Robust Correction of Repeating Errors in by Nonlinear Codes

Konrad J. Kulikowski, Mark G. Karpovsky

Abstract

For many devices and circuits even single faults can result in errors of a large Hamming weight at the output of a device complicating the task of error detection and correction. Traditional error detection and correction techniques are based on classical linear error detecting codes that are designed to detect or correct errors of a small multiplicity or burst errors. We present codes and architectures suitable for both on-line and off-line error detection and correction which can detect and correct *any error of any multiplicity* by exploiting the laziness of error manifestation. Faults in many combinational circuits can lead to "lazy" or repeating errors at the outputs of a device to be protected. The proposed decoding algorithms can correct any error of any Hamming weight if the same error affects at least three different consecutive distorted output vectors of a protected device for binary algebraic errors. Likewise, the proposed decoding algorithms can correct any error of any Hamming weight if the same error affects two different output vectors of a protected device for arithmetic errors.

I. Introduction

Faults in combinational circuits can be problematic for traditional error detection methods based on classical linear codes. Single faults in combinational circuits can lead to errors of a large Hamming weight at the output of a device which cannot be corrected by the linear codes used for protection [1].

Classical linear error detecting codes typically used in hardware devices (i.e. Hamming, parity, BCH) concentrate their error detecting power on errors of a small multiplicity and are characterized by their minimum distance d. Codes with a minimum distance d are guaranteed to detect all errors of Hamming weight less than d or correct all errors with Hamming weight less or equal to $\lceil (d-1)/2 \rceil$. Errors with a larger Hamming weight are not guaranteed to be detected and can often be miscorrected. Codes used for concurrent error detection or correction in hardware devices are typically limited to a small minimum distance codes (typically $d = 2, 3$ or 4 for most implementations).

Due to the possible large Hamming weight of output errors in combinational circuits resulting from faults in the internal components, techniques based on traditional designs are limited in their error correction ability, often resulting in more data corruption than correction. While a redesign of a combinational circuit to minimize the Hamming weight of errors due to faults is possible, such techniques can be costly in hardware and are not effective when multiple faults are considered [2].

The authors are with the Department of Electrical and Computer Engineering, Boston University, Boston, MA, 02215. {konkul,markkar}@bu.edu

In this paper we present codes capable of detecting and/or correcting any error, of any multiplicity by exploiting the laziness of errors resulting from faults within combination circuits. The codes can correct any error provided that the error repeats for at least two or three outputs. They can be applied for correction of algebraic and arithmetic errors and are efficient when errors have a high "laziness" or a high probability of repeating. Such lazy errors can be found in many combinational circuits for various fault models. The constructions of the error correcting codes used in this paper are based on systematic robust error detecting codes [3] [4] and have efficient encoding and decoding procedures and specific assumptions on error distributions are required for the proposed error correction method.

The rest of the paper is organized as follows. First we define robust error correction and discuss lazy errors. General constructions of robust error correcting codes are discussed next and we show that constructions of previously known robust error detecting codes also results in robust error correcting codes. We next show that these codes have efficient decoding algorithms. Finally, we present two case studies and applications followed by conclusions.

II. Robust Error Correcting Codes

R-robust error-correcting codes are codes which can correct *any* error if the same error affects at-least R different codewords. Let $\mathbf{A} = (a_1, a_2, ..., a_M)$ be an ordered set of M q-ary n-dimensional vectors, and $\mathbf{A}_e$ be the *translate* of $\mathbf{A}$ by an element e, $\mathbf{A}_e := (a_1 + e, a_2 + e, \ldots, a_M + e)$.

Definition 1. ***(Robust Error-Correcting Code (R-RECC))*** *Let $C \subset GF(q^n)$ be a code and let $\mathbf{A}$ be an ordered set whose elements are a subset of C where $N = |\mathbf{A}|$. If any ordered set $\mathbf{A}$, where $N \geq R$ can be uniquely determined from $\mathbf{A}_e$ (knowing C) for any $e \in GF(q^n)$ then C is a **R-robust error-correcting code** ($GF(q^n)$ is the field of q-ary n-dimensional vectors where q is a power of a prime). A R-robust error-correcting code is called **strong** if the order of the elements in the original set **A** can be correctly determined and is called **weak** if the elements of **A** but not their order can be determined.*

We will say that a code $C \subset GF(q^n)$ with $|C| = M$ is a $(n, M)_q$ code.

Remark 1. *As we will see later a code can be a weak R-robust error-correcting code but a strong $(R+1)$-robust error-correcting code.*

Example 1. *Consider a binary code $C \subset GF(2^6)$, $C = \{000000, 001001, 010011, 011100, 100101, 101110, 110111, 111010\}$. Any pair of codewords has a unique difference (componentwise XOR) which can be used to identify the pair. The difference of the elements $A = \{000000, 001001\}$ for example is 001001. It can be verified that no other two codewords of C have the same difference. This difference or signature of the two codewords remains constant even if set A is translated by an error and hence the error can be corrected since the original set of codewords can be determined. Code C is a weak 2-robust error correcting code.*

Robust error correcting codes can correct any error if several different codewords are distorted by the same error. To be effective, such codes require repeating errors or errors with a high laziness.

III. Laziness of Errors

Definition 2. ***(Laziness)*** *The* ***laziness*** $L(e)$ *of an error* e *in a digital device is the conditional probability that if an erroneous output* $\tilde{w}_i$ *was a result of an error* e *on the expected output* w_i*, the next erroneous output* $\tilde{w}_{i+1}$ *was also the result of the same error on the expected output* w_{i+1}*.*

$$L(e) = Pr(e = \tilde{w}_{i+1} - w_{i+1}; e = \tilde{w}_i - w_i), \tag{1}$$

where $w_i \neq w_{i+1}, e \neq 0$*.*

Permanent stuck-at faults in linear networks consisting of XOR gates or fanout-free logic implementations will result in internal faults manifesting themselves as repeating errors at the outputs of the devices and result in laziness of one or close to one. Failures in interconnect networks such as busses also result in repeating errors since faults can directly manifest themselves as errors. For such devices a single fault has a very high probability of manifesting itself in a constant error pattern regardless of the data it distorts. Errors with $L(e) \cong 1$ can also be a manifestations of transient faults. For fine-grained asynchronous circuits, for example, transient faults can result in a stream of erroneous data distorted by errors with $L(e) = 1$ [5].

Errors with high laziness can also occur in cryptographic devices where an adversary is the cause of the malfunctions. It has been shown that the erroneous outputs resulting from faults in cryptographic devices can be used for cryptanalysis [6]. The fault injection methods of an attacker are typically much slower than the operation of the device often causing a single fault to affect several cycles of data. The slow fault injection mechanism can result in several outputs of the device to be distorted by the same error. The devices where errors have a high laziness can be viewed as a special case of channels with memory [7].

It is important to note that the laziness of errors does not imply a continuous error. An error might only repeat when the fault generating the error manifests itself at the outputs of the device, but the manifestation of an error can be separated by long blocks of non-erroneous data.

Example 2. ***(Laziness of errors in combinational networks from single stuck-at faults)***

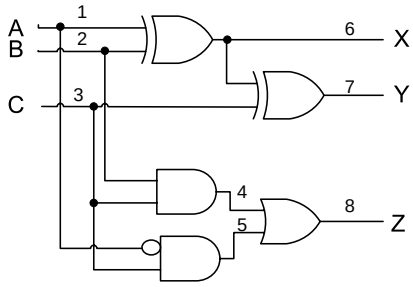

Fig. 1. A nonlinear circuit with lazy errors from stuck-at faults.

