

Quantitative Cost Modeling of Error Protection for Network-on-Chip

M. C. Neuenhahn, D. Lemmer, H. Blume, T. G. Noll

Abstract— The importance of Systems-on-Chip (SoC) is increasing more and more. These SoCs will consist of many functional units like e.g. embedded processor cores, embedded memories or FPGA-like structures. The huge communication demands of these SoCs can hardly be satisfied by bus-based architectures, as they do not provide enough flexibility and performance. New promising approaches to fulfill these requirements are so called Network-on-Chip (NoC) architectures. In the deep sub- μ regime, on-chip interconnects are becoming more and more sensitive to noise caused by e.g. power supply, crosstalk or radiation induced effects. These noise effects are likely to reduce the reliability of data.

In this paper different error protection techniques for NoCs are presented. They are integrated into a NoC design framework, which targets FPGA-based implementations as well as VLSI-implementations. Using this generic and modular framework NoCs can be easily generated allowing to modify NoC-specific parameters, like e.g. word length, error protection technique or routing algorithm.

Error protection itself consists of error detection, correction (e.g. Parity, or Hamming) and handling (e.g. retransmission or retransmission with sliding-window enhancement). NoCs featuring these different error protection techniques have been implemented on an FPGA as well as using a 90 nm standard-cell design flow. The corresponding costs in terms of silicon area were derived varying those NoC-specific parameters which influence the error protection techniques (e.g. error protection parameter or word length). Arithmetic functions describing these costs are provided. These functions can be used to estimate implementation costs in an early design-stage of NoC-development and are therefore a valuable means to decide for the suitable error protection technique. The performance of the implementations was evaluated using a cycle accurate FPGA-based NoC-emulator, which allows to evaluate the performance of NoCs for various traffic and error patterns.

Index Terms— System-on-Chip, Network-on-Chip, FPGA, Error Protection

Manuscript received on October 1, 2007

M. C. Neuenhahn is with the Chair of Electrical Engineering and Computer Sciences, RWTH Aachen University, Germany (e-mail: neuenh@eecs.rwth-aachen.de)

D. Lemmer was with the Chair of Electrical Engineering and Computer Sciences, RWTH Aachen University, Germany.

H. Blume is with the Chair of Electrical Engineering and Computer Sciences, RWTH Aachen University, Germany (e-mail: blume@eecs.rwth-aachen.de)

T. G. Noll is head of the Chair of Electrical Engineering and Computer Sciences, RWTH Aachen University, Germany (e-mail: tgn@eecs.rwth-aachen.de)

I. INTRODUCTION

The increasing complexity of state of the art VLSI-Chips, due to technology improvements, will lead to System-on-Chip (SoC) [1] with up to hundred functional units on a single chip [2]. These functional units can be part of a homogeneous processor array or heterogeneous array consisting of different units, for example processor cores, hardware macros, embedded FPGA-like structures or memories [3].

The communication demands of applications mapped onto these complex SoCs will be very high. For example, high throughput shall be provided at minimal latency, with guaranteed quality of service (QoS) and as much flexibility as possible. Furthermore, the communication should cause minimal costs, in terms of area and power consumption. As the number of functional units in SoCs will increase in the future the communication-structure should although be scalable. Traditional communication structures like on-Chip busses or point-to-point connections cannot fulfill these demands. Often mentioned solutions to this problem are Network-on-Chip (NoC) architectures [4].

These NoCs feature a variety of NoC-specific parameters, like e.g. word length, topology, routing-algorithm, switching technique or error handling which influence performance and costs caused by the NoC, like area or power dissipation. In order to find an optimal set of NoC-parameters which fulfills the communication demands of a given application at minimal costs, an efficient design-flow for NoCs is presented in chapter II.

While scaling technology to the deep sub- μ regime, interconnects are more and more susceptible to errors [5]. These errors can have multiple sources, like e.g. crosstalk [6], power-supply noise, leakage noise, process variations or soft errors.

In literature, different techniques to avoid or correct errors on interconnect are presented [7], for example reducing the crosstalk-energy [8], [9] or using codes to detect and correct errors [10]. Most of these solutions do not support error handling or do not quantitatively analyze the implementation costs in terms of area and/or performance penalties.

