# Fault Tolerance Techniques for Hybrid CMOS/Nano Architecture

Aissa Melouki, Saket Srivastava and Bashir M. Al-Hashimi
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{am06r,ss3,bmah}@ecs.soton.ac.uk

**Abstract**

We propose two fault tolerance techniques for hybrid CMOS/nano architecture implementing logic functions as Look-Up Tables. We compare the efficiency of the proposed techniques with recently reported methods that use single coding schemes in tolerating high fault rates in nanoscale fabrics. Both proposed techniques are based on error correcting codes to tackle different fault rates. In the first technique, we implement a combined two dimensional coding scheme using Hamming and BCH codes to address fault rates greater than 5%. In the second technique, Hamming coding is complemented with Bad Line Exclusion technique to tolerate fault rates higher than the first proposed technique (up to 20%). We have also estimated the improvement that can be achieved in the circuit reliability in the presence of Don't Care Conditions. The area, latency and energy costs of the proposed techniques were also estimated in the CMOS domain.

## I. Introduction

Molecular electronics holds the promise to overcome the physical limitation of lithography-based VLSI technology and offer the possibility of significantly denser circuits. However, tremendous growth in device density will be accompanied by a substantial increase in hard [1], [2], [3] and soft [4], [5], [6] faults. To achieve acceptable levels of manufacturing yield and computational reliability, fault tolerance must be integrated into the design flow of nanoscale circuits.

Much work has already been done in the area of fault tolerance/avoidance for nanotechnology to increase circuit reliability in the presence of increased hard and soft error rates. One of the proposed techniques, Triple-Modular-Redundancy (TMR), is based on the use of three copies of the same module and an arbitration unit [7], [8]. TMR technique fails when there are faults in more than one module. The reliability of TMR is also limited by that of the final arbitration unit making this approach insufficient in the presence of high defect rates [7]. Reconfiguration is another technique that can circumvent physical defects by first mapping defects on reconfigurable fabrics then synthesising a feasible configuration to realise an application for each nanofabric instance [9], [10], [11]. However, defect mapping and reconfiguration is performed on a per chip basis which poses a scalability challenge. The

prohibitively low reliability of these new nanodevices dictates that they must be interfaced with CMOS circuits to tolerate the inevitable high fault rates. This leads to a new paradigm of hybrid CMOS/Nano architecture [12], [13], [14], [15] to perform reliable computing using unreliable components (nanodevices). In this architecture, nanoscale devices offer a highly dense fabric for data storage and computation, whereas CMOS components are utilised for interfacing and for highly critical circuit operations. Both CMOS circuits and high fault rates will reduce the net density delivered by these nanodevices.

Recently, Error Correcting Codes (ECC) have been proposed as a promising approach to improve the reliability and yield of heterogeneous CMOS/nanodevices systems. In [12], [6], ECCs were mainly used for the suppression of soft errors rather than physical defects i.e. maintaining the fault tolerance level rather than enhancing defect tolerance. In [12], authors suggested a hybrid fault tolerance technique based on Hamming code and reconfiguration. In [6], the authors proposed an implementation of ECCs based on the theory of Markov random fields (MRF) to combat soft faults thus increasing the reliability of hybrid systems. In [13], two nanoelectronic memory fault-tolerant system design approaches based on Bose-Chaudhuri-Hocquenghem (BCH) codes were suggested. Previously, single error correcting codes such as Hamming and BCH have been used in the context of reliable memory designs [12], [16]. In [16], the authors explored combining error correction codes with various repair techniques to combat the high defect rates in hybrid CMOS/Nano fabrics with particular focus on memory architectures. The previous works have only addressed fault tolerance in memory architectures. ECC-based techniques can also be applied to memory-based implementation of logic circuits (i.e. Look-up Tables) which includes Don't Care Conditions (DCCs). The presence of DCCs in Boolean logic functions presents a strong case to apply these techniques to circuits implemented as Look-Up Tables (LUTs) on CMOS/nanodevice fabrics. As we will demonstrate in this work, the existence of DCCs can be exploited in this type of architecture since it helps in masking of erroneous bits which is not possible in memory design.

Fig. 1 gives an overview of the targeted hybrid CMOS/nano architecture. The proposed architecture is technology-independent i.e. the nanoscale fabric is built using any of the recently proposed nanodevices including carbon nanotubes (CNT) or silicon nano-wires (NW). The techniques proposed in this work target CMOS/nano computational architecture incorporating a LUT implementation of logic functions, as outlined in [17]. LUT implementation is an effective functional-coding approach that provides low-level protection of individual Boolean functions [18], [19]. In our experiments, the LUTs under test are represented by randomly generated symmetric matrices of sizes ranging from $2^3 \times 3$ to $2^6 \times 6$ where the probability of 0 and 1 are equal. The errors are injected randomly in the nanofabric causing the corresponding bits to change their values (i.e. $1 \rightarrow 0$ or $0 \rightarrow 1$). We have assumed random distribution of errors to simulate the worst case scenario as correlated errors are technology specific. The proposed fault tolerant techniques are based on ECC and partial redundancy to address the permanent and transient faults in nanoscale LUTs. In the first technique, we implement a two dimensional coding scheme using Hamming and BCH codes to address both hard and soft errors in the presence of high fault rates. In the second technique, we target the high physical defect rates in the nanofabric by integrating ECCs with bad line exclusion technique. In this technique, the high bit density offered by nanodevices is exploited to provide the necessary spare rows to compensate for

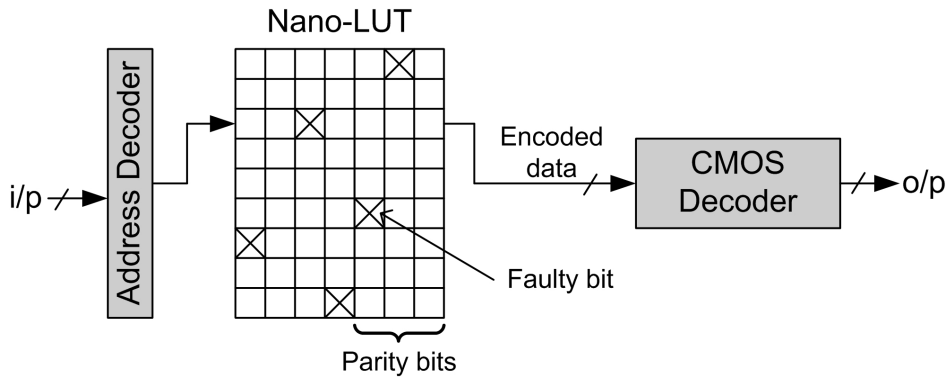


Fig. 1. Hybrid Nano/CMOS Architecture Overview

the defective lines. While the exact manufacturing defect rate and transient error rate are not yet pinpointed, it is believed that they will easily exceed 10% [2]. The authors in [17] assume small fault rates (less than 10%) in nanofabrics for small LUT sizes with 50% of LUT entries set as Don't Care Conditions (DCC). We first investigate the effectiveness of our proposed techniques over the methods [17], [12] in high defect rate scenario.

## II. Primitives

We first examine the ineffectiveness of using single ECCs such as Hamming and BCH in the presence of high error rates for different LUT sizes.

### *A. Hamming*

Hamming is a single-error-correcting and double-error-detecting code i.e. the code is capable of correcting one error and detecting two errors in a codeword. A typical Hamming code is $(2^m-1,\ 2^m-m-1)$, in other words, for $2^m-m-1$ data bits, $m$ parity bits need to be added for full protection.

To demonstrate the reliability improvement that can be achieved from the techniques discussed in this work, we have performed experiments on randomly-generated symmetric LUTs where the probability of 0 and 1 are equal. The LUTs are of sizes ranging from $2^3\times 3$ (3 inputs, 3 outputs) to $2^6\times 6$ (6 inputs, 6 outputs). The circuit failure probability $F_m$, resulting from randomly injecting $m$ errors, is obtained by calculating the ratio of defective LUTs after decoding to the total number simulation iterations $I=5000$. In this work, we assume that a nanodevice is subject to both stuck-open and stuck-short defects with equal probabilities. We also assume that errors are uniformly distributed across the fabric where both physical and transient errors are random and statistically independent. For comparison purposes, we use the simple Hamming code [17] as a reference point for the evaluation of our proposed techniques.