TABLE I

LAZINESS OF ERRORS $L(e)$ FOR CIRCUIT FROM FIGURE 1

	Laziness of error e=						
fault location	001	010	011	100	101	110	111
1	-	-	-	-	-	0.333	0.333
2	-	-	-	-	-	0.333	0.333
3	-	0.333	0.333	-	-	-	-
4	1	-	-	-	-	-	-
5	1	-	-	-	-	-	-
6	-	-	-	-	-	1	-
7	-	1	-	-	-	-	-
8	1	-	-	-	-	-	-
total $L(e)$	1	0.666	0.333	-	-	0.555	0.333

Consider the circuit in Figure 1. In the presence of single stuck-at-zero faults the errors at the output of the circuit can exhibit high laziness and the errors are not limited to single bit distortions. The calculated laziness of all the errors in the presence of single stuck-at-zero faults at each of the possible eight locations is shown in Table I where all inputs to the circuit are assumed equiprobable. The first eight rows in Table I show the $L(e)$ *for each error when a single stuck-at-zero fault affects the corresponding location in the circuit. For each fault location some errors exhibit very high laziness while other errors do not appear for the specific fault (represented by a dash in the table). For example, for a stuck-at-zero fault at location 1 and input* $(ABC) = (100)$ *we have* $e = (110)$*. Out of remaining 3 inputs* (101)*,* (110) *and* (111) *only one, namely* (110) *results in the same error* (110)*. Thus the laziness of* (110) *for this fault is 0.333. The final row shows the laziness of each error if any of the single stuck-at-zero faults and all input vectors are equiprobable. When any of the single stuck-at-zero faults can occur, for example, if the error* 110 *is observed at the output there is a* $\frac{1}{3}(1 + 0.33 + 0.33) = 55.5\%$ *probability that the next erroneous output will be distorted by the same error.*

As the example motivates, there can be many instances for which faults can result in errors of a large Hamming weight making them uncorrectable by classical minimum distance codes or by using several copies of the device. These errors, however, can have a high laziness. Although the example above used a small circuit of the analysis of laziness, as we will shown with a case study in Section VIII, large laziness of errors can also be observed in larger nonlinear circuits. This laziness of errors can be exploited to improve the error correction properties with the use of *robust* error correcting codes.

IV. CONSTRUCTIONS

A. General Design

The *multiset of differences* of the ordered set $\mathbf{A}$ is the ordered set of pairwise differences of all the elements $\triangle\mathbf{A} := (a_i - a_j; a_i, a_j \in A, a_i \neq a_j)$. For a set $\mathbf{A}$ with elements in $GF(q^n)$

$$\triangle\mathbf{A} = \triangle\mathbf{A}_e \tag{2}$$

for any e. The ordered multisets of differences of an ordered set $\mathbf{A}$ are the same for all translations of $\mathbf{A}$.

If any subset $\mathbf{A}$ of R or more elements (codewords) of a code (valid outputs of the protected device) has a unique multiset of differences then the multiset can be used to identify $\mathbf{A}$. Since translation does not change the multiset of differences, the set $\mathbf{A}$ can be uniquely identified even when affected by an error. If $\mathbf{A}$ can be uniquely identified irrespectively of the translation then the error is corrected.

Theorem 1. *A code $C \subset GF(q^n)$ is strong R-robust error correcting if for any ordered sets $\mathbf{A}, \mathbf{B}$ whose elements are a subset of C and $|\mathbf{A}| = |\mathbf{B}| = N, N \geq R$ their ordered multisets of differences are different $\triangle\mathbf{A} \neq \triangle\mathbf{B}$.*

Proof The construction of a robust error correcting code based on the uniqueness of a multiset of differences follows from the above discussion. In order for the code to be strong the code must be able to restore the order of elements in an ordered set. The code is strong if also all permutations of the elements in an ordered set $\mathbf{A}$ result in different ordered multiset $\triangle\mathbf{A}$. □

Example 3. *The code $C = \{(0|0), (1|1), (2|4), (3|4), (4|1)\}$, where $(x|y)$ denoted concatenation of elements $x, y \in GF(5)$ is a strong non-binary 2-robust error correcting code. Any ordered subset of two codewords has a unique multiset of differences. For example, the ordered set of two codewords $\mathbf{A} = ((0|0), (1|1))$ has a unique ordered multiset of differences $\triangle\mathbf{A} = ((4|4)), (1|1))$ which can be used to identify $\mathbf{A}$. Moreover the ordered set of differences is unique for every permutation of the order of elements of $\mathbf{A}$ so the code is a strong 2-robust error correcting code since each ordered set of at least two codewords has a unique multiset of differences.*

Robust error correcting codes based on the uniqueness of the multiset of differences can be constructed using known constructions of robust error detecting codes [3] [4]. In the next section we summarize some of the properties of the robust error detecting codes and show that they are also robust error correcting due to the properties of Theorem 1.

B. Robust Error Detecting Codes

Systematic robust error detecting codes were first presented in [3]. These codes can provide for almost uniform error detection against all errors and they have no undetectable errors [8].

Definition 3. ***(R-robust error detecting code (R-REDC))*** *A code $C \subset GF(q^n)$ with $R =: \max_{0 \neq e \in GF(q^n)} |\{w | w \in C, w + e \in C\}|$ is called a **R-robust error detecting code**.*

There are two main properties of the robust error detecting codes that make them different from classical error detecting codes. Robust error detecting codes have no undetectable errors and their error detection is data dependent which results in a high probability of detection for repeating errors. For a R-robust error detecting code there are at most R out of $|C| = M$ codewords which can potentially mask an error.

Good robust error detecting codes are those that achieve the lowest possible robustness R for a given size of a code $M = |C|$. For binary robust codes with $q = 2$, the smallest possible R is 2. For q-ary codes where $q > 2$ minimum R is 1. In both cases, systematic codes achieving $R = 2$ or $R = 1$ for binary and non-binary codes respectively have a rate of one-half. Three constructions of such robust error detecting codes are summarized below (from [3] [4]).

Construction 1. ***(Binary Cubic Robust Duplication Code)*** *Let $w = (x|y), x, y \in GF(2^k)$ where $(x|y)$ denoted concatenation of x and y. The robust duplication code C contains all vectors w which satisfy $x^3 = y$, where all the computations are in $GF(2^k)$. The code is a $(2k, 2^k)$ 2-robust error detecting code.*

Example 4. *The code from Example 1 is binary cubic robust duplication code with $k = 3$ and $n = 6$ where the primitive polynomial used for construction of $GF(2^3)$ is $\alpha^3 + \alpha + 1$.*

Construction 2. ***(Binary Inverse Robust Duplication Code)*** *Let $w = (x|y), x, y \in GF(2^k)$. The robust duplication code C contains all vectors w which satisfy $x^{-1} = y$, where $0^{-1} = 0$ and all the computations are in $GF(2^k)$. The code is a $(2k, 2^k)$ 2-robust error detecting code.*

Construction 3. ***(Non-Binary Quadratic Robust Duplication Code)*** *Let $w = (x|y), x, y \in GF(q^k)$, $q > 2$. The robust duplication code C contains all vectors w which satisfy $x^2 = y$ where all the computations are in $GF(q^k)$. The code is a $(2k, q^k)_q$ 1-robust error detecting code.*

These codes can detect any error if the error affects in the same way at least three different codewords for the binary codes and at least two different codewords for the non-binary code. Other constructions and variants of robust codes with larger R can be found in [3] , [4], [8]. Although not explored in detail in this paper, some robust error detecting codes correspond to difference sets and can be generated by many classical combinatorial structures such as projective planes [4].

