IET Communications

Research Article

IET Journals
The Institution of Engineering and Technology

Multi valued parity generator based on Sudoku tables: properties and detection probability

ISSN 1751-8628
Received on 5th March 2019
Revised 25th April 2020
Accepted on 12th May 2020
doi: 10.1049/iet-com.2019.0247
www.ietdl.org

Shahab Ghalamdoost Pirbazari[1], Alireza Souri[2], Reza Faghih Mirzaee[3] ✉, Sam Jabbehdari[1]

[1]*Department of Computer Engineering, North Tehran Branch, Islamic Azad University, Tehran, Iran*
[2]*Young Researchers and Elite Club, Islamshahr Branch, Islamic Azad University, Islamshahr, Iran*
[3]*Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran*
✉ *E-mail: r.f.mirzaee@qodsiau.ac.ir*

Abstract: Parity-check is a simple yet effective error detection method. Several other more sophisticated error detection techniques are founded upon single parity-check (SPC). Exclusive-OR (XOR) is known as the parity generator in binary logic. This study suggests some multi-valued parity generators (MPGs), studies their behaviour, and provides a full discussion about their necessary and optional properties. The concept of Sudoku with some customised rules is used to create MPGs, which are the extended versions of binary parity generator in higher radixes. They are capable of revealing all single-digit errors. Additionally, they can detect incorrect data delivery with high probability when more than one error occurs (with even higher probability than XOR). The probability of error detection is analytically calculated for the occurrence of two to five errors in different bases. The calculations are then experimentally verified by a formal verification method. The authors' investigations show that error detection probability increases in higher radixes, and it is independent of dataword size.

1 Introduction

Noise and other impairments might cause errors during data transmission from source to destination. Error detection and correction techniques are required to enable reliable delivery of data over an unreliable communication channel. Error-detecting techniques such as parity-check, checksum, and cyclic redundancy check can help to discover the occurrence of error by incorporated redundancy in original data. Although more redundant bits provide better error control, they impose more overhead and reduce information transmission rate [1, 2].

Parity-check is one of the earliest and most popular error detection codes. Due to its simplicity, effectiveness, and low encoding/decoding complexity, it has been used in many hardware applications such as small computer system interface, peripheral component interconnect, and memory buses [3]. Single parity-check (SPC) can be useful in low bit error rate (BER) systems, where data transmission can be repeated. The extended versions of SPC such as multidimensional parity-check (MDPC) and low-density parity-check (LDPC) codes provide more powerful error detection and correction capabilities in a parallelisable decoding scheme [4, 5]. They are basically founded upon the initial concept of the parity bit.

Multiple-valued logic (MVL) offers several potential applications for the improvement of signal processing and modern circuit design [6–9]. More information with fewer interconnections can be transmitted in MVL [8], and it also provides better functional capabilities in information process [10]. Several successful MVL circuit methodologies have been suggested in the literature [11–13]. Although reduced speed and noise margin are the drawbacks of MVL, it can still improve the overall performance of a system [14]. For example, the quaternary ROM in the Intel 8087 coprocessor provides 31% area savings compared to a binary ROM [14, 15]. Another example is the current-mode ternary multiplier presented in [16], which has 629 fewer transistors than the binary counterpart. The lack of sufficient MVL algorithms and the two-state behaviour of electronic components are the other reasons for keeping MVL back from prevailing. The transformation must occur for the existing computation methods and coding schemes in order to make MVL systems pervasive in the future.

As far as we know, there are few MVL parity generators in the literature. According to [17], for the purpose of error detection in higher radixes than two, the parity for all of the states other than '0' must be known. This means that $r-1$ parities need to be appended to a k-digit dataword in base r. For example, parities of the logic values '1', '2', and '3' have to be produced separately in quaternary logic. As a result, the final codeword will have $k+3$ digits (k digits of dataword and 3 digits of redundancy). An alternative solution is to append only one parity digit [18]. This way, there will be a single operator responsible for parity; the same as Exclusive-OR (XOR) in binary logic. Moreover, in comparison with [17], fewer gates are required, relative redundancy decreases from $\frac{r-1}{k+r-1}$ to $\frac{1}{k+1}$, and information transmission rate increases significantly.

Some MVL parity generators have been introduced in [18]. According to [18], *commutativity*, *associativity*, and *self-reversibility* are the properties of an MVL parity generator. However, we will show although they bring about some advantages, none of these characteristics is essential for error detection. Instead, there are other fundamental properties that need to be considered carefully. Multi-valued parity generators (MPGs) can be *non-commutative*, *non-associative*, *non-self-reversing*, and still be practical for error detection. Furthermore, ternary parity has originally been introduced in [19]. Another ternary parity generator has been presented in [20] with full discussions about its transistor-level design. However, neither of them adequately explains the properties and detection probabilities.

Additionally, there are several non-binary error-correcting codes over high-order Galois fields (GFs) in the literature [21–26]. The non-binary LDPC (NB-LDPC) is an example with higher performance than binary LDPC. The main differences between the given MPGs in this paper and the previous codes are:

(i) Although NB-LDPC and other non-binary error correction codes provide additional performance gain when symbols with high-order modulations are sent, they have basically and originally presented for dealing with error in a binary system with the aim of achieving higher throughput by reducing the number of computing stages [25]. The current paper is exclusively devoted to MVL communication systems with higher radixes than 2.

(ii) NB-LDPC has a costly and complex computational decoding [27, 28] while the entire error detection procedure by the presented operators is straightforward in both transmitter and receiver.
(iii) The order of a GF is always a prime or a power of a prime number [25] whereas MPGs can be defined for every radix, e.g. base 6.
(iv) Non-binary error correction codes are mostly based on polynomial and GF arithmetic, whereas the new operators are defined by a set of positional rules.
(v) Most of the previous non-binary works using GF aim to provide error correction without presenting any SPC operator. In contrast, the topic of this paper is dedicated to multi-valued SPC for error detection. MPGs are in fact equivalent to binary XOR (as parity generator) in higher radixes.

