delayed flip-flop* is used in the system (as the delayed flip-flop is used more in a more congested network). As the congestion in the network starts to increase, the amount of penalty incurred by Scheme 1 starts to decrease, an effect explored in detail in Section 5.3. The penalty of the T-error scheme also decreases slightly with increase in congestion. Depending on the amount of congestion in the network, application characteristics and error rate, the choice between the two schemes can be made. For networks that have high congestion or low error rates, Scheme 1 provides almost similar performance as Schemes 2 and 3.

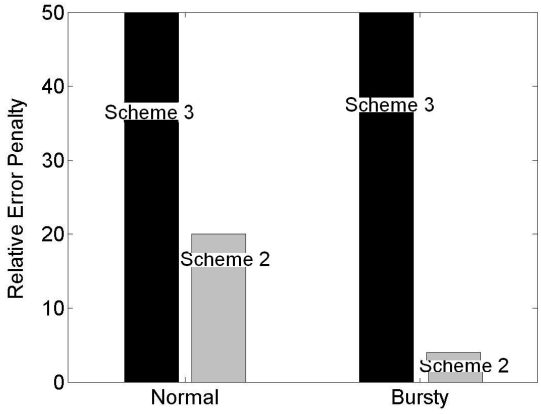

Figure 5.8: Schemes 2 and 3

In Figure 5.8 we plot the relative latency penalty for Schemes 2 and 3 for transmitting 1000 flits of data for two classes of applications: applications with normal data traffic (where the cores send data at random times that follow a uniform distribution) and applications with bursty data traffic. As discussed earlier, for continuous flow of data traffic on the link, Scheme 2 incurs a single cycle penalty at a link pipeline stage for only the first timing error that occurs in the continuous stream (as further timing errors are avoided). Thus Scheme 2 has low penalty for the bursty data traffic when compared to the normal traffic flow. As Scheme 3 incurs a single cycle penalty for each timing error, it has higher penalty for error correction than Scheme 2.

We synthesized (using Synopsys DC compiler) the robust links to get area estimates of the proposed schemes. Table 5.1 shows the total link area for the different schemes (normalized to the area of Scheme 1) for 32 and 64-bit flit-sizes (the width of the links) for a 5×5 mesh with each each link having 4 pipeline stages.

Table 5.1: Relative Link Area

	32	64
Scheme 1	1	1
Scheme 2	1.91	1.72
Scheme 3	1.3	1.24

5.5 Summary of Results

In this chapter, we have presented experimental data and simulation plots that enumerate the characteristics and relative performance of the three timing-error tolerant

schemes. Compared to traditional error correcting schemes, the proposed designs give substantial improvements in latency, even at high error rates upto 5%. Our results present quantitative differences in area, receiver latency and overall performance between the three schemes for different topologies, applications and network conditions. We observe that the effective error penalty at the receiver not only depends on the chosen scheme but also depends upon the network flow (congested or smooth) and traffic conditions (bursty or continuous). Given some performance specifications and NoC architecture, the simulation results can be used to make an appropriate choice between three robust link design methodologies, for a reliable communication network.

Chapter 6

Conclusion

With shrinking feature sizes and faster clock cycles, wire delay as a percentage of total delay is increasing. A communication centric design approach - *Network on Chip* is used to address the problems of latency, throughput and scalability of interconnection network between cores. Wires are becoming more susceptible to various noise sources such as cross-talk, coupling noise, soft errors, process variations, inductive loops etc. The delay variations due to such interferences consume an increasing percentage of the useful clock cycle. Conservative design approaches that consider worst case conditions will lead to inefficiency and poor performance since they account for worst case noise-induced delay variability that may not occur frequently. Aggressive design approaches are required where designs are built for normal (ignoring data-dependent delay variations) operating conditions and they provide minimal penalty in presence of noise-induced timing errors. This work presents a Timing-Error Tolerant Design methodology (*Terror*) to design NoC links. It is used to design links aggressively such that the communication system is tolerant to timing errors and has minimal effect on the performance. *Terror* system focusses on correcting transient timing errors such as those caused by cross-talk and does not address static errors caused by stuck-at faults, permanent failures etc. In a Terror enabled system, latency does not increase linearly with error rate but is bounded by the number of buffers on the link. This makes it more suitable for high-error rate conditions where traditional error correcting systems would degrade. As compared to retransmission scheme for correcting errors on links, a Terror enabled scheme gives up to 35% savings in latency.