Example 5. *The code from Example 3 is a non-binary 1-robust quadratic duplication error detecting code where codewords are in the form of $C = \{(x|x^2 \pmod 5)\}$ Any error $e = (x|y)$ where $x, y \in GF(5)$ is masked for at most 1 codeword. The error $e = (1|1)$, for example, is masked only for the codeword $w = (0|0)$ since $w + e = (1|1)$ is also a codeword of C. Addition of the error with any other codeword results in a vector that does not satisfy $(x|x^2 \pmod 5)$. (see [3] for a general proof of robustness for quadratic codes)*

In the next section we will show that robust error detecting codes can also be used for error correction of repeating errors. The binary robust duplication codes can correct any error if it affects at least three different codewords and

the non-binary robust duplication code can correct any error if it affects at least two different codewords.

C. Robust Error Detecting Codes are also Robust Correcting

Robust error-correcting codes for which subsets of $N \geq R$ codewords have unique multisets of differences of codewords are equivalent to robust error-detecting codes and offer robust error detection in addition to correction. Any method used to construct robust error-detecting codes can also be used to construct robust error-correcting codes. To show this we first start with a basic result about differences of elements of a set.

Lemma 1. *For ordered sets* $\mathbf{A}$ *and* $\mathbf{B}$ *whose elements are subsets of* $GF(q^n)$ *where* $\mathbf{A} \neq \mathbf{B}$, $|\mathbf{A}| = |\mathbf{B}|$, *and* $\triangle\mathbf{A} = \triangle\mathbf{B}$, *there exists a unique* $e \in GF(q^n)$ *such that* $\mathbf{A}_e = \mathbf{B}$.

Proof Let $\mathbf{A} = (a_1, a_2, \ldots, a_N)$ be an ordered set. The ordered multiset of differences of a set, $\triangle\mathbf{A}$, can be uniquely represented by the differences d_i, $1 < i \leq N$ where

$$a_1 - a_i \quad = \quad d_i.$$

The resulting ordered set of d_i, call it $\bigtriangledown\mathbf{A} = (d_2, d_3, \ldots, d_N)$, uniquely represents $\triangle\mathbf{A}$ as all code differences can be generated from $\bigtriangledown\mathbf{A}$. There are

$$\binom{q^n - 1}{N - 1}(N-1)!$$

different possible ordered multisets of differences of N elements. For each multiset $\triangle\mathbf{A}$ there exists the corresponding set $\mathbf{A}$. Since $\triangle\mathbf{A} = \triangle\mathbf{A}_e$ there are at least q^n different ordered sets $\mathbf{A}_e$ that can generate each of the distinct ordered multisets of differences. There are thus

$$\binom{q^n - 1}{N - 1}(N-1)!q^n = \binom{q^n}{N}N!$$

unique sets and their translates which have unique ordered multisets of differences . Since there are at total of $\binom{q^n}{N}N!$ unique ordered sets of N elements, there cannot exist two different N element ordered sets which have the same multiset of differences but are not translates of each other. □

Remark 2. *A* R*-robust error correcting code* $C \subset GF(q^n)$ *is strong if for any ordered set* $\mathbf{A} = (a_1, a_2, ..., a_N)$ *where* $a_i \in C$ *and* $N \geq R$, $\mathbf{A} \neq \mathbf{A}_e$ *for any* $e \neq 0, e \in GF(q^n)$

Theorem 2. *A* R*-robust error-detecting code is also a strong* $(R+1)$*-robust error-correcting code.*

Proof Let $C \subset GF(q^n)$ be a R-robust error-detecting code and let $\mathbf{A}$ be an ordered set whose elements are a subset of C where $|\mathbf{A}| \geq R+1$. Any such $\mathbf{A}$ has a unique $\triangle\mathbf{A}$ which can be used to identify $\mathbf{A}$. If there was ordered set $\mathbf{B}$ such that its elements are a subset of C and $\triangle\mathbf{A} = \triangle\mathbf{B}$ then from Lemma 1 there must exist an $e \in GF(q^n)$ such that $\mathbf{A}_e = \mathbf{B}$. From the definition of a R-robust error detecting code it easy to see that, $\forall e \neq 0, |C_e \cap C| \leq R$ thus there are no two different ordered $R+1$ sets $\mathbf{A}$ and $\mathbf{B}$ whose elements are from C where there exists e such that $\mathbf{A}_e = \mathbf{B}$. Clearly $\triangle\mathbf{A} = \triangle\mathbf{A}_e$ and thus $\triangle\mathbf{A}$ can be used to uniquely identify the set $\mathbf{A}$. □

Example 6. ***(Strong correction with non-binary codes)*** *Consider the non-binary quadratic code of Example 5. The code is a 1-robust error detecting code and by Theorem 2 is a strong 2-robust error correcting code.*

Theorem 3. *A R-robust error-detecting code over a field of characteristic two is a weak R-robust error-correcting code.*

Proof Any R codewords have a unique multiset of differences but different permutations of the ordered set of R different codewords can have the same ordered multiset of differences. We prove this by contradiction. Assume first that the multiset of differences $\triangle A$ of a subset of R (unordered) codewords $A \subset C$, $A = \{a_1, a_2, \ldots, a_R\}$ of a R-robust error detecting code is not unique. There must therefore be another set $B \subset C$, $B = \{b_1, b_2, \ldots, b_R\}$ with the same set of differences where $\triangle A = \triangle B$, $A \neq B$. Since these two sets have to be translates of each other (Lemma 1) we can therefore write

$$a_1 + a_i = d_i, \qquad b_1 + b_i = d_i, \tag{3}$$

where $1 < i \leq R$. Therefore

$$a_1 + b_1 = \quad a_i + b_i, \tag{4}$$

where $1 < i \leq R$ and + is componentwise addition modulo two.

The two sets A, B cannot be disjoint since that would imply that the original robust error-detecting code was $2R$-robust. To be distinct the two sets can have at most $R-2$ elements in common where clearly $a_1 \neq b_1$. Hence if there is another set with the same set of differences there must be at least $\lceil (R-1)/2 \rceil + 1$ pairs of codewords with the same difference and hence that would imply that the original error detecting code is more than R-robust, which is a contradiction. Thus every unordered subset of R codewords of a binary R-robust error detecting code has a unique multiset of differences. The code, however is not strong since by definition of a R-robust error detecting code there exists a R element subset A of C where $A = A_e$ for some nonzero $e \in GF(2^n)$. □

The three construction of robust error detection codes described in Section IV-B can be used for robust error correction. In addition to their error detecting properties, the binary codes are 2-robust weak correcting and 3-robust strong correcting. The non-binary quadratic codes are strong 2-robust error correcting.

Example 7. ***(Correction with binary cubic codes)*** *The code from Example 1 is a binary cubic robust duplication code where the primitive polynomial for $GF(2^3)$ is $\alpha^3 + \alpha + 1$. It is a 2-robust error detecting code. As shown in Example 1 it is a weak 2-robust error correcting code. It is also a strong 3-robust error correcting code. Any ordered set of 3 or more different codewords has a unique ordered multiset of differences of codewords.*

In the next section we analyze decoding methods for robust error correcting based on unique multisets of differences.