The generic and parametrical NoC-architecture, which is used to implement the error-protection technique, is introduced in chapter III. Implementation details of the error protection techniques, which are separated into error detection and correction and error handling, are given in chapter IV. The

implementation results are presented in terms of used area for a VLSI-based implementation. Finally a performance evaluation is presented in chapter VI.

II. NOC DESIGN-FLOW

In order to build NoCs that fulfill given communication demands at minimal costs in a very short development-time, the design-flow shown in Figure 1 was evolved.

System-Analysis: In the first stage, a given application is analyzed and separated into several tasks. These tasks are then mapped to hardware-architectures like embedded DSPs, processor cores etc. [11]. Based on this mapping communication demands of this application can be described as a so called task graph [12].

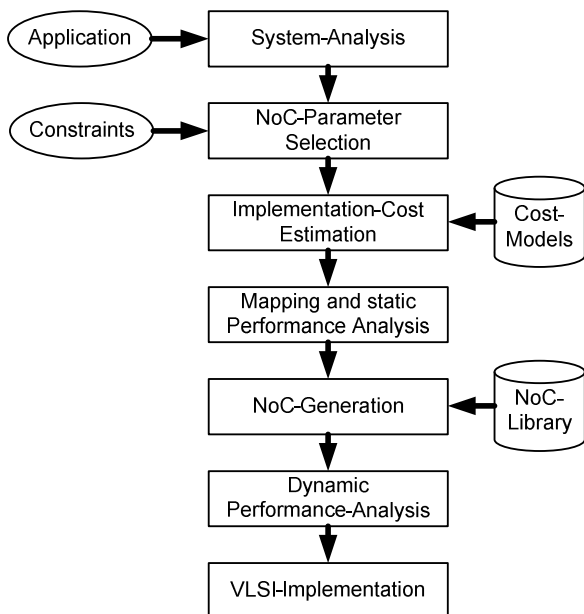


Figure 1: NoC Design-Flow

NoC-Parameter Selection: Derived from these communication demands the user chooses NoC-specific parameters for example topology, data word length or routing algorithm which describe a specific NoC which likely fulfills these demands.

Implementation Cost Estimation: The costs caused by this NoC are estimated in the next step, using mathematic functions modeling the implementation costs. These functions describe costs like area, power dissipation or also static timing measures. If these costs meet the given constraints, the functional units of the system are mapped to the network-interfaces of the NoC. Otherwise, other NoC-parameters have to be chosen.

Mapping and static Performance Analysis: The mapping can be performed by hand or by using an automated, cost aware mapping algorithm, as proposed in [13]. During the mapping process a static performance analysis is performed, providing information about potential bottlenecks or unused NoC resources.

NoC-Generation: If costs and performance meet given

specifications, the NoC is automatically generated. The outputs of this generator are a VHDL-description for simulation, standard-cell synthesis, and FPGA-based emulation. Additionally a SystemC- and Colored-Petri-Net-description is provided for dynamic performance analysis at different levels of abstraction [14].

Dynamic Performance Analysis: The dynamic performance analysis, based on simulation or emulation, reports NoC-resources, which might become a bottleneck and performance figures like average or maximal latency. If the task-graph features deadlines, the number of missed deadlines can be reported. This dynamic analysis also provides information about blocked resources and communication overhead in contrast to the static analysis.

To achieve significant results, this analysis should be performed with real world data amounts. Using a Simulator (VHDL-simulator or even a SystemC based simulator) this performance analysis requires unacceptable long simulation times. Therefore, an FPGA-based emulator [15] was used which has a much higher emulation speed.

VLSI-Implementation: Finally, the NoC based on the generated VHDL-code is implemented using standard-cells. If the synthesized design does not match the given constraints, modifications at different steps of the design-flow have to be performed.

In this work we present implementations of different error-protection techniques for NoC. Using these implementations, cost models for NoC-parameters are derived and the performance of these error-protection techniques is evaluated.