Theoretically, the probability of a row of length $l$ having $m$ defective bits is given by the following binomial equation:

$$P(m)=\binom{l}{m}P^m(1-P)^{l-m} \tag{1}$$

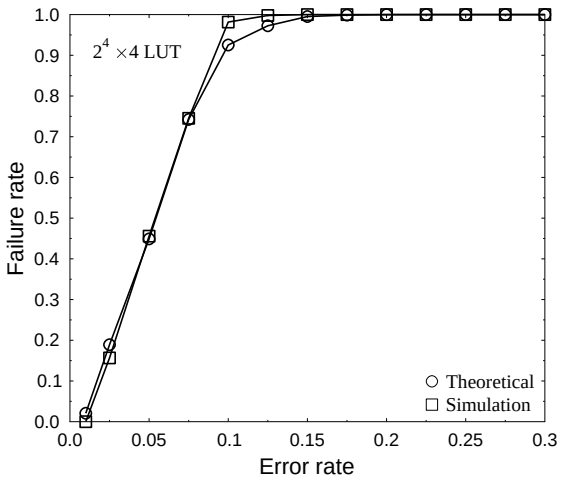


Fig. 2. Hamming - Failure rate obtained through simulation and theory

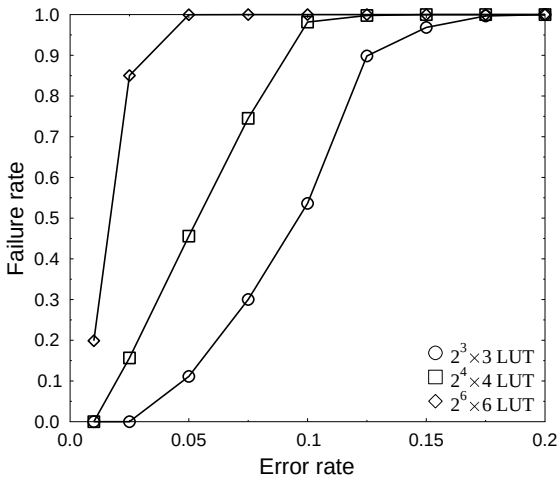


Fig. 3. Hamming - Effect of varying LUT size on failure rate

where $P$ is the error rate of the fabric. The probability that the Hamming decoder fails to correctly decode an erroneous codeword is equal to the probability of having more than one error per row. Using equation (1), this is given by the following equation:

$$P_{row} = \sum_{k=2}^{r+r_{par}} \binom{r+r_{par}}{k} P^k (1-P)^{r+r_{par}-k} \tag{2}$$

where $r$ and $r_{par}$ are the number of bits and number of parity bits in a row respectively. The failure rate of a LUT with $c$ columns is equal to the probability that at least one row is defective and it can be computed as:

$$P_{failure} = \sum_{k=1}^{c} \binom{c}{k} P_{row}^k (1-P_{row})^{c-k} \tag{3}$$

In the case of $2^4 \times 4$ LUT, equations (2) and (3) can be rewritten as:

$$P_{row} = \sum_{k=2}^{4+3} \binom{4+3}{k} P^k (1-P)^{4+3-k}$$

$$P_{failure} = \sum_{k=1}^{16} \binom{16}{k} P_{row}^k (1-P_{row})^{16-k}$$

Fig. 2 illustrates the failure rate obtained both theoretically (using equations 2 and 3) and through the above simulation procedure. As can be seen, the two graphs are almost identical, validating the derived theoretical equations.

Fig. 3 shows the variation of failure rate with respect to several percentages of injected error rates for different LUT sizes. For defect rates as small as 1%, the Hamming code perfectly detects and corrects all faults for LUTs of sizes smaller or equal to $2^4 \times 4$ as reported in [17]. However, it can be seen that even for a small LUT size of

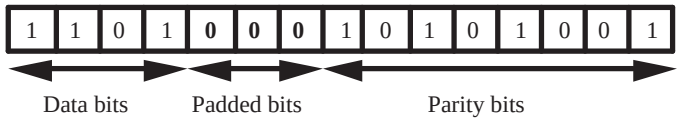


Fig. 4. BCH - Padding

$2^3 \times 3$, and error rate greater than 5%, more than 10% of circuits fail. We can also observe that as the LUT size increases, the fault tolerance of the circuit falls more rapidly indicating the inefficiency of this scheme.

### *B. Bose-Chaudhuri-Hocquenghem (BCH)*

In this section, we examine fault tolerant approach using stronger BCH coding and evaluate its efficiency in dealing with the high defect probabilities in nanoscale LUTs. BCH is a multilevel, variable length and easy to decode ECC used to correct multiple random errors in a codeword. The simplest form of BCH codes is the single error-correcting BCH(7,4,1) which is equivalent to Hamming code. We first examine BCH [13] with 0% Don't Care Conditions using BCH(15,7,2), which adds 8 parity bits in order to detect and correct 2 errors in the codeword. The required word length is 7 bits; however the size of each entry in our LUT is only 4-bits in the case of $2^4 \times 4$ LUT. Therefore, we need to pad the data bits with the necessary bits so that it is equal to the required data word size, as shown in Fig. 4.

The graph shown in Fig. 5 represents the failure rate achieved using the 2-bit error-correction BCH and in the absence of DCCs. The fault tolerance obtained from the simulation results revealed a performance very similar to Hamming. The reason behind this is that even though the BCH(15,7,2) code can tolerate more errors than single error correction techniques, there is a higher probability of errors per codeword in LUTs. The padded bits and the redundant bits added to the data word doubles the probability of faults in each entry of the LUT. The codeword is 15-bit long for the 2-bit error-correction BCH which is twice longer than the 7-bit long codeword for the single error correcting Hamming code. Hence, strong ECCs have the capability of tolerating more errors at the cost of more parity bits added to the codeword, which in turn makes them more vulnerable to higher fault rates; and hence a rapid drop in their efficiency as the fault rate increases in the LUTs.

Instead of coding row entries in LUTs, we use stronger ECCs such as BCH codes to encode columns. BCH(31,16,3) for example can detect and correct 3 errors per column, but at the cost of adding 15 parity bits. The results obtained from simulations are shown in Fig. 5. For low error rates, BCH exhibits a better performance than Hamming. At 5% of errors, we can notice a 70% improvement in failure rate over Hamming. However, when errors exceed 10%, this coding technique completely fails.

## III. Proposed Fault Tolerance Techniques

In this section, we present two different hybrid fault tolerant techniques to address the high fault rates in nano-LUTs. As we have seen earlier, single error correction schemes prove to be inefficient in dealing with such high error rates. The first proposed approach combines Hamming and BCH to target higher error rates (greater than 5%), while the second technique combines Hamming with Bad Line Exclusion to address error rates as high as 20%.

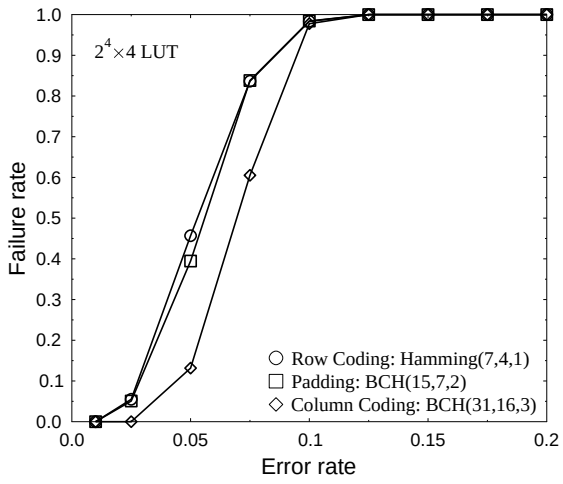


Fig. 5. Failure rate comparison between different 1-D coding techniques

### *A. Combined Two-Dimensional Coding : Technique 1*

To reduce the failure rate for fault rates exceeding 5%, we implemented a two-dimensional coding technique based on Hamming and BCH codes (also known as *product code* [20]). The idea is to encode both rows and columns in LUTs as shown in Fig.6(a). The single error correcting Hamming code is used to encode data bits in each row of the LUT, and then a BCH code is used to encode each column. Retrieving data from the encoded LUT comprises of two decoding steps. In the first step, columns are first decoded using the BCH decoder. This step will allow the detection and correction of the biggest portion of errors because of the capability of the BCH decoder to correct more errors in the codeword than the Hamming decoder. Then, in the second decoding step, the Hamming decoder is used to remove the remaining faults.