V. Block Decoding

The decoding of robust error-correcting codes based on unique multiset of differences of codewords is in the general form computationally prohibitive but more efficient methods which use the algebraic structure of the code are also possible for some codes. We start by analyzing the more general list decoding method. Efficient algebraic algorithms for codes from Constructions 1, 2, and 3 are also presented.

A. General List Decoding

The general list decoding method for strong error correction is based on a precomputed set of valid multisets of differences for all possible ordered N-element subsets of codewords. Prior to decoding the algorithm requires a setup phase where all the multisets of differences of all possible ordered N codeword subsets of the code are precomputed. When an ordered set $\mathbf{A}$ of N erroneous outputs is received the set of differences is computed and compared to each precomputed set of possible multisets of differences. If there is a match then the set associated with the matching precomputed multiset of differences is determined to be the set of intended outputs. If there are no matches the algorithm assumes an inconsistent error (i.e. an error which was not the same for all N outputs) and refuses to decode.

Correct decoding only occurs iff all of the N outputs of a protected device are distorted by the same error. The probability of correct decoding is equal to the probability of having N consecutive equal error patterns and depends on the properties (such as the laziness) of the errors. When the errors are not all the same the general algorithm will either continue decoding and decode incorrectly or determine an inconsistency in the errors and refuse to decode.

For a $(n, M)_q$ strong R-robust error-correcting code which uses $N \geq R$ outputs for decoding there are q^{nN} possible distortions to the Nn q-ary digit stream. Using list decoding the code can correct all errors which repeat for all of the outputs. The code can therefore correct q^n distortions to the Nn digit stream.

Wrong correction will occur when an error will distort the set of outputs to an another set which has a different, (but valid) multiset of differences of codewords. Sets which have a valid multiset of differences are any N subset of the codewords and any of their translates (see Lemma 1).

Using list decoding based on multisets of differences of codewords the fraction of distortions which will result in wrong correction can be made arbitrarily small by increasing the number of outputs N used for decoding or by increasing the dimension (number of digits) n in each output. The ratio β of refusal to decode to wrong correction is

$$\beta = \frac{q^{nN} - \binom{M}{N} q^n N!}{\binom{M}{N} q^n N! - q^n},$$

which increases exponentially with respect to both N and n. The codes and decoding methods based on multiset of differences of codewords can have arbitrarily small probabilities of miss-correction allowing correction even when errors are not completely lazy. For strong decoding using $N = 3$ distorted outputs the binary cubic robust duplication codes where the dimension of the codes is $2k$ the ratio $\beta \simeq 2^k$ for large k.

The list search for the correct value is computationally intensive and clearly not practical for real applications. The algebraic structure of a code can be used to decrease the complexity of decoding while achieving the same detection and correction parameters.

B. Algebraic Decoding of Binary Duplication Cubic Codes

Robust codes which have a well defined algebraic structure can be decoded using a much simpler algorithm than list decoding. Robust correction for the binary cubic error detecting codes, for example, can be done efficiently while maintaining the same properties with respect to miscorrection as in list decoding. We next outline algebraic methods for decoding for both binary and arithmetic (non-binary) errors. Algorithm 1 presented in this section refers to decoding for binary codes where errors are modeled as componentwiese modulo 2 additions with the codewords. Algorithm 2 presented in the next section refers to decoding for non-binary arithmetic robust codes where errors are modeled additive in the respective field.

Let $C = \{w_1, w_2, \ldots, w_M\}$ be a code where $w_i = (x_i | y_i = x_i^3)$, $x_i, y_i \in GF(2^k)$. Let error $e = (e_x | e_y) \in GF(2^{2k})$; $e_x, e_y \in GF(2^k)$, $e \neq 0$. Since the code is based on a binary 2-robust error detecting code, by Theorem 2 and Theorem 3 it is a weak 2-robust error-correcting code as well as a strong 3-robust error-correcting code.

If an error e affects two different codewords the relationship between the correct and erroneous outputs $\tilde{w}_i = (\tilde{x}_i | \tilde{y}_i)$ is

$$x_1 + e_x = \tilde{x}_1, \tag{5}$$

$$x_1^3 + e_y = \tilde{y}_1, \tag{6}$$

$$x_2 + e_x = \tilde{x}_2, \tag{7}$$

$$x_2^3 + e_y = \tilde{y}_2. \tag{8}$$

where $+$ is componentwise XOR. The value of x_1 from (5) can be substituted into (6) and likewise the value of x_2 from (7) can be substituted into (8). The two resulting equations after substitution can be summed and after algebraic manipulation we have

$$x_2^2 + x_2(\tilde{x}_1 + \tilde{x}_2) + (\tilde{x}_2^2 + \tilde{x}_1^2) + \frac{\tilde{y}_1 + \tilde{y}_2}{\tilde{x}_1 + \tilde{x}_2} = 0. \tag{9}$$

Equation (9) is a quadratic equation in $GF(2^k)$ with respect to x_2, hence there can be at most two solutions which correspond to the correct outputs, x_1 and x_2 (if $x_1 \neq x_2$). However, with only two distorted outputs, unless there are further restrictions on the messages, it is not possible to determine their order. Solving the quadratic equation is thus decoding (weak correction) of the robust codes. Since the order of the codewords cannot be determined the method provides only for weak correction. The quadratic equation can have zero or two solutions. When the quadratic has no solutions it is the indication that the multiset of differences was not consistent with any valid set of two codewords and should be interpreted as a refusal to decode.

For strong correction by Theorem 2 three distorted outputs are necessary and the same error must affect all three to be correctable. With three distorted outputs the decoding can be simplified to solving a linear equation since Equations (5) to (9) can be repeated taking the second and third distorted codewords which will produce another quadratic equation

$$x_2^2 + x_2(\tilde{x}_2 + \tilde{x}_3) + (\tilde{x}_2^2 + \tilde{x}_3^2) + \frac{\tilde{y}_2 + \tilde{y}_3}{\tilde{x}_2 + \tilde{x}_3} = 0. \tag{10}$$

From (9) and (10) we can obtain a linear equation

$$x_2(\tilde{x}_1 + \tilde{x}_3) + (\tilde{x}_1^2 + \tilde{x}_3^2) + \frac{\tilde{y}_2 + \tilde{y}_3}{\tilde{x}_2 + \tilde{x}_3} + \frac{\tilde{y}_1 + \tilde{y}_2}{\tilde{x}_1 + \tilde{x}_2} = 0, \tag{11}$$

which has one unique solution for x_2. Similar equations can be written for x_1 and x_3:

$$x_1(\tilde{x}_2 + \tilde{x}_3) + (\tilde{x}_2^2 + \tilde{x}_3^2) + \frac{\tilde{y}_1 + \tilde{y}_3}{\tilde{x}_1 + \tilde{x}_3} + \frac{\tilde{y}_1 + \tilde{y}_2}{\tilde{x}_1 + \tilde{x}_2} = 0, \tag{12}$$

$$x_3(\tilde{x}_1 + \tilde{x}_2) + (\tilde{x}_1^2 + \tilde{x}_2^2) + \frac{\tilde{y}_2 + \tilde{y}_3}{\tilde{x}_2 + \tilde{x}_3} + \frac{\tilde{y}_1 + \tilde{y}_3}{\tilde{x}_1 + \tilde{x}_3} = 0. \tag{13}$$

By computing the three correct outputs x_1, x_2, x_3 the respective errors can also be computed. If all three errors are consistent (all the same) then the decoded outputs are returned. Inconsistencies in the computed errors signify a non-constant error which resulted in an invalid multiset of differences. This is indicated by the refusal to decode.