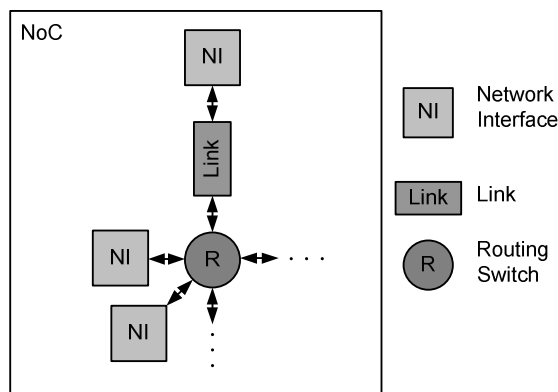


Figure 2: Network-on-Chip

III. MODULAR AND GENERIC NOC

The modular and generic NoC framework consists of three basic building blocks: Network Interface (NI), Routing Switch (R) and Link. The Network Interface is the gateway between NoC and a functional unit, the Routing Switch routes data through the NoC and the link represents the connections between these building blocks (see Figure 2). The arrangement of these blocks is defined by the NoC-parameter topology.

Error-protection is implemented basically in the Network Interface, so this building block is explained in detail. It consists of two independent parts: one for sending data over

the NoC (NI-Send, see Figure 3) the other one used for receiving data from the NoC (NI-Receive, see Figure 3).

The sending part of the Network-Interface consists of a block *Control*, which realizes the switching technique, e.g. circuit or packet switching. The modified data stream is transferred to the block *ECC/EDC*, where error correcting or detecting codes are applied to the data. In case that sending data over the network failed, because a needed NoC-resource is already used, the block *Connection-Failure Handling* initiates a new attempt to send this data.

If an error occurred during the data transmission (e.g. one data word is corrupted), this is reported to the block *Error Handling*. This block starts a retransmission of this data, which is stored in the block *Send-Buffer*.

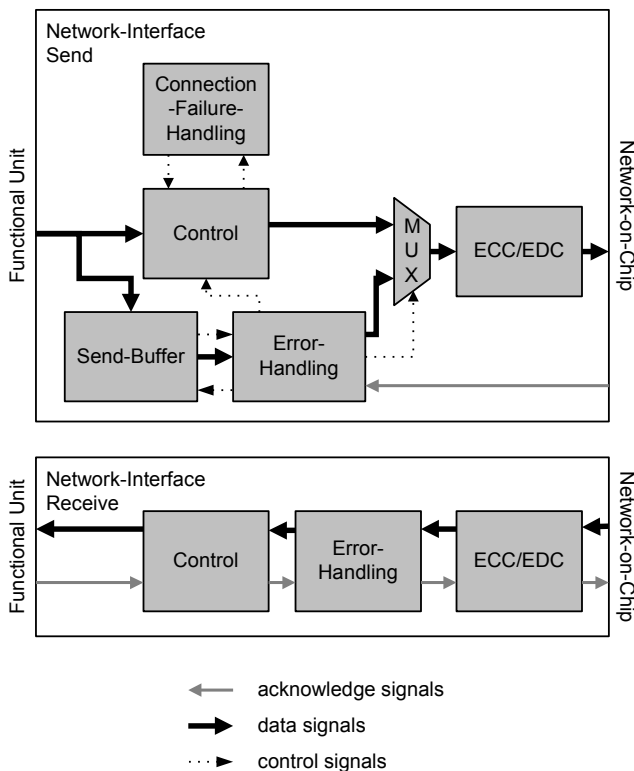


Figure 3: Network Interface

In the receiving part of the NI the block *ECC/EDC* is used to detect potential errors in the received data words, if the appropriate error protection technique is used. A detected error is reported to the block *Error-Handling*, where for example a retransmission of the erroneous data can be requested. The block *Control* is used to implement the switching technique.

IV. ERROR PROTECTION

Error protection, as it is implemented in this work, is divided into two parts: error detection and correction and error handling. The first part is used to detect errors, correct them if possible and report non correctable errors to the error handling part. The error handling has to cope with these errors, for example by retransmitting the corrupted data word.