Based on the Terror design, three different *robust link design* methodologies were explored which had a trade-off between hardware complexity, area and performance

(latency, throughput). Buffer storage requirements on a typical NoC switch were almost avoided by using the link as a *storage* medium to store data. Depending upon the application characteristics and communication patterns, the schemes can be selectively used and interchanged in a *plug and play* fashion. Comparisons and experiments made on various benchmark suites and network topologies showed a substantial improvement (as large as 1.5×) over conservative system.

Bibliography

[1] Pierre Guerrier , Alain Greiner, "A generic architecture for on-chip packet-switched interconnections", Design, Automation and Test in Europe Conference and Exhibition 2000. Proceedings , 2000, pp. 250-256.

[2] Andrei Radulescu and Kees Goossens. In Shuvra Bhattacharyya and Ed Deprettere and Juergen Teich, editors, "Communication Se rvices for Networks on Silicon", Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation. Marcel Dekk er, 2003.

[3] L.Benini and G.De Micheli, "Networks on Chips: A New SoC Paradigm", IEEE Computers, pp. 70-78, Jan. 2002.

[4] R. Ho, K.Mai, M. Horowitz, "The Future of Wires", Proceedings of the IEEE, pp. 490-504, April 2001.

[5] K.Aingaran et al., "Coupling Noise Analysis for VLSI and ULSI Circuits", IEEE ISQED 2000, pages 485- 489, March 2000.

[6] K. L. Shepard and V. Narayanan. "Noise in Deep Submicron Digital Design". IEEE/ACM ICCAD-96, pages 524- 531, November 1996.

[7] N. R. Shanbhag, "Reliable and efficient system-on-chip design", IEEE Computer, Vol. 3, Issue: 3, pp. 42-50, March 2004.

[8] L.P.Carloni, K.L.McMillan, A.L.Sangiovanni Vincentelli, "Theory of Latency-Insensitive Design", IEEE Trans. on CAD of ICs and Systems, Vol.20, No.9, pp. 1059-1076, Sep 2001.

[9] Y.Zhao, L.Chen and S.Dey, "Online Testing of Multi-source Noise-induced Errors in the Interconnects and Buese of System-on-Chips", IEEE proceeding of International Test Conference 2002, 10, October, 2002.

[10] J.Warland, P.Varaiya, High-Performance Communication Networks, Morgan Kauffmann Publishers Inc, 1996.

[11] A.K.Uht, "Going Beyond Worst-Case Specs with TEAtime", IEEE Computer, pp. 51-56, March 2004.

[12] D.Ernst e al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation", Micro Conference, Dec 2003.

[13] M. Favalli, C.Metra, "Low-level error recovery mechanism for self-checking sequential circuit", DFT 97, pp. 234-242, oct. 1997.

[14] M.Singh, S. M.Nowick, "MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines", Proc. ICCD 01, Sept. 2001.

[15] E.Dupont,, M.Nicolaidis, P.Rohr, "Embedded Robustness IPs for Transient-Error-Free ICs", IEEE Design & Test, vol. 19 ,Issue 3,pp. 56-70, May 2002.

[16] F.Klass, "Semi-dynamic and dynamic flip-flops with embedded logic", VLSI Circuits 98, pp. 108-109, June 1998.

[17] S.Matsushita, "Design experience of a chip multiprocessor merlot and expectation to functional verification", ISSS 2002, pp. 103-108, Oct 2002.

[18] B. Ackland et al., "A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP", IEEE Journal of Solid-State Circuits, vol. 35, Issue 3, pp. 412-424, Mar 2000.

[19] M.B.Taylor et al., "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams", ISCA 2004.

[20] A. Jain et al., "A 1.2GHz Alpha Microprocessor with 44.8GB/s. Chip Pin Bandwidth", ISSCC Digest of Technical Papers, pp.240-241, Feb. 2001.