The algebraic decoding method for the binary cubic codes is summarized in Algorithm 1. For a set of different N distorted outputs received, the algorithm solves the linear Equations (11)-(13) for all $\binom{N}{3}$ triples of erroneous outputs. Calculating the expected outputs allows the calculation of the error. A consistent error pattern for all of the distorted outputs is taken as the error. Refusal to decode occurs when any of the received outputs are equal, an error is not detected in more than two outputs, and when the computed error is not consistent for all outputs.

C. Algebraic Decoding of Binary Duplication Inversion Codes

Similarly to the binary cubic codes, the codes binary inversion codes can also be efficiently decoded. Following the same analysis procedure as for the cubic codes it can be shown that for weak correction using two outputs decoding involves solving a quadratic equation shown below.

$$x_2^2(\tilde{y_1} + \tilde{y_2}) + x_2(\tilde{y_1} + \tilde{y_2})(\tilde{x_1} + \tilde{x_2}) + (\tilde{x_1} + \tilde{x_2}) = 0. \tag{14}$$

For strong correction three distorted outputs are necessary and the same error must affect all three to be correctable. With three outputs the decoding can be simplified to solving a linear equations. For the inversion codes those linear equations are listed below.

$$x_2(\tilde{x_1} + \tilde{x_3}) + \frac{\tilde{x_1} + \tilde{x_2}}{\tilde{y_1} + \tilde{y_2}} + \frac{\tilde{x_2} + \tilde{x_3}}{\tilde{y_2} + \tilde{y_3}} = 0, \tag{15}$$

Algorithm 1 Strong algebraic block decoding of a binary cubic robust code

Require: ordered set $\tilde{\mathbf{A}} = (\tilde{w}_1, \ldots, \tilde{w}_N)$ of distorted codewords,$\tilde{w}_i = (\tilde{x}_i|\tilde{y}_i)$, $|N| \geq 3$

Ensure: decoded set $\mathbf{A}$ of N outputs and error e or refusal to decode

```
if x̃_i^3 = ỹ_i for two or more w̃_i then
    return refuse to decode
else
    for all triples of outputs solve Equations (11), (12), (13) and determine the respective errors
    if the calculated errors for all triples are not all equal then
        return refuse to decode
    else
        return for the consistent error, return Ã_e, e
    end if
end if
```

$$x_1(\tilde{x_2} + \tilde{x_3}) + \frac{\tilde{x_1} + \tilde{x_2}}{\tilde{y_1} + \tilde{y_2}} + \frac{\tilde{x_1} + \tilde{x_3}}{\tilde{y_1} + \tilde{y_3}} = 0, \tag{16}$$

$$x_3(\tilde{x_1} + \tilde{x_2}) + \frac{\tilde{x_2} + \tilde{x_3}}{\tilde{y_2} + \tilde{y_3}} + \frac{\tilde{x_1} + \tilde{x_3}}{\tilde{y_1} + \tilde{y_3}} = 0. \tag{17}$$

The algebraic decoding algorithm is the same as for the binary cubic codes summarized in Algorithm 1. For a set of different N distorted outputs received the algorithm solves the linear equations above for all outputs triples (Equations (15) (16) (17)). Calculating the expected outputs allows the calculation of the error.

D. Algebraic Decoding of Non-Binary Quadratic Duplication Codes for Arithmetic Errors

Efficient algebraic robust correction algorithm is also possible for arithmetic errors (which are typical errors for arithmetical devices such as adders or multipliers [9]). We show that a similar decoding algorithm, as was shown for the binary codes, is possible for non-binary quadratic codes defined by $C = \{(x|x^2 \bmod q)\}$ where $x \in GF(q)$ and q is a prime. These codes are systematic 1-robust error detecting codes and thus only two outputs are necessary to perform the strong error correction.

Let $C = \{w_1, w_2, \ldots, w_M\}$ be a code where $w_i = (x_i|y_i = x_i^2 \bmod q)$, $x_i, y_i \in GF(q)$. Let error $e = (e_x|e_y)$, $e_x, e_y \in GF(q)$, $e \neq 0$. A codeword w is distorted by an error e by modulo addition of the information and redundant portions of the code $\tilde{w_i} = w_i + e = (\tilde{x} = x_i + e_x \bmod q|\tilde{y} = y_i + e_y \bmod q)$. Such an error model can be applicable to arithmetic devices since the hardware computing the information and redundant portions is implemented disjointly.

If an error e affects two different codewords the relationships between the correct and distorted outputs (where all operations are $\bmod q$) are

$$x_1 + e_x = \tilde{x}_1, \tag{18}$$

$$x_1^2 + e_y = \tilde{y}_1, \tag{19}$$

$$x_2 + e_x = \tilde{x}_2, \tag{20}$$

$$x_2^2 + e_y = \tilde{y}_2. \tag{21}$$

From Equation (18)-(21), (similarly as in for Equations(5)-(8)) we have

$$x_2(2\tilde{x}_1 - 2\tilde{x}_2) + \tilde{x}_1^2 + \tilde{x}_2^2 - 2\tilde{x}_1\tilde{x}_2 - \tilde{y}_1 + \tilde{y}_2 = 0 \bmod q. \tag{22}$$

Equation (22) is linear with respect to x_2 which results in a unique solution for x_2 resulting in strong correction using only two distorted outputs.

The algebraic decoding method for the non-binary quadratic duplication codes and arithmetic errors is summarized in Algorithm 2. A consistent error pattern for all of the outputs is taken as the error. Refusal to decode occurs when any of the received outputs are equal, an error is not detected in more than one outputs, and when the computed error is not consistent for all outputs.

Algorithm 2 Algebraic block decoding of a non-binary quadratic duplication robust code

Require: ordered set $\tilde{\mathbf{A}} = (\tilde{w}_1, \ldots, \tilde{w}_N)$ of different distorted outputs,$\tilde{w}_i = (\tilde{x}_i|\tilde{y}_i)$, $|N| \geq 2$

Ensure: decoded set $\mathbf{A}$ of N outputs, refusal to decode

1: **if** $\tilde{x}_i^2 = \tilde{y}_i$ for one or more w_i **then**
2: **return** refuse to decode
3: **else**
4: for all triples of outputs solve Equation (22), and determine the respective errors
5: **if** the calculated errors for all pairs are not all equal **then**
6: **return** refuse to decode
7: **else**
8: **return** for the consistent error, return $\tilde{\mathbf{A}}_e$, e
9: **end if**
10: **end if**

Example 8. *Consider the non-binary quadratic code of Example 5. As shown in Example 3 the code is a strong 2-robust error correcting code for $q = 5$ and can be decoded using Algorithm 2. Assume the ordered set of two codewords $\mathbf{A} = ((0|0), (1|1))$ is distorted by an error $e = (1|1)$ and results in a distorted ordered set $\tilde{\mathbf{A}} = ((1|1), (2|2))$. Solving Equation (22) with $\tilde{x}_1 = 1$, $\tilde{x}_2 = 2$, $\tilde{y}_1 = 1$, $\tilde{y}_2 = 2$ we obtain $x_2 = 1$ and therefore $x_1 = 1$ and the decoder can correctly determine the error to be $e = (1|1)$.*

To conclude this section we note that the proposed nonlinear arithmetic codes can detect or correct all arithmetical errors while the well-known linear AN-arithmetic codes [10] can be used only for specific classes of errors.

VI. Decoding of Streamed Data

In practical implementations for hardware devices the decoding must be performed on a continuous stream of data that is the output of a device. Two possible methods of processing the continuous data and their limitations with the block decoding methods are outlined below.

