Analysis of Forward Error Correction Methods for Nanoscale Networks-On-Chip

Teijo Lehtonen
Turku Centre for Computer Science (TUCS)
Turku, Finland
tetale@utu.fi

Pasi Liljeberg
University of Turku
Department of Information Technology
pakrli@utu.fi

Juha Plosila
Academy of Finland
Research Council for Natural Sciences and Engineering
juplos@utu.fi

ABSTRACT

The amount of errors in future nanoscale technologies is expected to increase dramatically when compared to technologies that have line width larger than 90 nm. In nanoscale CMOS circuits fault tolerance is one of the most important design constraints to sustain system reliability at an acceptable level. We analyze different error correcting coding methods for on-chip communication networks of future nanoscale multiprocessor systems. The implemented communication circuits are compared in terms of error correction capability, circuit area and power consumption. In addition, performance of implemented systems is evaluated under different error scenarios by taking into account variable number of single bit errors, burst errors, and their combinations.

Categories and Subject Descriptors

B.8.1 [**Hardware**]: Performance and Reliability—*Reliability, Testing and Fault-Tolerance*

General Terms

Design, Reliability

Keywords

fault tolerance, forward error correction, nanoscale circuits, on-chip communication

1. INTRODUCTION

It is expected that the number of both manufacture-based and run-time generated errors will dramatically increase in future nanoscale systems [1]. Errors originate from noise, process variations and joint effect of these. Impact of different noise sources such as crosstalk, power supply and timing noise is profound in future nanoscale systems due to lower supply voltage, proximity of wires and smaller sizes of active devices. In nanoscale technologies probability of errors caused by neutrons [5], alpha particles and electromigration increases [9]. Furthermore, random dopant fluctuations, the impact of which will increase with technology scaling, cause threshold variations that in turn lead to random variations in delays [6]. Hence, fault tolerance of nanoscale systems needs to be one of the most important design metrics.

The reliability of nanoscale systems is strongly affected by the reliability of communication links, a single malfunctioning link may paralyze a large portion of a system. The errors can be classified into three categories: permanent, intermittent and transients [4]. An efficient fault tolerance method needs to take into account all of these.

There is a number of solutions to cope with the increasing number of errors. For on-chip communication links such as network-on-chip (NoC) links information redundancy ie. coding is the most feasible solution. For now the research has focused on tolerating a very limited number of simultaneous errors, commonly one or two [2, 7, 8, 9]. When considering future chips it is likely that there are more simultaneous errors, some of them are permanent and the others transient, and the probability for burst errors is expected to increase. An efficient fault tolerance approach should take into consideration both multiple simultaneous single errors and burst errors where several adjacent bits are erroneous.

There are two possible ways for the error recovery when an error has been detected. In automatic repeat query (ARQ) the receiver informs transmitter about the errors and requests a retransmission. Other approach is forward error correction (FEC), where the information carried by the check bits is used to correct the error. Both of the approaches have their pros and cons. ARQ has been reported to be more power-efficient [2]. This is based on the assumption that since the error detection capability t_d of a code is higher than the error correction capability t_c ($t_d = d - 1$, $t_c = \lfloor (d-1)/2 \rfloor$, where d is the code distance), the link can use lower voltage for getting the same bit error rate. Also the FEC decoder is more complex than the one for ARQ and thus consumes more energy. On the other hand the time needed for retransmissions is the drawback of the ARQ approach. If errors occur infrequently the additional delay may be acceptable, but the approach cannot guarantee certain throughput. In the presence of permanent errors the retransmission principle does not work, while FEC with a code strong enough still works.

We study several FEC codes beyond simple codes that correct only single errors. We analyze their error tolerance against multiple simultaneous single errors, different lengths of burst errors and mixtures of both. We present hardware realizations for the codes and report the performance values.