We can further improve the fault tolerance of this technique by using systematic BCH code along with *check bits* as illustrated in Fig.6(b). In systematic block codes, data bits remain unchanged in the codeword, and the parity bits are attached to the end of the data bit sequence. We exploit the fact that the number of 1's in any wrong decoded word will most probably be different from the number of 1's in the expected correct word. Therefore, we use check bits to store the number of 1's of each column after all row entries of LUT are encoded using Hamming. If the number of faults per column exceeds the error correcting capability limit of Systematic BCH, the BCH decoder will generate the wrong output and hence cause the entire technique to fail. Therefore, to avoid failure, the check bits are always compared with the number of 1's of the output of BCH decoder, if they are not equal, the codeword remains unchanged and all the faults in the first 16 bits of the corresponding column are left to the second iteration of decoding to be corrected using the Hamming decoder. The flow chart in Fig. 7 illustrates the decoding process to retrieve row $M$ from a LUT of size $2^N \times N$.

Assuming the same error probability $P$ for each bit, the probability of having $m$ defective bits in a column of

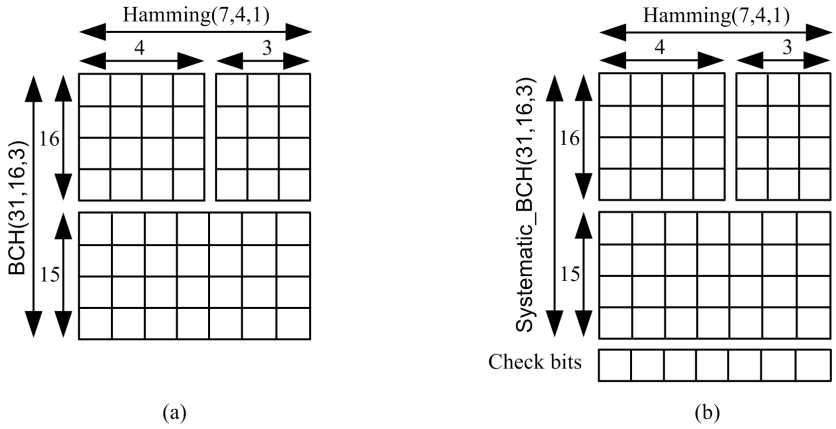


Fig. 6. 2D Coding for $2^4 \times 4$ LUT (a) Hamming & BCH (b) Hamming & Systematic BCH with check bits. In the row encoding, Hamming(7,4,1) is used to detect and correct one error per row by adding three parity bits. BCH(31,16,3) is used to encode columns can detect and correct three errors per column at a cost of adding 15 parity bits.

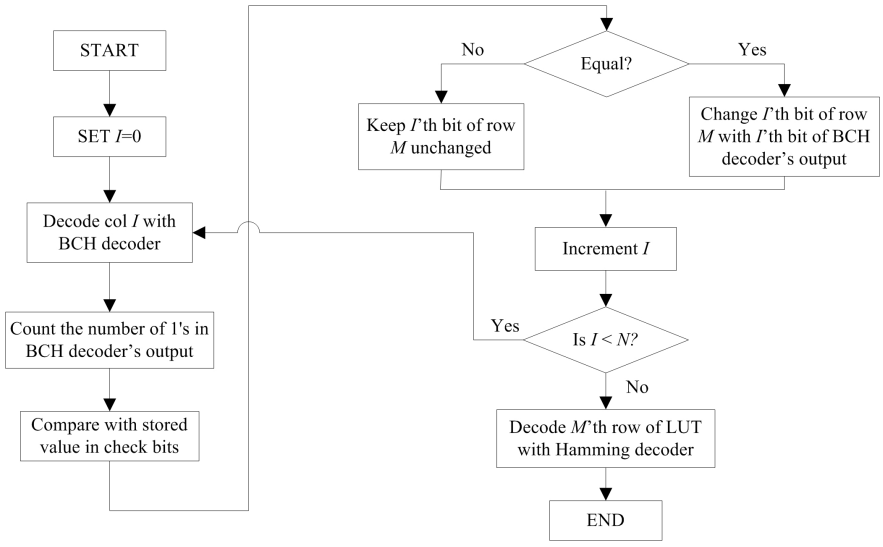


Fig. 7. Flow chart to illustrate the multiple-step decoding process of the proposed 2D coding technique

length $(c + c_{par})$ follows the binomial distribution given in equation (1). Therefore, the probability that the BCH decoder fails to correct a column because the number of faults exceeds its correction capability *bch_err* is given by:

$$P_{col} = \sum_{k=bch\_err+1}^{c+c_{par}} \binom{c+c_{par}}{k} P^k (1-P)^{c+c_{par}-k} \tag{4}$$

where $c$ and $c_{par}$ are the number of bits and number of parity bits in a column respectively.

The BCH correction of columns reduces the probability of a bit being erroneous by a factor of $P_{col}$. Therefore, the remaining faults which are randomly distributed over the rows will have a new error probability $P_{new}$ that is

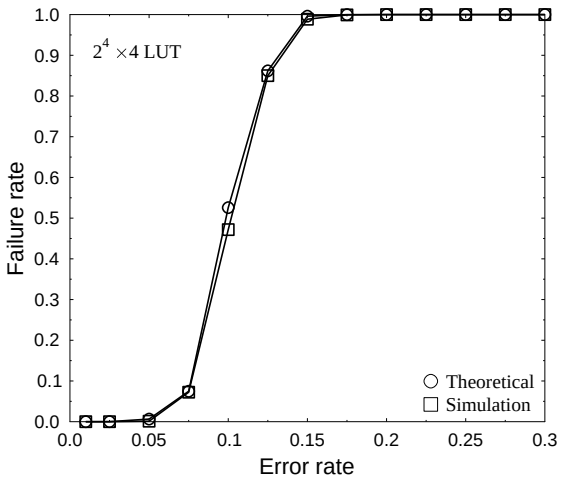


Fig. 8. 2D Coding - Failure rate obtained through simulation and theory

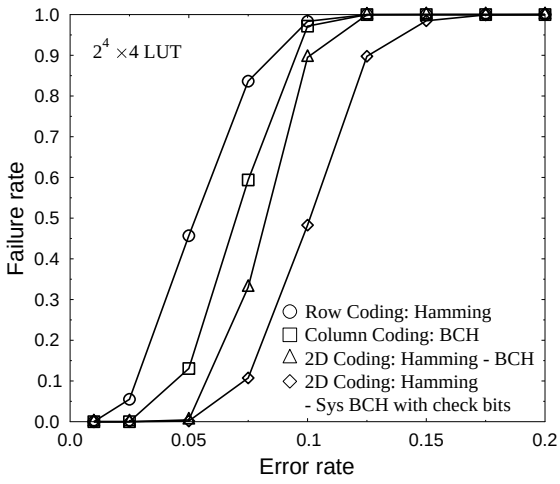


Fig. 9. Failure rate comparison betweeen 1D and 2D coding techniques

given by the following equation:

$$P_{new} = P \times P_{col} \tag{5}$$

Using this new error rate, the failure rate for each row after Hamming decoding is obtained using equation (2), as follows:

$$P_{row} = \sum_{k=2}^{r+r_{par}} \binom{r+r_{par}}{k} P_{new}^{k} (1-P_{new})^{r+r_{par}-k} \tag{6}$$

Hence, the final failure probability of our combined 2D coding technique is computed as:

$$P_{failure} = 1-(1-P_{row})^{r} \tag{7}$$

For the example used in Fig. 6, equations (4), (6) and (7) reduce to:

$$P_{col} = \sum_{k=3+1}^{16+15} \binom{16+15}{k} P^{k} (1-P)^{16+15-k}$$

$$P_{row} = \sum_{k=2}^{4+3} \binom{4+3}{k} P_{new}^{k} (1-P_{new})^{4+3-k}$$

$$P_{failure} = 1-(1-P_{row})^{16}$$

Fig. 8 shows the failure rate obtained both theoretically ($P_{failure}$) and through simulation in the case of $2^4 \times 4$ LUT. As we can see, there is an excellent correlation between experimental and theoretical results.