A. Segmented Block Decoding

A continuous stream of distorted outputs can be processed with the algorithms presented in Section V by dividing the stream into disjoint N element sets and performing block decoding (see Figure 2). Sequential block decoding of non overlapping blocks is simple to implement, the method can result in inefficient decoding for non constant errors even when they have a large span. Unless the length of the burst of identical errors will be exactly the width of the decoder N, many of the errors will not be decoded properly without additional recovery procedures.

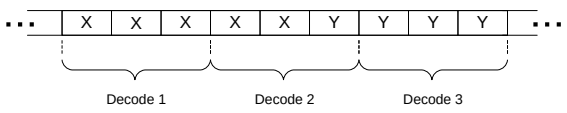

Fig. 2. Segmented block decoding where $N = 3$

To illustrate the potential limitation consider a block decoder for a strong 3-robust error-correcting code with $N = 3$. Consider a burst of two errors depicted by "X" and "Y" in Figure 2. The bursts of errors can be decoded correctly if each of the bursts of errors falls within exactly a complete decoding blocks. However, if the errors change within a decoding block as in the block 2 in Figure 2 $N = 3$ distorted outputs will not be decoded.

B. Continuous Decoding

To reduce number of uncorrected outputs at error transitions a decoding procedure based on continuous decoding can be adopted similar to decoding methods of convolution codes [11]. In the decoding procedure the block decoding is performed in an overlapping fashion. The method allows for more consistency checks as decoding proceeds since the effective block length of the procedure is increased.

The continuous decoding is based on the normal block decoding in overlapping blocks (see Figure 3). The overlapping nature of the decoding can produce conflicting results at the output of the decoder. A computed error of a following decoded block can be used to verify the error of the previous block. This continuous overlapping decoding procedure reduces the chances of miscorrection without increasing the number of outputs in the individual block decoding algorithm. In addition, it also allows efficient detection of a change of an error. Continuous decoding, however, requires a factor of $N - 1$ more block decoding steps than block decoding

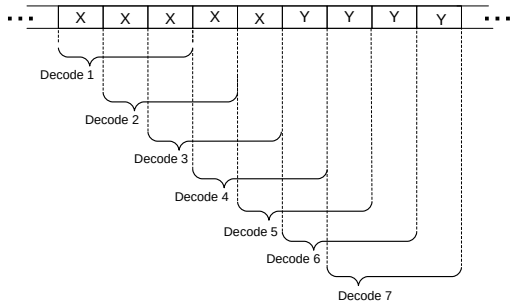

Fig. 3. Continuous decoding where $N = 3$

C. Other Decoding Considerations

1) Correction on detected errors: Errors or manifestation of faults can be non continuous. Even in the case of lazy errors, an error might only repeat when the fault causing the error manifest itself, but the manifestations can be separated by a long block of non-erroneous outputs. For example, for the stuck-at fault model the errors, even when constant, will not be continuous but separated by outputs with no errors.

For such non-continuous manifestations, one approach is to perform error detection first and do correction only on the outputs found to be erroneous. A limitation of such an approach is that some outputs will not be corrected even when a constant error affects all outputs. Any output can be distorted by an error that will map it to a different codeword. Such distortions cannot be detected and would not be corrected by a scheme combined with error detection.

2) Different Codewords: For successful decoding the algorithms presented require and assume at least R different codewords to be affected by the same error. The requirement can be satisfied by additional counter bits appended to each of the codewords to ensure that each N consecutive codewords are different. Such an approach would require $\lceil \log_2 N \rceil$ additional bits. For correction based on only erroneous or non continuous errors the requirement is harder to satisfy without additional restrictions.

VII. Case Study: Correction using Cubic Robust Codes for Lazy Errors

In this section we develop and analyze a decoding architecture based on the robust cubic error correcting codes (Construction 1). We investigate the hardware requirements associated with decoding and the properties of the errors for which the codes become efficient.

A. Decoding Architecture for Strong Correction

The structural representation of a decoder for $N = 3$ is shown in Figure 4. The architecture is based on a continuous decoder for the cubic robust codes which assumes a continuous error blocks and performs correction on all incoming outputs of a device.

The architecture is composed of a shift register which buffers three n bit outputs of a device that are the inputs to a block decoder. The block decoder implements Algorithm 1. The block decoder takes the three distorted codewords of a robust code as input and outputs a computed error and/or a status tag depending on the outcome of the block decoding procedure. The tag signifies one of two possible outputs of the decoder: correction, or refusal to decode. The output of the block decoder passes through the block which enforces the decoding policy.

The decoding policy block implements rules for updating the errors in the error and tag shift registers depending on the output of the block decoder. The details of the updating policies are mainly used to customize the way the complete decoder deals with boundary conditions of change of errors. For the design when the output of the block decoder results in correction the decoding block updates the tags and errors in the shift register that resulted from refusal to decode by the block decoder. The decoding policy block can be implemented using comparators.

Depending on the left most tag of the lower shift register the output of the complete decoder is multiplexed between either the original output or the corrected output. If the tag signifies that the error was a result of a refusal to decode the original output is used as the output, and a corrected output is used otherwise. The corrected output is simply the XOR of the original output and the computed error in the shift register.

The complete architecture consists of two shift registers, a multiplexer, comparators, XOR gates, and a block decoding block. The block decoding block is the most hardware and computation intensive part of the decoder (see Figure 4).

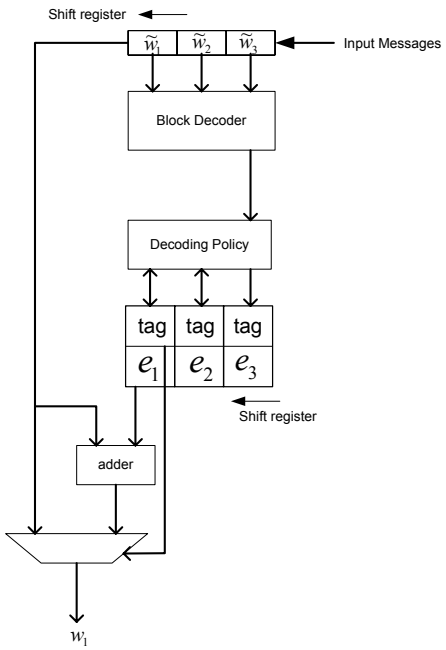

Fig. 4. Continuous decoder with $N = 3$

The block decoder requires seven additions, four multiplications, and one multiplicative inversion over $GF(2^{n/2})$

for each of the three linear equation the decoder solves. The complete block decoder requires 21 additions, 10 multiplications and one inversion operations over $GF(2^{n/2})$.

B. Example of strong correction

Consider the stream of expected outputs $A = ((1|1), (4|1), (2|4), (0|0), (1|1))$ which are codewords of a non-binary quadratic duplication code $C = \{(x|x^2 \bmod 5)\}$, $C = \{(0|0), (1|1), (2|4), (3|4), (4|1)\}$. The expected outputs are distorted by a stream of errors $E = ((0|1), (0|1), (1|1), (1|1), (1|1))$ such that the observed set of outputs is $\tilde{A} = A + E = ((1|2), (4|2), (3|0), (1|1), (2|2))$. The decoding steps of the stream of erroneous outputs $\tilde{A}$ using a continuous strong decoder with $N = 2$ are summarized in Table II.