Nano-Net 2007 September 24-26, 2007, Catania, Italy.
 ISBN 978-963-9799-10-3
DOI 10.4108/ICST.NANONET2007.2035

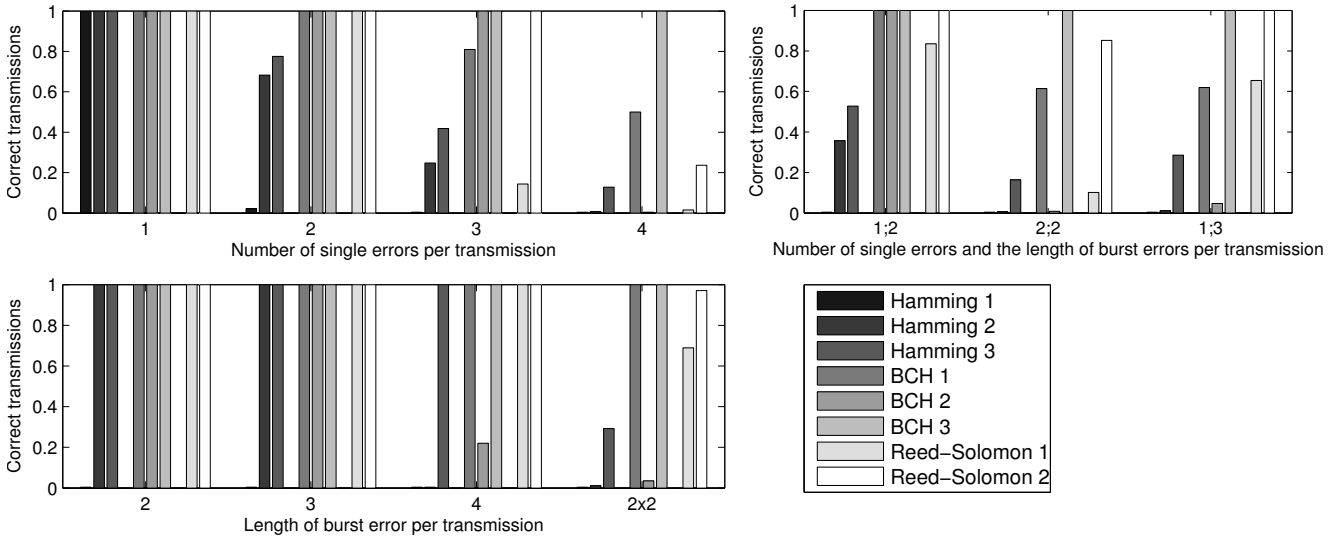


Figure 1: Error correction capability of the coding approaches under different fault scenarios.

2. CODING APPROACHES

For finding the codes for the analysis we need to set some objectives. The code must correct at least two simultaneous single errors ($t \geq 2$). In this way the system still tolerates single transients in the presence of a single permanent error. The coding approach must also correct at least burst errors of length 3 ($b \geq 3$). This takes care of the situation when a single erroneous wire affects its adjacent wires. The coding approach must not introduce too much redundancy. The limit for extra wires is set to 50 % of the original number of wires, thus the code rate (width of dataword / width of codeword) must be at least $\frac{2}{3}$ ($R \geq \frac{2}{3}$). Since the target is on-chip communication link, the encoding and decoding should be easily and effectively implementable using standard logic. The width of the data link is set to 64, thus data width $k = 64$ and code length $n \leq 96$.

Hamming codes [3] are the most widely used codes in the previous research of NoC link error protection [2, 7, 8, 9]. The distance of Hamming codes is 3 or 4 for extended Hamming code, so it can correct single errors ($t = 1$). Therefore, the Hamming codes do not fill the minimum requirements set. Nevertheless, we take Hamming code ($n = 71$, $k = 64$) with rate $R = 0.90$ as a reference to our other codes.