A. Error Detection and Error Correction

Error detection and correction is usually performed on data word or data block level. The ability to detect and correct errors is based on redundancy, added to the data words. In this work we focused on error detection and correction on word level.

a) Parity Code

The parity code is a simple code which detects a single error (SED). In the sending part of the NI the parity of a data word is calculated. The code word, which is going to be transmitted over the NoC, is built up by the data word and the parity bit. The receiver also calculates the parity of the received data word and compares it with the received parity bit. If the calculated and the received parity value do not match an error has occurred. The parity bit calculation is done using a XOR-tree.

b) Hamming Codes

Hamming codes are widely used codes for error protection in digital communication and storage systems. The basic Hamming codes are linear distance-3 codes. Using these codes single error correction (SEC) or double error (DEC) detection is possible [16]. Using enhanced Hamming codes (a distance-4 code) [16] simultaneous single error correction and double error detection (SEC/DEC) or triple error detection (TED) is possible.

The Hamming encoder adds several parity bits to the data word. The amount of parity bits, which has to be added to the data word, depends on the word length of the data word. Each parity bit is calculated from specified bits of the data word. The number of parity bits used for Hamming codes is given in Table 1.

The Hamming decoder features an error detecting stage and an optional error correcting stage. The detecting stage is built up similar to the encoder and compares the calculated parity bits to the received parity bits. Based on this result a correction word can be calculated in the correction stage and can be used to correct the received codeword.

TABLE 1: PARITY BITS USED FOR DIFFERENT ERROR DETECTION CODES

Data word length	Number of parity bits		
	Parity code	Hamming code	Enhanced Hamming code
5 .. 11	1	5	6
11 .. 26	1	6	7
27 .. 57	1	7	8
58 .. 120	1	8	9
121 .. 246	1	9	10

As the Hamming code can be used as an error detecting or a error correcting code, the basic Hamming code can be used to detect and correct single bit errors (SEC) or to detect double bit errors (DED). The enhanced Hamming code can correct single bit errors and detect double bit errors (SEC/DED) and

detect triple bit errors. An overview of the implemented error correcting codes is given in Table 2.

B. Error Handling

Non correctable errors, which are detected, have to be handled by the *Error-Handling* block. Possible options are accepting these errors (send and forget, S&F) or requesting a retransmission of the data.

For retransmission of corrupted data two protocols were implemented and evaluated. The first one is called send-and-wait (S&W) protocol. The sending NI waits after a data word was dispatched until the correct reception is reported (*ackn*). Then the next data word is going to be sent (see Figure 4). If this acknowledgment does not arrive in a defined time period (*time to acknowledge*), the data word is going to be sent again. The time to acknowledge has to be chosen very carefully because it has a direct impact on the performance when data has to be retransmitted.

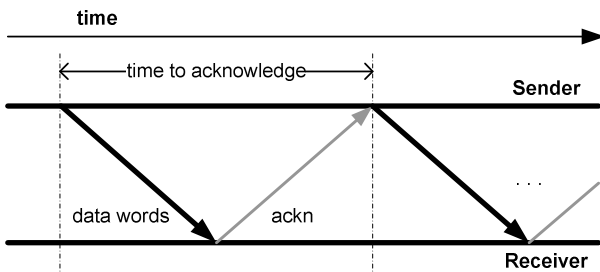


Figure 4: Error handling, using send-and-wait protocol

As this protocol results in a reduced throughput rate by waiting for an acknowledgment the second protocol, which is called sand-and-wait with sliding window enhancement (S&W SWE), solves this problem. Using this protocol data words are sent continuously because the acknowledge signal can be received later. If a non-correctable error is detected, the corrupted data word and all data words following this data word have to be retransmitted. Therefore, data words which have not been acknowledged have to be stored in the sending NI.

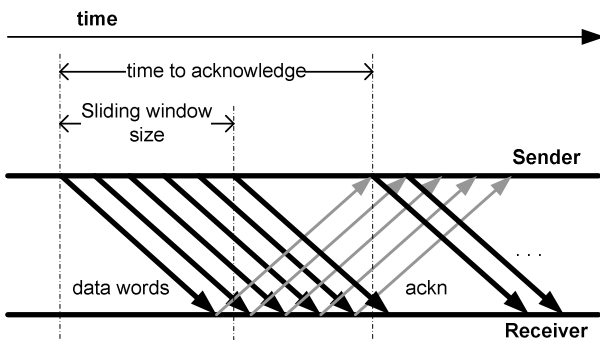


Figure 5: Error handling, using sand-and-wait protocol with sliding window enhancement

The number of data words which can be sent over the NoC without receiving an acknowledge signal is defined by the parameter *sliding window size*. This parameter also defines the

size of the *Send-Buffer* where the not acknowledged data words are stored. If the time needed to send the data words of a sliding window is smaller or equal to the time needed to receive an acknowledge signal (*time to acknowledge*) there is no performance penalty, if no non-correctable error occurs.