[21] Srinivasa R. Sridhara, Naresh R. Shanbhag.," Coding for system-on-chip networks: a unified framework", Design Automation Conference, 103-106, June, 2004.

[22] Paul Peter P. Sotiriadis., "Interconnect Modeling and Optimizationin Deep Sub-Micron Technologies", Ph.D dissertation, Massachusetts Institute of Technology, 2002

[23] Yungseon Eo, Seongkyun Shin, William R. Eisenstadt, and Jongin Shim, "A decoupling technique for efficient timing analysis of VLSI interconnects with dynamic circuit switching", IEEE Transactions on Computer-aided design of Integrated Circuits and Systems, September, 2002.

[24] Patel, K.N. Markov, I.L, "Error-Correction and Crosstalk Avoidance in DSM Busses", Oct, 2004, 1076-1080, vol 12, issue: 10

[25] Kei Hirose, Hiroto Yasuura, "A Bus Delay Reduction Technique Considering Crosstalk", Design, Automation and Test in Europe, 2000.

[26] Wang, D.T.; Mcnall, "A statistical model based ASIC skew selection method", 2004 IEEE Workshop on Microelectronics and Electron Devices, 64-66.

[27] Paul Wielage and Kees Goossens, "Networks on Silicon: Blessing or Nightmare ", KeyNote speech, Euromicro Symposium On Digital System Design (DSD 2002), September, 2002

[28] W.J.Dally, B.Towles, "Route Packets, not Wires: On-Chip Interconnection Networks", Design and Automation Conference DAC 2001, pp. 684-689, Jun 2001.

[29] VSI Alliance, "http://www.vsi.org/"

[30] Open Core Protocol, "http://www.ocpip.org/"

[31] S.J.Krolikoski, et. al, "Methodology and Technology for Virtual Component Driven Hardware/Software Co-Design on the System-Level", ISCAS 99, pp. 456-459, June 1999.

[32] Srinivasan Murali, Giovanni De Micheli, "SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs", DAC 2004.

[33] Antoine Jalabert, Srinivasan Murali, Luca Benini, Giovanni De Micheli, "xpipesCompiler: A Tool for instantiating application specific Networks on Chip", Proc, DATE 2004.

[34] H.S Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks", MICRO, Nov. 2002.

[35] Chen, L.H.; Marek-Sadowska, M.; Brewer, F, "Coping with buffer delay change due to power and ground noise", DAC, 2002, pp.860 - 865.

[36] Marculescu, R, "Networks-on-chip: the quest for on-chip fault-tolerant communication", Proc. IEEE Computer Society Annual Symposium on VLSI, pp. 8-12, 2003.

[37] D. Bertozzi et al., "Low Power Error-Resilient Encoding for On-Chip Data Buses", DATE 2002, pp. 102-109.

[38] F. Worm et al., "An Adaptive Low-power Transmission Scheme for On-chip Networks" , ISSSA ISSS, October 2002, pp. 92-100

[39] M. Pirretti et al.,"Fault Tolerant Algorithms for Network-On-Chip Interconnect", Proc. of ISVLSI, Feb 2004.

[40] L. Li et al., "A Crosstalk Aware Interconnect with Variable Cycle Transmission", DATE'04. February 2004

[41] P. Vellanki et al., "Quality-of-Service and Error Control Techniques for Network on Chip Architectures", GLSVLSI 04, Apr 2004 .

[42] H. Zimmer et al.,"A Fault Model Notation and Error-Control Scheme for switch-to-Switch Buses in a Network-on-Chip", ISSS/CODES, 2003.

[43] Mizuno et al., "Elastic interconnects: repeater-inserted long wiring capable of compressing and decompressing data", SSC Conf., pp. 346-347, Feb 2001.

[44] V. Chandra et al., "A Power aware system level interconnect design methodology for latency insensitive systems", ICCAD 2004, pp. 275-282, Nov 2004.

[45] Mizuno et al., "Elastic interconnects: repeater-inserted long wiring capable of compressing and decompressing data", SSC Conf., pp. 346-347, Feb 2001.

[46] W. J. Dally et al., "Principles and Practices of Interconnection Networks", Morgan Kaufmann, Dec, 2003.