Interleaving is an efficient way to cope with burst errors. It means partitioning the data word into parts and encoding each of them separately and after that taking one bit of each at a time to get the final code word. The interleaving affects mainly the burst error tolerance but it also has an effect on tolerance against simultaneous single errors. Many single errors can be corrected if they affect separate interleaving sections. Therefore, Hamming codes together with interleaving could be a possible solution. Two approaches are analyzed. In the first one there are three interleaving sections. The data has been divided to them 22+21+21=64 and the used codes are 2x(26,21)+(27,22) resulting to rate $R = 0.81$. In the other approach the data is divided into four 16-bit parts and each of them is coded with Hamming code (21,16). The rate for this approach is $R = 0.76$.

Table 1: The analyzed codes.

Code	n	k	Rate	Notes
Hamming 1	71	64	0.90	
Hamming 2	79	64	0.81	2x(26,21)+(27,22)
Hamming 3	84	64	0.76	4x(21,16)
BCH 1	88	64	0.73	2x(44,32)
BCH 2	85	64	0.75	
BCH 3	92	64	0.70	
Reed-Solomon 1	85	65	0.75	RS(17,13)
Reed-Solomon 2	95	65	0.67	RS(19,13)

The *Bose-Chaudhuri-Hocquenghem* (BCH) codes [3] are a class of linear block codes that can be easily constructed according to specifications for correcting as many errors as is required. We analyze two different BCH codes. One with error correcting capability of three and the other with four. The codes are (86,64) and (92,64) and their rates 0.75 and 0.70 respectively. BCH codes can also be combined with interleaving. An approach with two interleaving sections, both encoded using double-error correcting code (44,32), is analyzed. The rate of this approach is 0.73.

The abovementioned BCH codes are binary but also nonbinary BCH codes exist. The most interesting ones of these are the *Reed-Solomon* (RS) codes [3]. The RS codes are optimal meaning that they provide the maximum distance at the used number of check symbols. The use of a nonbinary code with binary coded symbols is an effective way for battling against burst errors. On the other hand it is not very effective in single error tolerance. Two RS codes are analyzed. The both codes use elements of Galois Field 2^5 as symbols, so five bits are needed to encode each symbol. The first code is (17,13) having $n = 85$, $k = 65$ and rate $R = 0.75$. The second one has one larger error correction capability and it is (19,13) Reed-Solomon code resulting to $n = 95$, $k = 65$ and $R = 0.67$.

The codes are listed in Table 1.

3. FAULT TOLERANCE ANALYSIS

In order to compare the error correction capability of the presented codes, we have run simulations on different error scenarios. The scenarios include 1 to 4 single errors, bursts of length 2 to 4 and combinations of them. The results are presented in Figure 1, where the probability for correct transmission is presented for different codes under different error scenarios. In the top-left graph the single error correction capability is illustrated. We see that all presented

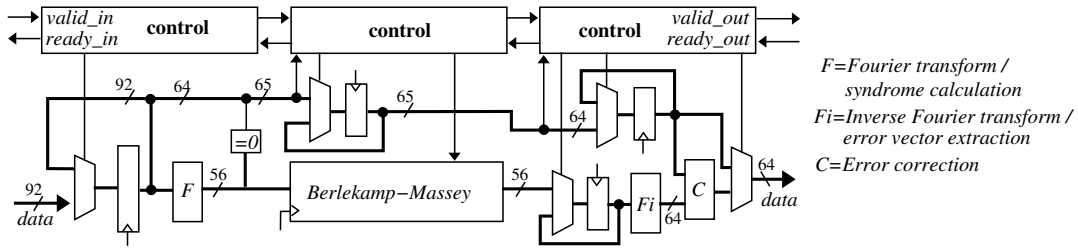


Figure 2: Structure of the decoder for code *BCH 3*.

codes manage single errors but only one (*BCH 3*) corrects four simultaneous single errors in all cases. The codes that use interleaving correct more errors than the codes with similar error correction capability but no interleaving. This is due to the fact that in some cases the errors hit different interleaving sections and the code can correct them. See for example the codes *Hamming 2* and *Hamming 3* with two simultaneous single errors. *Hamming 2* has three interleaving stages and therefore statistically every third case is such that the two errors hit the same interleaving section, which cannot be corrected. Similarly there are four interleaving stages in *Hamming 3* and in one fourth of all the cases the error cannot be corrected.