V. IMPLEMENTATION RESULTS

The NoC-parameter world length, error detection and correction and error handling influence the implementation costs for error-protection in NoCs. In this chapter, the implementation costs for NoCs targeting a VLSI-implementation are analyzed. Other parameters like e.g. topology or routing-algorithm have also an influence on the implementation costs but are not regarded here.

The area needed for a VLSI-implementation of a NI was obtained by the following design flow. At first the VHDL-code describing the NI was synthesized using Synopsys Design Compiler [17]. The resulting netlist was used in the place-and-route tool Encounter [18]. For verification and power evaluation the resulting layout is extracted into a SPICE-netlist [19] which is simulated using NanoSim [20]. In this work a 90 nm technology from TSMC and a standard-cell library from Artisan Components [21] is used. The links, connected to each NI are assumed to have a length of 1 mm. The resulting capacity on each output port is calculated to 0.148 pF (5 x minimum distance to neighbors, toggling against each other) and the operating frequency is 1 GHz. All results are computed using slow operating conditions.

TABLE 2: IMPLEMENTED ERROR DETECTION AND CORRECTION TECHNIQUES

Shortcut	Functionality	Code
SED	Single error detection	Parity-Code
SEC	Single error correction	Hamming Code
DED	Double error detection	Hamming Code
SEC/DED	Single error correction and double error detection	Enhanced Hamming Code
TED	Triple error detection	Enhanced Hamming Code

The parameter error detection and correction has no influence on the implementation costs of the error handling. Therefore, the blocks *ECC/EDC* in the NI-send and NI-receive can be analyzed separately.

At first the error-detection and correction techniques shown in Table 2 were implemented for different data word lengths ranging from 16 bit to 255 bit. The resulting area needed for the encoder, which is part of the sending part of the NI, is shown in Figure 6. There is a nearly linear dependency of the area from the data word length. As to be expected, the SED-Code, a parity-code, features the smallest area, whereas the other codes, based on Hamming codes, have nearly the same area requirements in the encoder. The enhanced Hamming codes (TED, DED/SEC) need about the same area as the less complex Hamming codes (DED, SEC).

Similar results are shown in Figure 7, where the area-usage for the decoder, is presented. As for the encoder, the decoder

has a linear dependency from the data word length. The error correction, used in the SED and DED/SEC codes, has a significant impact on the area of the *ECC/EDC* block in the *NI-receive*.

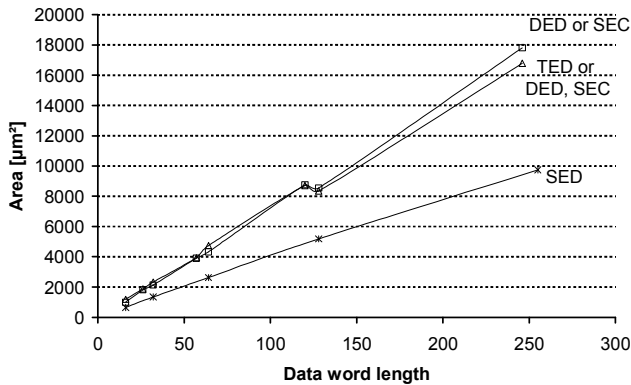


Figure 6: Area of block *Error-Protection* in *NI-Send* with varied data word length

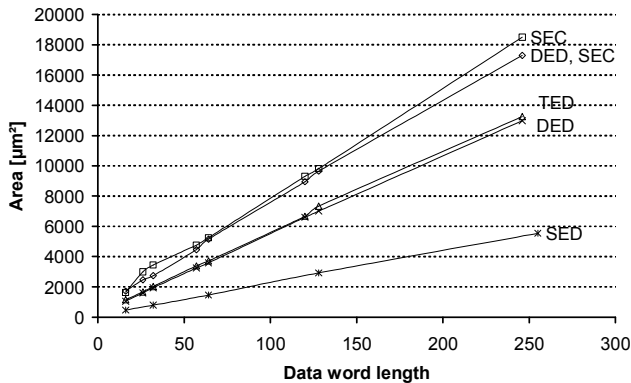


Figure 7: Area of block *Error-Protection* in *NI-Receive* with varied data word length