The plots for simulation results of the circuit failure rate for Hamming (7,4,1), BCH (31,16,3) and 2D coding techniques are shown in Fig. 9. It can be seen that for error rates smaller than 15% 2D coding technique (without check bits) exhibits better tolerance than both Hamming and BCH. For example, when the percentage of injected faults is 5%, 2D coding perfectly detects and corrects all injected faults, whereas Hamming code achieves a failure rate of approximately 45%. However, as the fault rate increases beyond 15%, this technique completely fails. This improvement in reliability is achieved at the cost of a higher number of parity bits which will result in additional area and energy overhead.

Fig. 9 also illustrates the enhancement achieved in fault tolerance by incorporating the check bits into the technique. 2D coding with check bits achieves significantly lower failure rates for error rates greater than 5% and upto 10% as compared to basic 2D coding technique resulting in an improvement of 37% in fault tolerance. As can be seen, using check bits will improve the fault-tolerance of our combined 2D coding technique. However, the need to store the number of 1's in a highly-reliable memory (i.e. store at most approximately $[log_2(num_{rows}) \times num_{columns}]$ bits in a CMOS memory) will incur an extra area and delay overhead that need to be evaluated based on practical IC designs as explained in Section IV.

Next we examine the effect of varying LUT size on circuit reliability. As can be seen in Fig. 10, the failure rate increases rapidly for bigger LUTs. For a fault density of 10%, the failure probability to successfully instantiate a LUT on the defective nanofabric increases from 5% in the case of $2^3 \times 3$ LUT to complete failure for $2^6 \times 6$ LUT. Comparing the results of Fig. 10 with Fig. 3, we can observe that combined 2D coding technique outperforms single dimensional coding in terms of fault tolerance despite its insufficiency in coping with fault rates higher than 10% and bigger LUT sizes. In the case of $2^6 \times 6$ LUT, we can observe a nearly 0% failure rate at 5% error rate, whereas Hamming completely fails.

Boolean functions are defined by their On-set, OFF-set and their DCC-set [17]. If an entry in a LUT is a DCC, the output can be either 0 or 1. We examine the impact of DCCs on our 2D coding technique. In order to theoretically calculate the circuit failure rate given the percentage of DCCs in the LUT, we first need to calculate the failure probability of the BCH decoder and the new error rate after decoding which are given by equations (4) and (5). After column decoding, the probability that the Hamming decoder fails to correctly detect and correct all errors does not only depend on the number of faults per each row, but also on the number of erroneous bits in the output of the decoder. Therefore, the probability that a given number of bits are erroneous in the output of the decoder, denoted as $P_{(n)bit}$, assuming a number of errors in the codeword has to be estimated where $n$ is smaller or equal to the number of bits of the decoded word. For a $2^4 \times 4$ LUT, the values of $P_{(n)bit}$ are shown in table I.

The failure rate of correctly decoding a row using the Hamming decoder is obtained using the following equation:

$$P'_{row} = \sum_{k=2}^{r+r_{par}} \left[ \binom{r+r_{par}}{k} P_{new}^{k} (1-P_{new})^{r+r_{par}-k} \times \sum_{n=1}^{r} \left[ P_{(n)bit} \sum_{m=1}^{n} \binom{n}{m} (1-P_{DCC})^{m} P_{DCC}^{n-m} \right] \right] \tag{8}$$

The total failure probability is computed as:

$$P_{failure} = 1 - (1 - P'_{row})^{r} \tag{9}$$

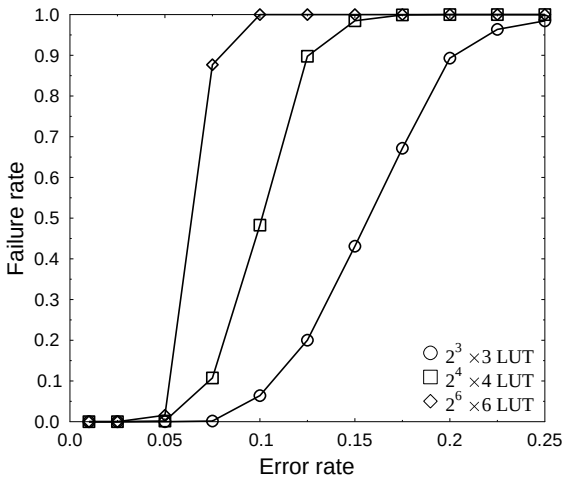


Fig. 10. 2D Coding - Effect of varying LUT size on failure rate

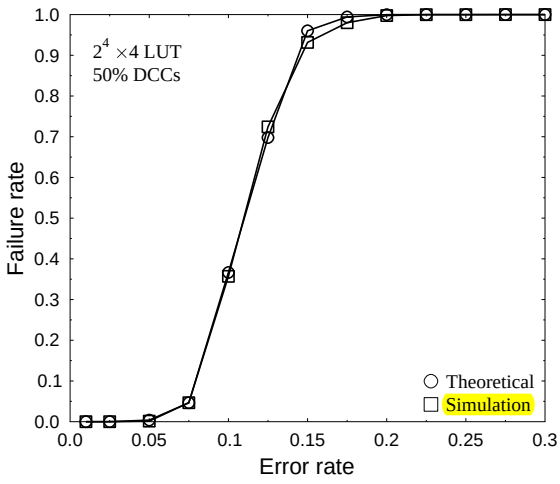


Fig. 11. 2D Coding - Failure rate obtained through theory and simulation in the presence of 50% DCCs

| $P_{(n)bit}$ | number of errors in a codeword | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | 6 | 7 |
| $P_{1bit}$ | 0.4306 | 0.1952 | 0.3639 | 0.1415 | 0 | 0 |
| $P_{2bit}$ | 0.4287 | 0.4274 | 0.4335 | 0.4271 | 0 | 0 |
| $P_{3bit}$ | 0.1407 | 0.3774 | 0.2026 | 0.4314 | 0 | 0 |
| $P_{4bit}$ | 0 | 0 | 0 | 0 | 1 | 1 |

TABLE I

HAMMING DECODER: DISTRIBUTION OF ERRONEOUS BITS IN THE OUTPUT WORD - $2^4 \times 4$ LUT

Fig. 11 presents the failure rate obtained theoretically, based on the previously outlined equations, and using the simulation procedure in the presence of 50% DCCs. The two graphs perfectly match each other which validates our theoretical predictions.

Fig. 12 shows the results obtained before and after injecting 50% of Don't Cares in $2^4 \times 4$ LUTs. As can be observed, the optimum improvement is recorded at 10% error rate where the failure rate is reduced from nearly 50% to 37%.

### *B. Hamming with Bad Line Exclusion : Technique 2*

ECCs are usually preserved for the suppression of transient faults to enhance computational reliability. However, the high defect densities in future nano-circuits [2] dictates involving coding techniques at the initial repair of hard errors to minimize the required amount of redundant resources. As can be seen from Fig. 9, ECCs alone are not

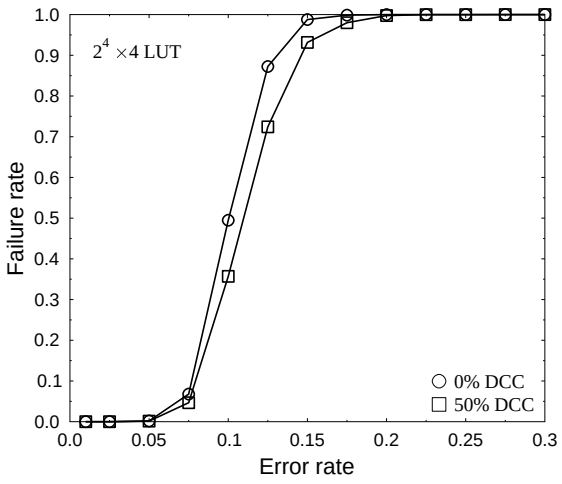


Fig. 12. 2D Coding - Effect of Don't Cares on Failure rate

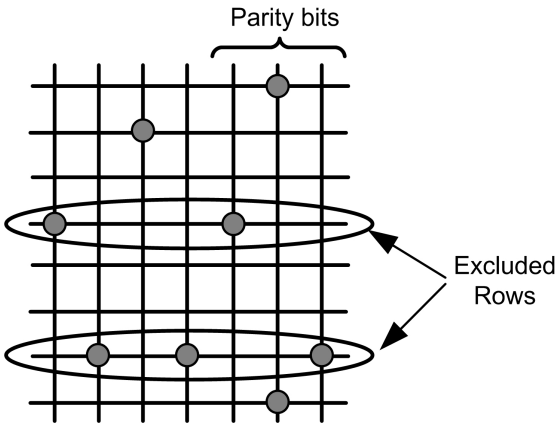


Fig. 13. Hamming with Bad Line Exclusion

able to address the issue of high defect rates induced during manufacturing. Hence it is imperative to use ECCs in conjunction with other techniques in order to detect and correct larger portions of physical defects (up to 20%).