Interleaving is an efficient way for tolerating burst errors as are also the Reed-Solomon codes, which can be seen from the bottom-left graph of Figure 1. BCH codes correct bursts as long as the total number of errors does not exceed their error correction capability. An interesting result is the case where instead of one long burst there are two smaller bursts (2x2). The codes with Hamming coding and interleaving are not effective in this scenario, since only seldom the two bursts hit the code so that there is only one error per section.

The combined cases presented in the third graph give a deeper insight into the correction abilities of the codes. From the graph it can be seen what happens when there is one permanent error in the link and a burst error occurs. The BCH and RS codes can handle the situation better than Hamming codes even with extensive interleaving.

4. CIRCUIT REALIZATIONS

The on-chip communication links are typically parallel links while the most commonly presented encoding and decoding circuits are designed for serial transmission and based on linear feedback shift registers. Therefore, other methods besides these have to be used for the encoding and decoding. On the other hand interleaving for parallel links is much simpler than for serial links. As for serial links a memory block would be needed, in parallel links the interleaving is just routing the wires in a specific manner.

The encoding of linear block codes can be done with matrix multiplication $c = aG$, where a is a dataword vector of length k, G is a $k \times n$ generator matrix and c is a codeword vector of length n. Since we are using systematic (separable) codes, only the first $n-k$ codeword symbols have to be calculated and the rest are just the symbols of the dataword. The binary matrix multiplication is simply calculating parity bits, which in hardware means trees of *xor* gates.

The decoding can be done by first calculating the syndrome by matrix multiplication $s = uH^T$, where u is a received code word vector of length n ($u = c + e$, where c is a transmitted codeword and e an error vector, both of length n), H^T is a transpose of $(n-k) \times n$ parity check matrix and s is a syndrome vector of length $n-k$. The syndrome gives the index of error vector in the coset leader table, so the error vector e (length k) can be easily determined. A coset is the set of all the error vectors that produce a same syndrome and the coset leader is the one having the minimum weight, ie. the minimal number of errors. The correction is done by taking *xor* function separately for each bit from the data part of the received codeword and the error vector.

The abovementioned decoding method is limited by the size of the coset leader table. When the number of check bits $n-k$ is high, the table is impractically large. In the *BCH 1* code there are 12 check bits, so the table has 4096 rows, which is quite at the upper limit of this method. For the other BCH codes the table row count would be over 2^{20} and therefore another method must be used for them.

The decoding of BCH codes can be done using *Berlekamp-Massey* (B-M) algorithm [3]. The algorithm is iterative and it needs $2t$ iterations, where t is the error correction capability of the code. The calculations are done in Galois Field 2^m, where $m = 7$ for the codes *BCH 2* and *BCH 3*. In addition to the actual algorithm a pre-processing circuit is needed. Using Fourier transform in $GF(2^m)$, $2t$ syndromes are calculated. This basically means quite similar trees of *xor* gates in hardware as for the syndrome calculation explained above. The error vector e is extracted by using a method called *Chien search* to find the zeros of polynomial $\Lambda(x)$ output by B-M algorithm. [3]

The decoding of RS codes is also based on B-M algorithm. The main difference is that in addition to error vector e also the error values are needed to do the error correction. The error vector e points the erroneous GF(2^5) symbol (5 bits) and the actual correction is done by adding the error value of that particular symbol and the transmitted symbol itself. The error values can be extracted using *Forney* algorithm with inputs $\Lambda(x)$ and $\Gamma(x)$, both gained from B-M algorithm. Also the Fourier transform requires slight changes compared to the one used for BCH decoding. In RS coding all the calculations are done with GF(2^5) symbols while for binary BCH codes the syndrome calculation is just calculating parities of different sets of bits. [3]

The target in the design approaches is to achieve a high throughput. Therefore the designs are pipelined. The encoding is done in one pipeline stage but the decoder units have at least two pipeline stages. In the simpler decoders the first pipeline stage is for deinterleaving (if necessary) and for calculating the syndrome. The error correction is done in the second pipeline stage. In the decoding approach using B-M algorithm three pipeline stages are identified: the Fourier transform ie. syndrome calculation, the actual algorithm and the error vector extraction and correction.