The implementation costs of error-handling techniques shown in Table 3 have been compared. Exemplarily the area needed for NI using SED, 32-bit data words and a sliding window size of 32 data words is show in Figure 8 (*NI-send*) and Figure 9 (*NI-receive*).

TABLE 3: IMPLEMENTED ERROR-HANDLING TECHNIQUES

Shortcut	Functionality
S&F	Send and forget (no error handling)
S&W	Send and wait
S&W SWE	Send and wait with sliding window enhancement

NI featuring protection techniques have a significant increased the area compared to NI without *Error-Protection*, especially in the sending part of the NI (*NI-send*). Using S&W SWE the requested area is by about a factor of 15 bigger than the area requested for S&F. This is due to the fact, that a memory for storing data words is needed.

For the *NI-receive* module the area is dominated by the error-decoding. The different error-handling techniques have

the same area usage.

The area needed to implement the blocks of the *NI-send* and *NI-receive* depend on the parameters *Error Detection and Correction and Error Handling*. Additionally, other parameters were evaluated in this work are the data word length and the size of the sliding window. In Table 5 cost models describing the used area for individual blocks as mathematical functions are presented. If a parameter has no influence on the area of a certain block, this is marked as don't care "*".

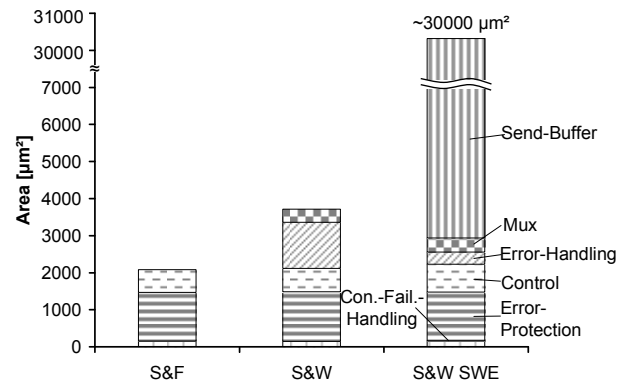


Figure 8: Area distribution of *NI-Send* for different *Error-Handling* techniques, SED, 32-bit data words and a sliding window size of 32 data words

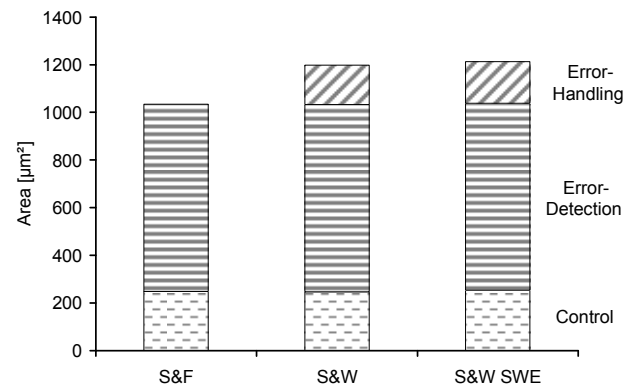


Figure 9: Area distribution of *NI-Receive* for different *Error-Handling* techniques, SED and 32-bit data words

VI. EXPERIMENTAL RESULTS

An FPGA-based NoC-emulator [15] was used to evaluate the performance of the different error-protection techniques. As shown in Figure 10 it is composed of three parts. The first part incorporates the NoC. It is synthesized using the same VHDL-code used for the VLSI-implementation. The second part is built up from traffic-sources and traffic-sinks. Traffic-sources generate data-traffic on the NoC according to requests from the third layer. This data-traffic is transmitted over the NoC to the data-sink, where the data is checked for errors. Additionally, the transmission-time is measured and reported to the third part.