To deal with higher defect rates, we have combined Hamming code with Bad Line Exclusion technique. This technique requires allocating enough redundant rows for each LUT to be repaired. The use of redundant wires to tolerate physical defects was presented in [16] and [21]. As will be shown, the amount of spare rows depends on two main factors: the defect rate of the fabric and the size of the LUT. Repair consists of two phases: a must-repair phase and a final-repair phase. In the must-repair phase, line exclusion is applied only in one dimension where the defective rows are excluded and replaced with spare rows if the number of defects per row exceeds the correction capability of Hamming. Therefore, defect counters for the faulty rows are required during the initial testing process of the nanofabric. In the final-repair phase, the Hamming decoder detects and corrects the remaining defects as shown in Fig. 13. During the initial analysis of the fabric, the bad rows are detected and their physical address is used to create a special table to map the continuous logical address to the actual physical location of defect-free rows. Such a mapping table has to be stored in a highly-reliable memory implemented in CMOS. The physical implementation of this logical-to-physical mapping table is beyond the scope of this work and is not included in the area overhead estimation of this technique in the next section.

To obtain the probability of failure, we first calculate the probability $P_{row}$ of a row of length $l$ being excluded. $P_{row}$ is equal to the probability of having more than one bad bit in that row given by:

$$P_{row} = \sum_{k=2}^{l} \binom{l}{k} P^k (1-P)^{l-k} \tag{10}$$

where $l$ is the number of bits in a row including the parity bits and $P$ is defect probability of each bit. Therefore, the probability of failure to instantiate a LUT on the fabric given the original number of rows $r$ and upper limit of

spare rows $r_{sp}$ can be computed as:

$$P_{failure} = \sum_{k=r_{sp}+1}^{r+r_{sp}} \binom{r+r_{sp}}{k} P_{row}^{k} (1-P_{row})^{r+r_{sp}-k} \tag{11}$$

In the case of $2^4 \times 4$ LUT and 25% spare rows, equations (8) and (9) can be rewritten as:

$$P_{row} = \sum_{k=2}^{7} \binom{7}{k} P^{k} (1-P)^{7-k}$$

$$P_{failure} = \sum_{k=4+1}^{16+4} \binom{16+4}{k} P_{row}^{k} (1-P_{row})^{16+4-k}$$

Fig. 14 shows the failure rates for a $2^4 \times 4$ LUT for different error rates in the presence of various amounts of spare rows. As can be seen, this technique is capable of tolerating an unprecedented percentage of defects when compared to the Hamming alone and combined 2D coding technique shown in Fig 9. This is demonstrated by a failure rate of nearly 0% for up to 20% of injected faults into randomly generated $2^4 \times 4$ LUT.

To further our analysis, we examine the effect of variation in number of spare rows on the failure rate for different LUT sizes. Fig. 15 demonstrates that as the error rate increases, more spares are needed to keep the level of fault tolerance close to 0%. In the case of $2^4 \times 4$ LUT for instance, only 25% of spare rows are needed i.e. 4 more rows, to completely tolerate up to 10% of faults rates. And as more errors are injected into the LUT, more spares should be allocated and hence decreasing the useful bit density of the fabric. It can also be observed that as the LUT size increases, the percentage of spares also increases to achieve low failure rates. For example, $2^6 \times 6$ LUT requires twice its original size to tolerate 20% defect rate. However, we can minimize such high redundancy by adopting a more powerful ECC such as BCH instead of Hamming.

While the authors in [17] assumed 50% don't care conditions (DCC), our results have shown remarkable improvement in fault tolerance even with zero percent DCCs in the LUT implementation. However, it can be seen from Fig. 16 that the fault tolerance of this technique is significantly improved when we assume some of the entries in the implementation as DCCs as compared to our results in Fig. 14.

The existence of DCCs ($P_{DCC}$) in LUTs significantly reduce the bit failure rate as outlined in the following equation:

$$P' = P \times (1 - P_{DCC}) \tag{12}$$

The new probability of a row being excluded after injecting Don't Cares is obtained by replacing the fault rate $P$ with the new fault rate $P'$ in equation (8) as follows:

$$P'_{row} = \sum_{k=2}^{l} \binom{l}{k} P'^{k} (1-P')^{l-k} \tag{13}$$

Fig. 17 illustrates the failure rate obtained both theoretically using equations 11 and 13 and through simulations. As can be observed, there an identical match between the two graphs indicating the correctness of our derived mathematical equations.

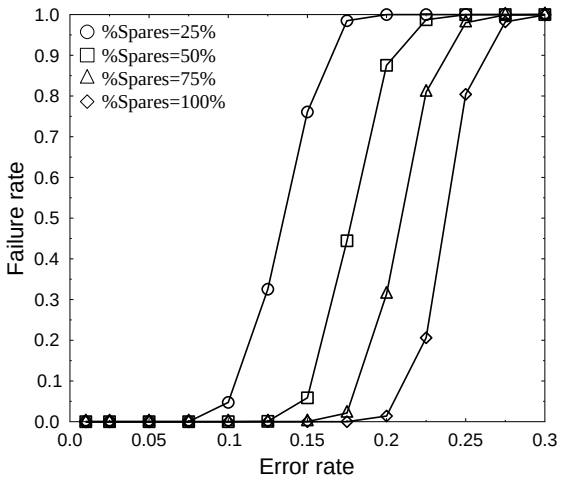


Fig. 14. Hamming & Bad Line Exclusion - Failure rate obtained for different percentages of spare rows-$2^4 \times 4$ LUT

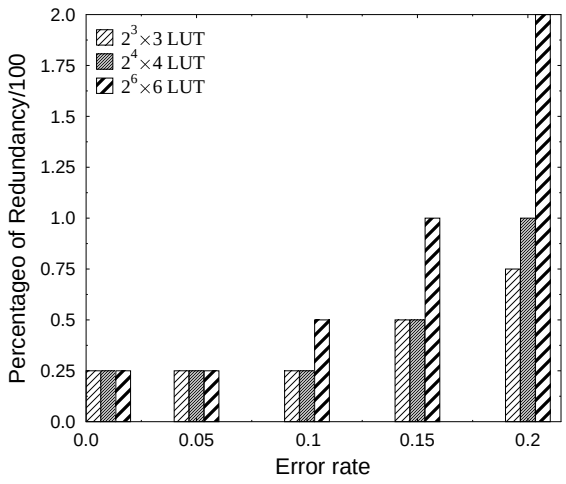


Fig. 15. Percentage of Redundancy needed to achieve 0% failure rate for different LUT sizes

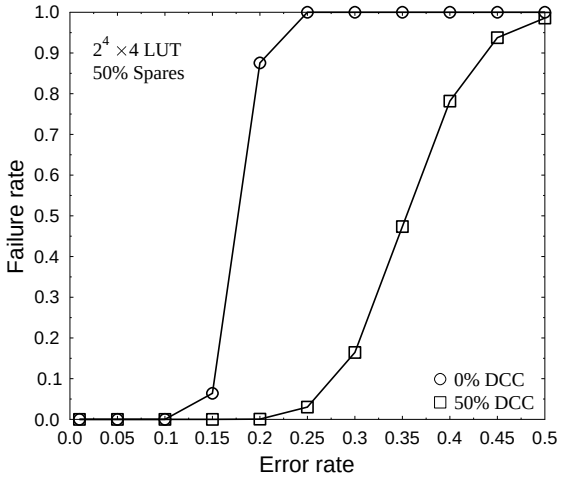


Fig. 16. Hamming & Bad Line Exclusion - Variation of Failure rate in the presence of Don't Care entries. Inclusion of 50% DCCs improves fault tolerance by almost two times.