The throughput in the case, where there is no errors is maximized. This means that the parts which are needed only for error correction (not for detection) are bypassed

if it results to higher throughput. For the codes based on Hamming coding, there is no benefit in bypassing the error correction part, so the decoders consist of simply two-stage pipeline and the throughput and latency are always the same. For the BCH and RS codes it is beneficial to bypass the correction part, if no errors are present and thus resulting in two different throughput and latency values, one with and the other without errors in the received codeword.

Because the throughput varies, signals for indicating when data is ready (*valid*) or when the circuit is ready for taking in new data (*ready*) are introduced. The same signals are convenient for indicating breaks in data transmission. For the sake of compatibility, the signals are inserted to every encoder and decoder.

The structure of the decoder for the *BCH 3* code is presented in Figure 2. The three pipeline stages can be seen clearly. The first stage (Fourier transform) takes one clock cycle. However, if the following stages are not ready, the first stage holds the data until it can be forwarded. The outputs of the first stage are the syndromes 56 ($2tm$) bits, data word 64 (k) and 1-bit information indicating if correction is required. The indication for the need of correction is obtained by checking if all the syndromes are zero.

The second stage either passes the data directly to the next stage or goes through the B-M algorithm and passes its result together with the data. The algorithm takes 8 ($2t$) clock cycles but if no errors are present the data is forwarded in only one clock cycle. The final stage either forwards the data to output (1 clock cycle) or extracts the error vector and does the correction before outputting (2 clock cycles).

Table 2: Characteristics of the designed circuits.

Circuit area [$(\mu\text{m})^2$]

Code	Encoder	Decoder	Total (gate eqv.)
Hamming 1	4683	8881	13564 (3090)
Hamming 2	3628	8837	12464 (2839)
Hamming 3	3335	8280	11645 (2653)
BCH 1	4626	61616	66242 (15089)
BCH 2	5023	74682	79704 (18156)
BCH 3	5807	107679	113486 (25851)
RS 1	6054	36688	42743 (9736)
RS 2	8090	56624	64714 (14741)

Power consumption [mW]

Code	Encoder	Decoder	Total
Hamming 1	0.27	0.45	0.73
Hamming 2	0.17	0.39	0.56
Hamming 3	0.14	0.31	0.44
BCH 1	0.19	2.01	2.20
BCH 2	0.14	3.11	3.25
BCH 3	0.15	4.68	4.83
RS 1	0.26	1.57	1.82
RS 2	0.40	2.85	3.25

Clock frequency [MHz], throughput [MWord/s] and decoder latency [clock cycles]

Code	Clock frequency	Throughput / Latency no errors	Throughput / Latency when errors
Hamming 1	1800	1800 / 2	1800 / 2
Hamming 2	2000	2000 / 2	2000 / 2
Hamming 3	2000	2000 / 2	2000 / 2
BCH 1	1800	1800 / 2	900 / 3
BCH 2	1100	1100 / 3	183 / 9
BCH 3	1100	1100 / 3	138 / 11
RS 1	1300	1300 / 3	325 / 9
RS 2	1300	1300 / 3	217 / 10

The designs were created using VHDL, mapped to 90 nm technology, synthesized and simulated to verify their operation. The place and route phase was omitted since the synthesis results give enough information for structure comparison and indicate the feasibility of the proposed structures. The timing margin was set to 20 % so the circuits were designed for a clock frequency 20 % higher than the one reported in Table 2 and they were also synthesized to meet the timing requirements of that higher clock frequency. For instance, *Hamming 2* and *Hamming 3* can operate at 2.4 GHz in normal operating conditions. This gives a reasonable margin for process variance and changes in the operation environment. Additionally, the outputs were set to stabilize 90 ps before the next clock edge, which is enough to satisfy the setup time constraints of the next module's input buffers (flip-flop setup time in the used technology is typically 70 ps). The design results are presented in Table 2.