TABLE II

CONTINOUS DECODER EXAMPLE FOR NON-BINARY QUADRATIC DUPLICATION CODE WITH $N = 2$

decode	Shift Reg. 1	Output of Block Decoder	Shift Reg. 2	Output
1	$((1\|2), (4\|2))$	$(0\|1)$	tag=$(1, 1)$ $e = ((0\|1), (0\|1))$	$(1\|1)$
2	$((4\|2), (3\|0))$	$(-\|-)$	tag=$(1, 0)$ $e = ((0\|1), (-\|-))$	$(4\|1)$
3	$((3\|0), (1\|1))$	$(1\|1)$	tag=$(1, 1)$ $e = ((1\|1), (1\|1))$	$(2\|4)$
4	$((1\|1), (2\|2))$	$(1\|1)$	tag=$(1, 1)$ $e = ((1\|1), (1\|1))$	$(0\|0)$
5	$((2\|2), (-\|-))$	$(-\|-)$	tag=$(1, -)$ $e = ((1\|1), (-\|-))$	$(1\|1)$

The decoding starts with shifting of the stream of erroneous outputs $\tilde{A}$ into the first shift register. When the first two erroneous outputs ($(1|2)$ and $(4|2)$) are in the shift register the block decoder solves the linear Equation (22) and outputs the calculated error for the two outputs in the shift register. For the first two outputs the error is correctly calculated to be $(0|1)$ which is stored in the lower shift register and used to correct the first output. Once the first output is corrected the first shift register values are shifted and now contain the second and third erroneous values which were distorted by two different errors. Solving Equation (22) the calculated error does not result in two codewords and so the block decoder refuses to decode in the decode step 2 (refusal to decode is denoted as $(-|-)$). The second erroneous output is corrected correctly since the error $(0|1)$ from decode 1 (see Table II) stored in the second shift register is used for correction. For decode 3 the two erroneous outputs are distorted by the same error $(1|1)$ and the decoder correctly determines the error and updates the unknown errors of the lower shift register. As can be seen the decoder results in correct decoding of all of the erroneous outputs.

C. Decoding Performance for Lazy Errors

Simulations were performed to evaluate the performance and the required parameters for which the decoder becomes efficient. The decoder was tested with respect to laziness of errors for various code parameters. For all simulations and experiments random outputs were distorted with randomly selected errors of a fixed laziness L. All errors were assumed to be equiprobable and assumed to have the same laziness.

For the simulations a stream of 10,000 random codewords of the cubic robust codes (x, x^3) were generated. The outputs were then distorted with random errors of a fixed laziness L. The stream of erroneous outputs was the input to the continuous decoder with $N = 3$ that was described above. The corrected outputs of the decoder were compared with the error-free outputs to determine how many outputs were corrected correctly. The results for the $N = 3$, $n = 12$ decoder are summarized by the dotted line in Figure 5.

As Figure 5 shows the performance of the decoder depends heavily on the laziness L of the errors. In order to be correctable an error must have a span of at least three outputs. Errors with a span of less than three consecutive outputs will be most often not decoded or in some instances miscorrected. The black solid line of Figure 5 shows the percentage of errors which has a span longer or equal to three.

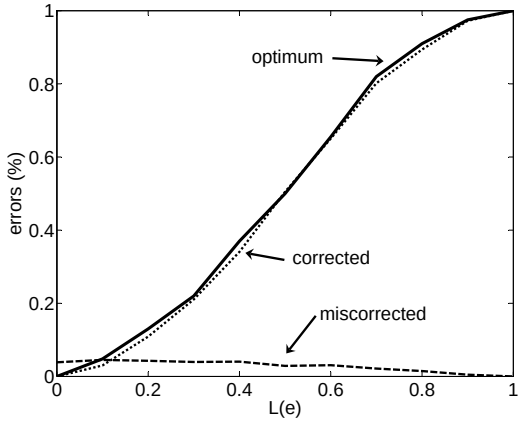

Fig. 5. Percentage of errors corrected and miscorrected as a function error laziness (dotted and dashed lines respectively) and percentage of correctable errors (solid line) for a robust decoder with $N = 3, n = 12$ using binary cubic duplication codes

The difference of the two lines (the dotted and solid lines) is the result of occasional miscorrection at a boundary of two spans of different errors. This probability of miscorrection can be made arbitrarily small by increasing either n or N of the block decoder. Increasing n will result in a larger width of components of the decoder. The value of n is often fixed by the source of the stream such as a piece of original hardware that require protection and often cannot be modified. Increasing the number of blocks, N, will likewise reduce the probability of miscorrection. An increase in N, however will result in an increase of the decoding complexity as well as a reduction of the percentage of correctable errors. An increase in N means that only errors of a span of at least N are correctable. The effects of N on the percentage of corrected errors is shown in Figure 6. While $N = 2$ provides the maximum percentage of correctable errors it only results in weak correction and has the highest probability of miscorrection.

As can be seen from the simulated curves, robust correction is best suited for the case when errors have a high laziness or typically have a long span.

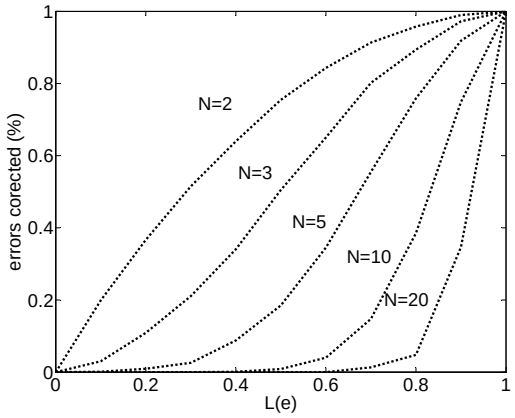

Fig. 6. Percentage of errors corrected as a function of error laziness for various N using binary cubic duplication codes where $n = 12$ for weak correction where $N = 2$ and strong otherwise.

VIII. Case Study: Strong Correction Using Inversion Robust Codes for Faults in Combinational Circuits

To test the feasibility of correction of errors due to faults in combinational devices a sample circuit was protected with modified robust inversion duplication codes (Construction 2). The circuit was then simulated and correction on the outputs of a faulty sample circuits was performed. To simplify the simulations only single stuck-at-zero faults were considered.

A circuit implementing the substitution box (SBox) of the Advanced Encryption Standard (AES) was used in the simulations [12]. The original unprotected circuit has and 8-bit input and an 8-bit output. The circuit implements the multiplicative inverse (over $GF(2^8)$) followed by an affine transform D of the input. The circuit was chosen for its nonlinearity to show that codes can be beneficial for a wide range of combinational circuits including those implementing highly nonlinear functions.

The predictor generating the redundant bits of the extended output needed to provide the robust protection to the circuit results in only a 7% hardware overhead. For the design the predictor for the circuit consisted of the same affine transform used in the Sbox that can be constructed with 14 two-input gates. The complete circuit consists of 207 two input gates. The output of the Sbox and the affine transform of the input results in a 16-bit extended output that is a codeword of a robust error correcting code. The code consists of vectors in the form

$$C = \{(x, y = D^*(Dx)^{-1})\}, \tag{23}$$

where D^* is the inverse of the affine transform D over $GF(2)$. The code C is a 2-robust error detecting code and a strong 3-robust error correcting code.

To evaluate the feasibility and benefits of robust correction the device was simulated in the presence of single stuck-at-zero faults. Stuck-at faults were inserted into the netlist of the complete circuit (including the predictor) and kept constant for 100 random inputs. After 100 inputs another randomly selected gate output was modified to

stuck-at zero and 100 more random inputs were simulated for the fault circuit. This was performed for a total of 10,000 random inputs for a total of 100 randomly selected stuck-at zero faults. The resulting 10,000 outputs were saved and processed.