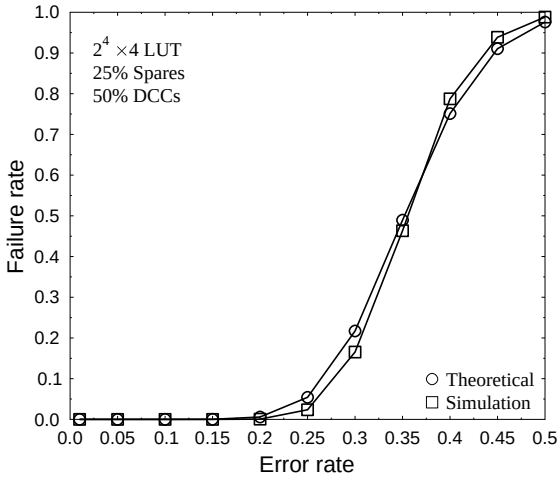


Fig. 17. Hamming & Bad Line Exclusion - Failure rate obtained through theory and simulation in the presence of 25% spares and 50% DCCs

## IV. Implementation

The realisation of fault tolerance in nano/CMOS nanoelectronic architecture will incur area, energy and operational latency overhead in CMOS domain [15]. Such overhead must be taken into account when investigating and evaluating hybrid CMOS/nanodevice fault tolerant architectures. The high reliability decoders are implemented in CMOS and

therefore incur an increase in the area and energy consumption compared to the denser and low energy nano-LUTs. Additional clock cycles are also lost in decoding and correcting codewords which cause latency overhead.

In order to obtain an estimate of the area, latency and energy overheads per codeword, the corresponding decoders were designed in VHDL and thoroughly tested through simulation using the appropriate test benches. Both Hamming and BCH decoders are serial i.e. they receive 1-bit input and generate a 1-bit output per clock cycle, therefore, the decoding latency is proportional to the codeword length. Using the $0.12\mu m$ CMOS standard cell library, the area overhead of the Hamming decoder is $906\mu m^2$ and decoding one 7-bit long codeword requires 13 clock cycles and an energy overhead of approximately $9pJ/MHz$. The BCH decoder incurs higher overhead due to its high complexity: an area overhead of $9132\mu m^2$, a latency of 69 clock cycles and an energy overhead of $286pJ/MHz$. However, this need to be further studied in the context of optimum design strategy where we negotiate the advantage of higher density and low power dissipation against increased delay overhead.

The CMOS components as well as the redundant parity bits and spare rows in our techniques will reduce the useful bit density offered by nanodevices. Therefore, another design parameter called *area per useful bit* ratio is used to compare the efficiency of the various techniques. Area per useful bit $(a)$ reflects the area necessary to achieve certain useful bit capacity and is obtained by dividing the total area of the fabric by the number of useful bits in the LUT.

The total area of the fabric comprises of the area of nanodevices and that of CMOS subsystems. We adopt a model presented in [14] to estimate the area of nanoscale memory. Each bank in the memory is composed of a set of crossed nanoscale wires supported by a set of interface microscale wires. For a nano-circuit of $2^n$ inputs and $m$ outputs, the area can be estimated as shown in [17]:

$$A = \left(W_{lith}(n + log_2 m) + W_{nano}2^n\right).\left(W_{lith}n + mW_{nano}2^n\right) \tag{14}$$

The main parameters in the model are the number of rows $2^n$ and columns $m$. The area of the nano-memory is dominated by the address lines which are microwires. $W_{lith} = 105nm$ is the wire pitch of the lithographic address wires and $W_{nano} = 10nm$ is the pitch for the nanoscale wires. For instance, in the case of a $2^4 \times 4$ LUT, the area of the nano-LUT before encoding is $0.84\mu m^2$.

| Technique | Total Area ($\mu m^2$) | Area/Useful bit ($\mu m^2/bit$) |
|---|---|---|
| Hamming alone [17] | 907 | 14 |
| BCH alone [13] | 9133 | 142 |
| Hamming & Bad Line Excl. (proposed) | 908 | 14 |

TABLE II

AREA/USEFUL BIT OF THE PROPOSED TECHNIQUES

Table II compares the area overhead of the proposed technique with the earlier approaches. While we have seen that Hamming in combination with bad line exclusion achieves much better failure rates as compared to error correcting schemes such as Hamming or BCH, this improvement in failure rate is achieved with little or no increase in area overhead when compared with the simplest correction technique proposed in [17]. It should be noted that BCH has the highest area overhead due to the complexity of its decoding circuitry. Table II also shows the area per useful bit ratio for a $2^4 \times 4$ LUT implemented using the three techniques.

While significant area improvement can be achieved over current CMOS for high density fabrics using hybrid nano/CMOS architecture [13], it can be shown that further improvement in terms of useful bit density can be achieved by sharing the decoders by multiple LUTs using time multiplexing strategy as outlined in [17]. Another way to minimise the CMOS area overhead is to synthesise logical circuits into smaller LUTs because the size of the decoder increases proportionally with the size of the LUT. Moreover, as shown in the experimental results (Fig. 10 and Fig. 15), using smaller LUTs allows achieving higher levels of fault tolerance at the cost of low area overhead.

## V. Conclusion

In this paper we investigated a promising look-up table based implementation of Boolean logic functions in heterogeneous CMOS/nanodevice architectures. Our studies showed that single error correcting codes such as Hamming or BCH prove their insufficiency in tolerating high error rates. We presented two hybrid fault-tolerance techniques that address faults caused due to physical defects and transient faults. In the first technique, we encoded both rows and columns of LUTs (using Hamming and BCH codes respectively) to target higher number of faults. This technique significantly improves the fault tolerance with respect to single error correction schemes for error rates greater than 5%. In the second technique, we complemented Hamming with bad line exclusion. This technique results in remarkable improvement of failure rate against a substantial fraction of bad nanodevices (up to 20%). This is achieved at the cost of minimal increase in area overhead compared with Hamming, yet with much higher efficiency in tolerating errors. Based on our studies this technique is very effective for LUT-based Boolean logic architectures. We have also shown that the presence of don't care conditions in LUTs can significantly improve circuit failure rates when we combine coding with bad line exclusion. Finally, we investigated the impact of these techniques in terms of area, latency and energy overheads and showed that improved fault tolerance can be achieved using the proposed techniques with little overheads compared to previous coding techniques.

## VI. Acknowledgement

The authors would like to acknowledge the EPSRC (UK) for funding this project in part under grant EP/E035965/1 as well as the Algerian Ministry of Higher Education and Scientific Research.

## References

[1] J. G. Brown and R. D. Blanton. A Built-in Self-test and Diagnosis Strategy for Chemically Assembled Electronic Nanotechnology. *J. Electron. Test.*, 23(2-3):131–144, 2007.