TABLE 4: PARAMETERS USED IN FUNCTIONS FOR MODELING THE SILICON AREA OF NI

Parameter	Symbol	Range
Data word length	n	16 ... 255
Sliding window size	s	16 ... 255

TABLE 5: MODEL FUNCTIONS FOR THE SILICON AREA USED IN NI COMPONENTS

Parameter		Functions [μm^2]
Error Coding	Error-Handling	
NI-Receive Control		
*	*	$3.6 \cdot n + 129$
NI-Receive, Error-Handling		
*	*	$3.8 \cdot n + 62$
NI-Receive, ECC/EDC		
SED	*	$21.3 \cdot n + 122$
SEC DED/SEC	*	$69.8 \cdot n + 745$
DED TED	*	$52.41 \cdot n + 311$
NI-Send, Control		
*	S&W, S&W SWE	$8.8 \cdot n + 329$
*	S&F	$7.5 \cdot n + 391$
NI-Send, Error-Handling		
*	S&W	$23.1 \cdot n + 685$
*	S&W SWE	352
*	S&F	0
NI-Send, Send-Buffer		
*	S&W, S&F	0
*	S&W SWE	$32.8(n + s) + 5046$
NI-Send, Mux		
*	S&F	0
*	S&W, S&W SWE	$12.3 \cdot n + 30$
NI-Send, ECC/EDC		
SED	*	$38.0 \cdot n + 150$
SEC DED	*	$71.4 \cdot n - 116$
SEC/DED TED	*	$66.2 \cdot n - 279$

The third part is implemented using a NIOS II soft-core processor (marked as *Controller* Figure 10) on the FPGA. This processor controls the experiments performed on the emulator and evaluates the results from the traffic sinks. As the processor is programmable in C/C++, different traffic patterns like for example uniform random traffic or application specific traffic patterns can be used.

The single experiments are initiated and the results are evaluated from a personal computer (*PC* in Figure 10).

FPGA-based NoC-Emulators with different error-handling techniques (S&W, S&W SLE) were synthesized on an Altera Stratix II FPGA [22] using the Quartus II 7.1 [23] synthesis software.

To show the performance of the proposed error protection

techniques several experiments have been performed on the FPGA-based emulators. In these experiments data was sent between two network-interfaces using three links and two routing-switches. In one link errors were added to the data words, which could not only be detected and not corrected by the error detection stage.

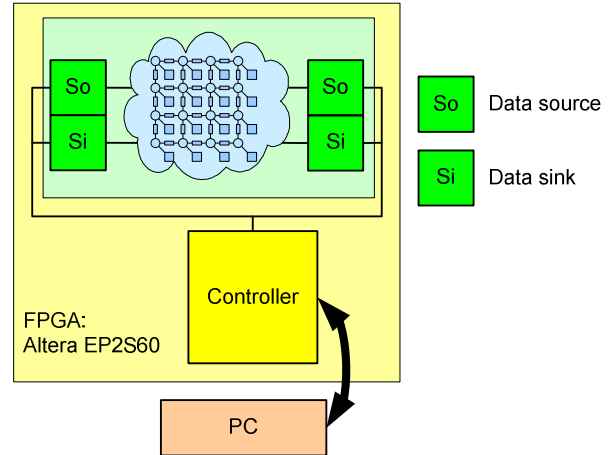


Figure 10: FPGA-based NoC-Emulator

For small error-rates, the S&W SWE outperforms the basic S&W protocol. This better performance comes at the costs of significantly increased area of the NI. The area increases by a factor 8 compared to the S&W protocol and by a factor of 15 compared to the S&F protocol as shown in chapter V. As only not correctable errors have to be retransmitted an alternative to retransmitting corrupted data is to improve the error correction and detection stage and avoid any retransmission.