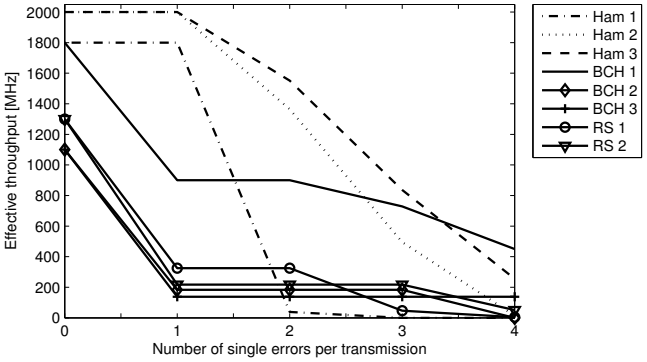
5. DISCUSSION

Based on the results above, we now discuss the suitability of the coding approaches for the error protection of on-chip links in future nanoscale technologies. At the moment there is no clear data available that could be used to model the wires in nanoscale technologies, especially the relation of single and burst errors in them. We can only conclude that there will be multiple single errors as well as burst errors. So based on this we cannot rank out any of the coding approach, except the one using Hamming coding over the whole word width. This approach was in fact taken as a reference to the analysis, since it is the most commonly presented approach in previous research.

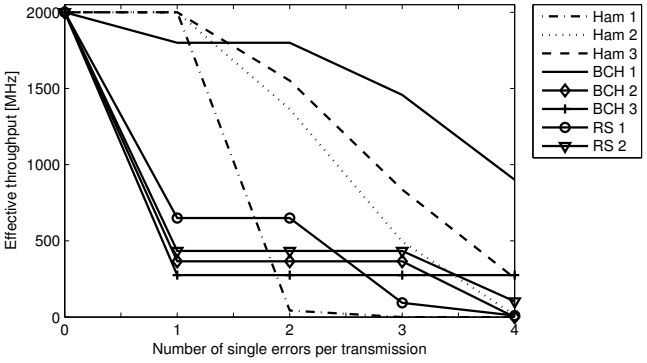
One possible way to compare the approaches is to consider their effectiveness under different types of errors. What happens to the error correction capability if one of the wires is permanently corrupted, e.g. due to electromigration. Almost all of the circuits (excluding *Hamming 1*) maintain at least somehow their tolerance against single errors (2 single errors) but for burst errors (combined 1;2) the approaches based on Hamming codes do not perform that well. All the approaches (again excluding *Hamming 1*) perform reasonably well under burst errors. However, it cannot be seen from the graphs that Reed-Solomon codes are the best ones for burst error tolerance. If the graphs were plotted for bursts of length 6, the Reed-Solomon codes would show error tolerance of 100 % while the result for all the others would be nearly zero.

The area overhead of the circuits should be considered in their target context. For instance in a mesh-shaped NoC with bidirectional links, there are six unidirectional links per resource, thus six encoder/decoder pairs. If a resource is of size 2 mm x 2 mm, the area overhead of the coding approaches in 90 nm technology is 2 %-17 %. Quite often, area overhead of 17 % is unacceptable. On the other hand, as the scaling proceeds towards nano regime, the size of logic gates is decreasing. This enables integration of more functionality to resources, still maintaining their size. In this scenario, the relative area overhead of codecs will decrease.

One of the main concerns in on-chip communication is the throughput. This is also one of the good features of the FEC scheme, since no time consuming retransmissions are needed. The circuit realizations show a huge variation



(a) The results on 90 nm technology.



(b) The speculated results for nanoscale technology.