[2] Z. Wang and K. Chakrabarty. Built-in Self-test and Defect Tolerance in Molecular Electronics-based Nanofabrics. *J. Electron. Test.*, 23(2-3):145–161, 2007.
[3] R. I. Bahar. Trends and Future Directions in Nano Structure Based Computing and Fabrication. *International Conference on Computer Design, 2006.*, pages 522–527, Oct. 2006.
[4] M. Jacorne, C. He, G. de Veciana, and S. Bijansky. Defect tolerant probabilistic design paradigm for nanotechnologies. *DAC '04. Proceedings. 41st*, pages 596–601, 2004.
[5] C. Zhao, S. Dey, and X. Bai. Soft-spot analysis: targeting compound noise effects in nanometer circuits. *Design & Test of Computers, IEEE*, 22(4):362–375, July-Aug. 2005.
[6] K. Nepal, R. I. Bahar, J. Mundy, W. Patterson, and A. Zaslavsky. MRF Reinforcer: A Probabilistic Element for Space Redundancy in Nanoscale Circuits. *Micro, IEEE*, 26(5):19–27, Sept.-Oct. 2006.
[7] K. Nikolic, A. Sadek, and M. Forshaw. Fault-tolerant techniques for nanocomputers. *Nanotech.*, 13(3):357–362, 2002.
[8] D. Thaker, R. Amirtharajah, F. Impens, I. Chuang, and F. Chong. Recursive TMR: scaling fault tolerance in the nanoscale era. *Design & Test of Computers, IEEE*, 22(4):298–305, July-Aug. 2005.
[9] M. Mishra and S. Goldstein. Defect tolerance at the end of the roadmap. *In ITC*, 1:1201–1210, 30-Oct. 2, 2003.
[10] C. He, M. Jacome, and G. de Veciana. A reconfiguration-based defect-tolerant design paradigm for nanotechnologies. *IEEE Design & Test*, 22(4):316–326, July-Aug. 2005.
[11] S. Copen Goldstein and M. Budiu. NanoFabrics: spatial computing using molecular electronics. *IEEE ISCA*, pages 178–189, 2001.
[12] C. Jeffery, A. Basagalar, and R. Figueiredo. Dynamic sparing and error correction techniques for fault tolerance in nanoscale memory structures. *Nanotechnology, 2004. 4th IEEE Conference on*, pages 168–170, Aug. 2004.
[13] F. Sun and T. Zhang. Defect and Transient Fault-Tolerant System Design for Hybrid CMOS/Nanodevice Digital Memories. *Nanotech.*, 6(3):341–351, 2007.
[14] A. DeHon, S. Goldstein, P. Kuekes, and P. Lincoln. Nonphotolithographic nanoscale memory density prospects. *Nanotechnology, IEEE Transactions on*, 4(2):215–228, March 2005.
[15] R. I. Bahar, D. Hammerstrom, J. Harlow, W. H. J. Jr., C. Lau, D. Marculescu, A. Orailoglu, and M. Pedram. Architectures for Silicon Nanoelectronics and Beyond. *Computer*, 40(1):25–33, 2007.
[16] D. B. Strukov and K. K. Likharev. Prospects for terabit-scale nanoelectronic memories. *Nanotech.*, 16(1):137–148, 2005.
[17] A. Singh, H. Zeineddine, A. Aziz, S. Vishwanath, and M. Orshansky. A heterogeneous CMOS-CNT architecture utilizing novel coding of boolean functions. *NANOARCH 07*, pages 15–20, Oct. 2007.
[18] N. R. Shanbhag, S. Mitra, G. de Veciana, M. Orshansky, R. Marculescu, J. Roychowdhury, D. Jones, and J. M. Rabaey. The Search for Alternative Computational Paradigms. *IEEE Design and Test of Computers*, 25(4):334–343, 2008.
[19] M. Ziegler and M. Stan. CMOS/nano co-design for crossbar-based molecular electronic systems. *Nanotechnology, IEEE Transactions on*, 2(4):217–230, Dec. 2003.
[20] T. Mizuochi, Y. Miyata, T. Kobayashi, K. Ouchi, K. Kuno, K. Kubo, K. Shimizu, H. Tagami, H. Yoshida, H. Fujita, M. Akita, and K. Motoshima. Forward error correction based on block turbo code with 3-bit soft decision for 10-Gb/s optical communication systems. *Selected Topics in Quantum Electronics, IEEE Journal of*, 10(2):376–386, March-April 2004.
[21] T. Lehtonen, P. Liljeberg, and J. Plosila. Online Reconfigurable Self-Timed Links for Fault Tolerant NoC. *VLSI Design*, 2007:13, 2007.

## VII. Response to the Reviewers

We would like to thank you all for spending your valuable time in conducting the review of our manuscript. We also thank you for your valuable and constructive suggestions to enhance the quality of this paper.

### A. Detailed Response

*1) Comments from Reviewer 1: This paper addresses an important topic related to the problem of fault tolerance in high-density digital electronic circuits. The main results related to the error-correction and bad line exclusion techniques seem to be valid, but the beginning of the paper is presented in a confusing way.*

- *It starts with the title, which mentions an architecture of some sort. Nanoelectronics (so far) does not belong to the mainstream VLSI design, so one would expect to see a definition of the object under test in the introduction to the paper, but there is only a reference to [17].*

  Our Response: To clarify this, a short description of the targeted CMOS/Nano architecture is included in the introduction (page 2, first three sentences starting with "The prohibitively low... critical circuit operations"). We have also added the definition of the object under test (which is LUTs) in the introduction (page 2, paragraph 3, sentences 4-5). Another reference (ref. [19]) was also added to further explain how the highly dense nanoscale fabric can be used to implement Boolean logic circuits as look-up tables.
- *The reference architecture is based on CMOS/carbon nanotubes fabric, but the latter are not mentioned in the paper under review.*

  Our Response: In this work, the targeted nanofabric is technology-independent. In [17], the authors targeted CMOS/CNT fabric, whereas our proposed techniques are designed for nanoscale devices in general. This has been clarified in the introduction section (page 2, paragraph 3, sentence 2).
- *It is unclear whether the architecture in question is Figure 1 from [17] (includes CF0..3, MUX, DEMUX, Scheduler, Decoder, FF-s) or a way the CNT combinational logic blocks (CF0..3) are mapped into the matrices. The further discussion is focused on the matrices alone.*

  Our Response: To clarify the architecture in question, we have introduced a new figure called "Hybrid Nano/CMOS Architecture Overview" (see Fig. 1). A description of how the LUTs are constructed using matrices is included in the introduction (page 2, paragraph 3, sentence 5).
- *Section 3 describes the proposed techniques. Only in this section it becomes clear how the matrices are constructed. This should be done in the introduction, where the object under test is defined.*

  Our Response: We thank the reviewer for pointing this out. We have clarified this in the introduction (page 2, paragraph 3, sentence 5).
- *It is unclear why the diagrams in Figures 7..11 do not show the case of a large LUT, as in Section 2.*

  Our Response: We have now ensured consistency throughout the paper by using LUTs of sizes of upto $2^6 \times 6$ when evaluating the proposed techniques in Fig. 3, Fig. 10 and Fig. 15.
- *In Section 4 time, energy and area overheads caused by the decoders are calculated. The encoders are not mentioned probably because they are not shown in Figure 1 in [17]. The other parts of the reference architecture*

*are also not mentioned.*

Our Response: The reviewer is right, the encoders are not included in the overhead estimations because they are not included in our targeted architecture (please see Fig. 1). The reason is that they are not included in the final layout of the circuit. After partitioning the logic circuit into *N* small LUTs (CF0...*N*), the encoder is used to encode their data entries (offline) before being written to the nanofabric.

- *The overheads are given as absolute values and not compared to the nanotube implementation of the combinational logic blocks.*

  Our Response: To clarify this, we have added a statement to compare the area of nano-LUT with the area of the CMOS decoders in (page 11, section ...). Moreover, equation 14 can be used to estimate the area of nano-LUTs.

- *Minor faults: the second sentence in the abstract is confusing, make it clear that the references in the first paragraph are specific to nanotechnology.*

  Our Response: The second sentence of the abstract has been revised.

- *To summarise the paper can be improved by expanding the introduction and by making it clear that the main idea is to deal with LUTs, the latter aspect being technology-independent.*

  Our Response: We thank the reviewer for his/her suggestion. We have made it clear in the introduction (page 2, paragraph 3, sentence 8) that the main idea is to tolerate faults in LUTs. We have also added another sentence in the introduction (page 2, paragraph 3, sentence 2) to highlight the fact that the targeted nanoscale fabric is technology-independent.

*2) Comments from Reviewer 2: The use of error correcting codes to improve circuit reliability is not new. The work is incremental, but interesting, as it tests a few codes and strategies. The work is nicely presented, however it needs to clarify some points.*

- *The use of crossbar type circuits have been discussed extensively by K. Lkharev, whose work is not discussed properly in the paper, except for ref. [16]*

  Our Response: We thank the reviewer for pointing this out. We have cited ref.[16] in the introduction section (page 2, paragraph 2, sentence 7).

- *The paper assumes a uniform defect distribution to apply an error correcting code to make the circuit more robust. This is of limited application in the scenario with stronger correlated defects, which is not considered in equation 1. This should be made clear in the text.*

  Our Response: The reviewer is right to emphasise on the type of error distribution considered in our simulation and theoretical analysis. Because our targeted fabric is technology-independent, we can not model accurately the distribution of correlated defects without targeting a particular technology. We also believe that random distribution of faults represent the worst case scenario and hence the lower bound of the targeted fault rates. This has been explicitly clarified in the Primitives section (page 3, paragraph 3, sentence 5) that the simulations are based on randomly distributed errors.