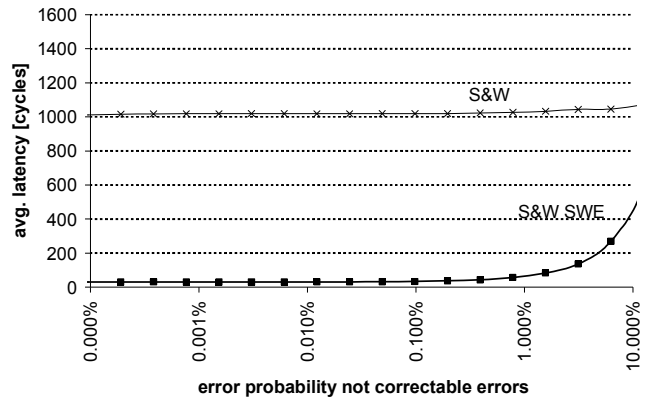


Figure 11: Performance comparison of different error handling techniques

VII. CONCLUSION

Error protection is becoming more and more important in future SoCs. Retransmitting corrupted data words with errors that could not be corrected, as an Error-Handling technique is quite expensive. Using a Send-And-Wait protocol huge performance penalties occur whereas a Send-and-Wait with a sliding window enhancement has much bigger area requirements.

The presented cost models enable a design space exploration in an early design stage as it is suggested in the presented design flow for NoCs.

REFERENCES

- [1] K. Keutzer, A. R. Newton, J. M. Rabaey and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design", *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 19, pp. 1523-1543, 2000.
- [2] S. Y. Borkar, et al., "Platform 2015: Intel® Processor and Platform Evolution for the Next Decade", Intel 2005.
- [3] H. Blume, H. T. Feldkaemper and T. G. Noll, "Model-Based Exploration of the Design Space for Heterogeneous Systems on Chip", *The Journal of VLSI Signal Processing*, vol. 40, pp. 19-34, 2005.
- [4] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm", *Computer*, vol. 35, pp. 70-78, 2002.
- [5] ITRS"International Technology Roadmap for Semiconductors, 2005 Edition, Executive Summary", 2005.
- [6] M. Cuvellio, S. Dey, B. Xiaoliang and Z. Yi, "Fault modeling and simulation for crosstalk in system-on-chip interconnects", *Proceedings of Computer-Aided Design, 1999. Digest of Technical Papers. 1999 IEEE/ACM International Conference on*, 1999, pp. 297-303.
- [7] H. Zimmer and A. Jantsch, "Error-Tolerant Interconnect Schemes", in *Interconnect-Centric Design for Advanced SoC and NoC*, 2005, pp. 155-176.
- [8] T. Lv, J. Henkel, H. Lekatsas and W. Wolf, "An adaptive dictionary encoding scheme for SOC data buses", *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002, pp. 1059-1064.
- [9] D. Chunjie, T. Anup and S. P. Khatri, "Analysis and avoidance of cross-talk in on-chip buses", *Proceedings of Hot Interconnects 9, 2001.*, 2001, pp. 133-138.
- [10] D. Bertozzi, L. Benini and G. De Micheli, "Low power error resilient encoding for on-chip data buses", *Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002, pp. 102-109.
- [11] H. Blume, H. Hubert, H. T. Feldkamper and T. G. Noll, "Model-based exploration of the design space for heterogeneous systems on chip", *Proceedings of Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on*, 2002, pp. 29-40.
- [12] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF: task graphs for free", *Proceedings of Hardware/Software Codesign, 1998. (CODES/CASHE '98) Proceedings of the Sixth International Workshop on*, 1998, pp. 97-101.
- [13] H. Jingcao and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints", *Proceedings of Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, 2003, pp. 233-239.
- [14] H. Blume, T. v. Sydow, J. Schleifer and T. G. Noll, "Petri Net Based Modelling of Communication in Systems on Chip", in *Petri Net: Theory and Application*: ARS-Publishing, 2007.
- [15] M. C. Neuenhahn, H. Blume and T. G. Noll, "Quantitative analysis of network topologies for NoC-architectures on an FPGA-based emulator", *Proceedings of Kleinheubacher Tagung*, 2006.
- [16] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*: Cambridge University Press, 2005.
- [17] Synopsys, "Design Compiler", 2007
- [18] Mentor Graphics, "Encounter", 2006
- [19] Cadence, "Design Framework", 2006
- [20] Synopsys, "NanoSim", 2006.
- [21] Artisan Components, Inc., "TSMC 90nm CLN90G Process SAGE-X v3.0 Standard Cell Library Databook", 2005
- [22] Altera Corporation, "Startix II Device Handbook": Altera Corporation, 2007.
- [23] Altera Corporation, "Quartus II v7.1 SP1", 2007