Out of 10,000 inputs for which one node in the circuit was stuck-at-zero, 4,498 resulted in an error at the output of the device. A histogram showing the Hamming weight distribution of the errors at the 16-bit extended output of the protected device is shown in Figure 7. As the Figure shows, even the single faults can results in errors of a large Hamming weight. The Hamming weight of the errors is limited to 8 due to separation of hardware generating the information and redundant bits, single fault can only propagate an error to either the information or redundant bits.

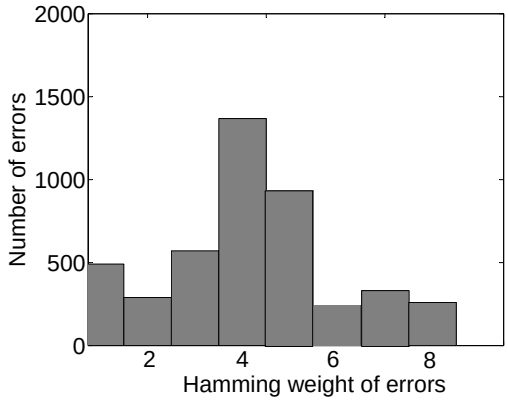

Fig. 7. Histogram for number of errors of a given Hamming weight resulting from single stuck-at-zero faults after 10,000 random inputs

The 10,000 outputs were used as an input to the robust error decoder with $N = 3$ for decoding based on the inversion based duplication codes. Since the errors at the output of the circuit were non-continuous the decoder performed decoding on outputs which were detected to be erroneous by the code.

Due to the robust error detection properties of the code all of the 4,498 erroneous outputs were detected by the code and were used in error correction. Of the 4,498 erroneous outputs used in correction 1,916 were corrected correctly (42.6%). Although the circuit implemented a highly nonlinear function, the average laziness of the errors due to the single stuck-at-zero faults was about 0.45. Of the 2,582 erroneous outputs that were not corrected only 236 were miscorrected (5.25%), and the rest were refused to be decoded (see Table III). The 5.35% of miscorrections can be in part attributed to the fact that no restrictions were placed that the outputs used for decoding are all different.

The errors exhibited a relatively large laziness even when it was the faults that were injected. Had the laziness of the errors been higher, correction would improve. Due to the structure of the circuit some of the errors exhibited no laziness and did not repeat. Therefore, even though the protecting code is capable of correcting all errors of any multiplicity since some errors did not repeat, they were uncorrectable in this implementation.

We note that to guarantee the same percentage (92.6%) of correct corrections as the proposed robust code, a

TABLE III

PERFORMANCE OF ROBUST ERROR CORRECTION AND DETECTION (x, x^3) CODES WITH $n = 16$ FOR SBOX OF AES

	number	percent
total errors	4,498	-
errors detected	4,498	100%
errors corrected	1,916	42.6%
errors miscorrected	236	5.25%

classical linear minimum distance BCH code would need to correct errors of Hamming weight up to 4 and thus needs a minimum distance of at least 9. Such a code requires more than the eight redundant bits and would require much more than the 7% overhead for the predictor than the robust code. Published implementations of the circuit with protection based on Hamming codes (d=3), for example, require more than 50% hardware overhead for encoding [8]. Alternatively, to achieve comparable protection triplication (TMR) may be used which requires a 200% overhead.

Furthermore, an additional benefit of the robust correction is its low percentage of miscorrection. Errors which were not corrected are still identified as being erroneous and not falsely corrected.

IX. APPLICATION SCENARIOS

The proposed error correction methods are effective against repeating errors that exhibit laziness. For many circuits encoding can be performed efficiently with relatively small overhead depending on the required robustness of the code used for the protection of the original device. Error correction requires buffering of multiple outputs and performing computations over a finite field and carry a significant processing or hardware requirement depending on weather it is implemented in software or hardware.

The proposed error correction technique may be used to provide reliability to simple devices such as sensor networks. Large networks of simple devices are deployed in an area which then wirelessly relay their data to a central node or log their data to local memory. The devices are often deployed unattended or in non-easily accessible areas prohibiting device replacement if the devices are faulty.

Such devices can be protected with the proposed robust error correcting codes. The logged streams of data encoded with robust error correcting codes from the devices can then be processed off-line by a more powerful central unit. By encoding the computations of the sensor nodes with codewords of robust codes erroneous outputs due to faults that can cause repeating errors may be corrected. In the case when errors are not correctable, the proposed robust codes can still provide for a very high level of error detection allowing discrimination of erroneous and correct data.

X. CONCLUSION

A R-robust error correcting code can correct any error if the error affects in the same way at least R different codewords. Efficient decoding is possible for binary robust codes based on the cubic construction and for non-

binary codes based on the quadratic construction. The binary codes are strong 3-robust error correcting codes and the non-binary codes are strong 2-robust error correcting codes for arithmetic errors. For these codes, the error correction has been shown to be effective if the span of an error is at least 3 or 2 outputs respectively. As a results, robust correction is most useful when errors have a high probability of repeating. The codes have a high probability of error detection and can be used even when the error is non-continuous.

The robust correction becomes efficient when the laziness of the errors is more than 0.3 using two output decoding. For correction using two outputs using the binary duplication codes (weak correction) the codes can correct almost 60% of the errors (see Figure 6). The probability of correction is practically independent of the Hamming weight of the error or the error distribution. As it was shown in Section VIII circuits which implement highly nonlinear functions can have high laziness of errors due to single stuck-at-zero faults where the correction can be effective on almost half of the erroneous outputs achieving better performance than comparable minimum distance codes.

REFERENCES

[1] P. K. Lala, *Self-checking and Fault-tolerant Digital Design*, 1st ed. Morgan Kaufmann, 2001.

[2] N. Jha and S. Gupta, *Testing of Digital Systems*. Cambridge University Press, 2003.

[3] M. G. Karpovsky and A. Taubin, "New Class of Nonlinear Systematic Error Detecting Codes," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818–1820, 2004.

[4] K. J. Kulikowski, M. G. Karpovsky, and A. Taubin, "Robust codes and robust, fault-tolerant architectures of the advanced encryption standard," *Journal of Systems Architecture*, vol. 53, no. 2-3, pp. 139–149, 2007.

[5] K. J. Kulikowski, M. G. Karpovsky, A. Taubin, Z. Wang, and A. Kulikowski, "Concurrent Fault Detection for Secure QDI Asynchronous Circuits," in *Workshop on Dependable and Secure Nanocomputing (WDSN)*, June 2008.

[6] C. Giraude, "DFA on AES," in *4th International Conference on Advanced Encryption Standard (AES) vol.3373 of Lecture Notes in Computer Science,*. Springer-Verlag, 2005, pp. 27–41.

[7] L. Kanal and A. Sastry, "Models for channels with memory and their applications to error control," *Proceedings of the IEEE*, vol. 66, no. 7, pp. 724–744, July 1978.

[8] K. J. Kulikowski, Z. Wang, and M. Karpovsky, "Comparative Analysis of Fault Attack Resistant Architectures for Private and Public Key Cryptosystems," in *Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2008.

[9] D. Siewiorek and R. Swarz, *Reliable Computer Systems: Design and Evaluation*. AK Peters, 1998.

[10] I. Koren and M. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, 2007.

[11] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Prentice Hall, 2004.

[12] *Advanced Encryption Standard (AES)*, Federal Information Processing Standards Publication 197, 2001.