Figure 3: Effective throughput: the throughput of correctly transmitted words.

in throughput. The comparison of combined throughput and error correction capability would be interesting. We do this by calculating the throughput of correctly transmitted words in different error scenarios. We call this the effective throughput. The effective throughput for circuit realizations done at 90 nm technology are presented in Figure 3(a). It can be seen that the effective throughput of *Hamming 1* drops to zero as the number of errors is more than one, and that the effective throughput of *BCH 2* and *BCH 3* is rather low the whole time because their base throughput is low. An interesting thing to notice are the reasonably high values for the interleaved codes. *Hamming 3* gives the highest effective throughput for error count below four, while for four errors *BCH 1* gives the best effective throughput.

As the technology is scaled down further into nanoscale dimensions, it is expected that the delay of logic will decrease, while the delay of wires, especially longer on-chip links, stays the same. This means that we could expect our encoders and decoders to work faster, but at the same time there is a limit for the speed of the link. In Figure 3(b) we have speculated a situation, where the speed of the logic would double in some technology smaller than 90 nm and at the same time the link throughput would be limited to 2 GHz. Now *BCH 1* becomes the most attractive approach already at two simultaneous errors and also *BCH 3* is better than the Hamming approaches with four errors.

The circuits were designed to minimize the error-free operation latency: the decoding takes only 2-3 clock cycles in all the designs. In the presence of errors the decoding latency increases. For *BCH 2*, *BCH 3* and RS decoder the latency is approximately 10 clock cycles. In a system using ARQ, a single retransmission takes at least 4 clock cycles: 1 for decoding, 1 for retransmission request, 1 for retransmission and 1 for decoding, so the 10 clock cycles is quite a lot. However, in the presence of permanent errors retransmission becomes useless.

The power consumption is an important criterion in designing on-chip communication. Most of the power is consumed in driving the long capacitive wires. In this work we have concentrated on the encoder/decoder circuits while a complete analysis should take into consideration the number of wires in different architectures and the energy consumed by the drivers. This will be a part of our future research.

6. CONCLUSIONS

Future digital systems implemented with nanoscale technologies are very sensitive to errors emerging from various sources. In order to construct a reliable and fault tolerant system, efficient error correction methods are needed. In this paper, different forward error correcting methods for nanoscale network-on-chip were studied and analyzed in terms of error correction capability, power consumption, throughput and area. The focus was on error scenarios where several single error and burst errors could occur simultaneously and independently of each other.

It was shown that the complex methods, such as BCH with interleaving and Reed-Solomon based methods, were efficient but the drawback was larger area and reduced throughput. However, such complex methods are feasible in future nanoscale technologies with a reduced cost per transistor and a reduced delay of logic.

7. REFERENCES

[1] International technology roadmap for semiconductors 2005. http://public.itrs.net.

[2] D. Bertozzi, L. Benini, and G. D. Micheli. Error control schemes for on-chip communication links: The energy-reliability tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):818–831, June 2005.

[3] R. E. Blahut. *Algebraic Codes for Data Transmission.* Cambridge University Press, 2003.

[4] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, 2002.

[5] M. Lajolo, M. Reorda, and M. Violante. Early evaluation of bus interconnects dependability for system-on-chip designs. In *International Conference on VLSI Design*, pages 371–376, 2001.

[6] H. Mahmoodi, S. Mukhopadhyay, and K. Roy. Estimation of delay variations due to random-dopant fluctuations in nanoscale cmos circuits. *IEEE Journal of Solid-State Circuits*, 40(9).

[7] S. Murali et al. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, Sep.-Oct. 2005.

[8] S. Sridhara and N. Shanbhag. Coding for system-on-chip networks: A unified framework. *IEEE Transactions on VLSI Systems*, 13(6), June 2005.

[9] H. Zimmer and A. Jantch. A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip. In *IEEE/ACM/IFIP Int. Conf. on Hardware/Software Codesign and System Synthesis*, pages 188–193, Oct. 2003.