- *In Figure 1, the plot labels says theoretical and simulation, while the caption states theoretical and experimental! The authors mixes up experiment with simulation. Simulation is not experiment.*

  Our Response: We thank the reviewer for pointing this out. All captions have been rewritten.
- *VHDL can only confirm the authors claim, as it does not grasp correlations of the nano world. In this sense this work is not specific for nanocircuits, it could be used for any circuit.*

  Our Response: We agree with the reviewer that VHDL is a high level description language and hence is not suitable for modeling nanocircuits. However, since we have presented a system-level model for a hybrid CMOS/Nano architecture, we use C++ estimate the fault tolerance exhibited by various techniques. VHDL is only used to present an estimate of CMOS area, latency and energy overheads.
- *As the error rate increases, there is an increase in area penalty to include the parity bits. Considering the area overhead, is it profitable to make nanosize circuits?*

  Our Response: Fault tolerance incurs area overhead through redundancy. We believe that the proposed fault tolerance techniques have moderate area overhead cost (see page 10, section 4, paragraph 2 and table 2) and effective in tolerating high error rates (see Fig. 9 and Fig. 14).
- *For a given failure rate, what is the minimum bit area without any code, and the minimum area with code. Is there a gain?*

  Our Response: We are unclear about this question. Table II provides information about *area/useful bit* ratio for the various techniques in the case of $2^4 \times 4$ LUT. We note here that the proposed techniques only achieve a gain in terms of reliability at the cost of increased area overhead.

*3) Comments from Reviewer 3: The paper presents two fault tolerant schemes for reliability. In the first scheme, two-dimensional coding using Hamming and BCH codes are used. In the second, Hamming codes are combined with a bad line exclusion technique. Here are some comments for the authors to improve the paper...*

- *Use of 2-D codes and spare units is not completely novel (e.g. they've been used in fault tolerant memories). The authors need to show how their approach is different and compare with previous related work.*

  Our Response: We agree with the reviewer that coding schemes and repair techniques that are based on spare units have already been used in memory designs. However, to the best of our knowledge, there are no other reported work targeting fault tolerance in CMOS/Nano LUT-based architecture which is the focus of this work. More importantly, the existence of don't care conditions in LUTs helps increase the efficiency of these techniques in building highly reliable systems at a reduced cost which is a key contribution of this paper (see Fig. 16).
- *Moreover, the purpose for these proposed techniques is not clear. In the introduction section (page 2), you will address the permanent and transient faults. However, in the conclusion section (page 14), you switch the target into faults introduced during manufacturing process, which are more likely permanent errors. Please clarify this.*

  Our Response: We have ensured consistency between the introduction and conclusion. The conclusion have

been revised taking into account that both physical and transient faults are addressed in this paper (page 15, Conclusion section, sentence 3).

- *One of the main problems in this paper is unfair comparisons. For example, in Fig. 4, Fig. 7 and Fig. 8, neither the information bit width nor code rate of the compared codes is equal. Please modify the experiment to show how to compare them in a fair way?*

  Our Response: The reviewer is correct to point out that the information bit width and the code rate of the compared codes are not equal. However, in Figs 5 and 9, the various coding schemes are compared in terms of their ability in detecting and correcting defective bits. While all the various coding schemes target the same LUT size ($2^4 \times 4$ in Fig. 5 and 9), this difference between them in the information bit width and code rate is due to the direction of coding as well as the number of parity bits associated with the code (which can be termed as the cost of correction). While the comparison can be termed as unfair, it highlights the cost of correction to achieve higher levels of reliability.

- *Another problem is missing very important experimental results. The technique 1, i.e. 2D error coding, has not been well evaluated, although you provide some latency analysis, absolute value for Hamming and BCH codec area and energy. There is no comparison to to other existing related approach, e.g. Ref. 17.*

  Our Response: We believe that we have adequately evaluated the 2D coding technique. In Fig. 9, we compared the level of reliability achieved by this technique with the previously proposed techniques. We also investigated the impact the varying the LUT size on its failure rate (see Fig. 10). This is in addition to the estimated area, latency and energy overheads in section 4 (page 14, paragraph 2). Because this technique uses both Hamming and BCH decoders, its area, latency and energy overheads are the sum of the overheads of both Hamming( [17]) and BCH ( [13]) decoders.

- *Because you add too much redundancy on the information (your code rate is 16/(7*31)), how could you win in terms of area and energy?*

  Our Response: It is correct that more redundancy is added in 2D coding technique (code rate of 16/(7*31)) as compared to Hamming. Because the aim is to target higher error rates in the nanofabric (see Fig. 9), extra parity bits and stronger decoders are needed to achieve such improvement. Therefore, this improvement in circuit's yield and reliability is obtained at the cost of more area and energy overheads. This has been clarified in (page 9, section 3(A), paragraph 1, last sentence). Moreover, the redundancy overhead, due to encoding in both dimensions, is exploited efficiently to improve the targeted error rate. As shown in Fig. 5, coding in one dimension is incapable of improving the targeted fault rate even if stronger coding schemes are used. For instance, although the padding technique adds more redundant bits (code rate of 16/(15*16)) than 2D coding approach (code rate of 16/(7*31)), this technique is unable to tolerate error rates higher than 1% as shown in Fig. 5.

- *In addition, the experiment results for technique 2 are also not complete. You only provide the area number. How about the energy?*

  Our Response: We have provided energy values for CMOS Hamming decoder in (page 14, section 4, paragraph

2). The energy overhead in nanoscale domain has not been included because our targeted nanofabric is technology-independent.

- *Plus, the total area for the proposed method is doubtable. How could that be possible that you only introduce 1 $\mu m^2$ overhead than Hamming code? It is difficult to comprehend how the use of Hamming codes with bad line exclusion can have almost the same area as Hamming codes alone (Table 2).*

  Our Response: As mentioned in section 4, the area of the fabric comprises of the area of the nanodevices and that of CMOS subsystems (page 14, paragraph 3, sentence 1). The difference between Hamming [17] and the proposed Hamming and Bad Line Exclusion technique is the number of spare rows allocated for repair. Assuming a 100% redundancy, the total area added to the original LUT area and Hamming decoder area is equal to $0.84\mu m^2$ and hence an increase of approximately $1\mu m^2$ in total area with respect to Hamming technique. This represents $< 0.1\%$ of the total area.
- *Do the authors include the area overhead of mapping low row address to the physical row address in the proposed method? Please address that.*

  Our Response: We have not included the area overhead of mapping logical row address to the physical row address in the physical implementation. We have mentioned this in section 3(B) (page 11, paragraph 2, last sentence).
- *The description for encoding and decoding process is incomplete. You briefly mention multiple-step decoding process in page 6. Please provide more details in flow chart or algorithm description, as well as the explanation for why you perform BCH decoding first and Hamming decoding later.*

  Our Response: We have added a new flow chart (see Fig. 7) to describe how the output is read from the encoded LUT, as requested by the reviewer. We have also included a small paragraph in section 3(A) (page 6, paragraph 1, sentences 4-7) to explain how the data is retrieved from the encoded LUT and why the BCH decoding is performed first.
- *The description for spare wire-related work in technique 2 is missing as well. Do you add spare wire on row or on column, how do you decide on the optimal number for the spare wires, etc?*

  Our Response: In technique 2, redundancy is added row-wise. Equations 10 and 11 can be used to evaluate the optimal number of spare rows $r_{sp}$ to tolerate a given defect rate $P$ and LUT size.
- *Missing references: Product code is a 2D code, please refer to that.*

  Our Response: We have added a reference to *product code* technique in section 3(A) (page 6, paragraph 1 , sentence 1).
- *Missing references: Using spare wire to tolerate permanent error has been addressed in Lehtonen, T., Liljeberg, P., and Plosila, J.: Online reconfigurable self-timed links for fault tolerant NoC, VLSI Design, 2007, Article ID 94676, p. 13*

  Our Response: We have added the paper mentioned above to our list of references and to section 3.2 (page 11, paragraph 2, sentence 3).
- *Equation (5) is confusing. Why is the new defect probability the product of the residual error rate of BCH*

*codes and the original defect probability?*

Our Response: We have revised the sentence that precedes equation 5 (page 8, last 2 lines) to clarify this. Assuming a uniform distribution of errors in the LUT, after BCH decoding of columns, the probability that any bit in a column remains erroneous is equal to the product of the probability of the bit being erroneous $P$ and the probability that the BCH decoder fails to correct it which is $P_{col}$.