After the introduction of MPGs, error detection probability for up to five errors is mathematically calculated. Formal verification is also applied to analyse the ability of the operators in detecting errors. The experimental results confirm our analysis. In brief, the main contributions of this paper are as follows:

(i) MPGs for multi-valued SPC are presented.
(ii) The characteristics of the given MPGs are thoroughly studied, and their optional and necessary properties are provided.
(iii) Their detection probabilities in different bases are also analytically and experimentally investigated.

The new multi-valued operators are created based on Sudoku tables with a set of custom rules. Sudoku is a number-placement puzzle, where a 9×9 table is divided into nine 3×3 subgrids. Some of the cells are initialised in advance. The goal is to fill the entire table so that each column, each row, and each subgrid contains all of the digits from 1 to 9. The idea of filling MPG tables in this paper is taken from Sudoku although the rules are not exactly alike.

The rest of the paper is organised as follows. Section 2 defines new MPGs and their characteristics in detail. The creation of MPG tables is explained in Section 3. Section 4 gives a full discussion of the error detection ability of MPG operators using analytical and formal verification methods. Finally, Section 5 concludes the paper.

2 Multi-valued parity generator

SPC is a simple and straightforward procedure. At first, the transmitter calculates parity for a block of data. To do so, a pairwise operator is required to perform successive calculations on dataword digits. In binary logic, XOR is the parity generator. Other operators are required to generate parity in higher radixes. MPGs are generally denoted by $\oplus$ in this paper. Parity calculations in the transmitter side are shown in Fig. 1*a*, where the final codeword is made up of k digits of dataword (m_1 to m_k) and one parity digit (p).

On the other side, the receiver receives a codeword consisting of m_1' to m_k' and p'. In the case of correct delivery, $m_i' = m_i$ (for $1 \leq i \leq k$) and $p' = p$. Parity is calculated once more in the receiver side by the same MPG operator (Fig. 1*b*). We assume that the last MPG returns zero if the received and recalculated parities, p' and p'', are identical. Otherwise, it is a sign of error occurrence.

Regardless of the radix used, the SPC procedure/structure will always resemble Fig. 1 and remain unchanged if the MPG operators are well defined. The whole procedure is similar to what happens in binary logic.

2.1 Necessary properties and requirements

The primary concern is that an MPG operator can provide the ability to detect all single-digit errors. This is equivalent to the capability of XOR in binary logic. Hamming distance (D) between two strings of equal length is the number of positions at which the corresponding symbols are different. A set of valid codewords with the minimum Hamming distance of $d_{\min}$ can always detect $d_{\min}-1$ errors [29]. Therefore, single-digit errors are detectable if $d_{\min} = 2$.

In order to meet this target, MPG must have a couple of necessary properties. In order to clarify them, consider

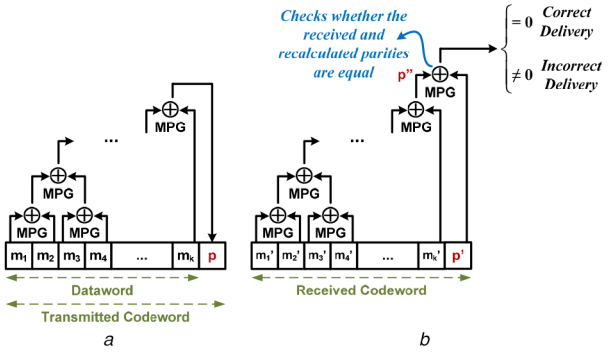

Fig. 1 *SPC procedure and structure by using MPG operators*
(a) Transmitter side, ***(b)*** Receiver side

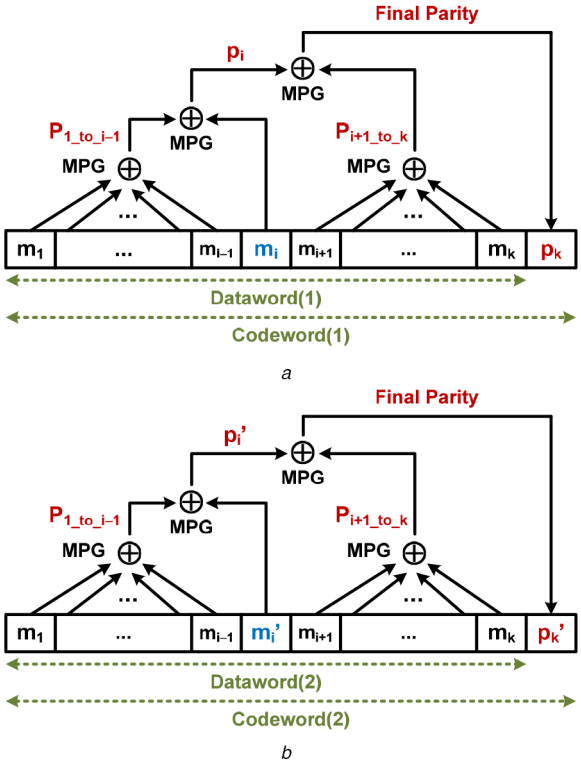

Fig. 2 *Two codewords with the Hamming distance of 2*
(a) Codeword(1), ***(b)*** Codeword(2)

Dataword(1) and *Dataword(2)* in Fig. 2 which have the same data pattern, except one digit (m_i in *Dataword(1)* versus m_i' in *Dataword(2)*). In order to have $d_{\min} = 2$ for the corresponding codewords, the second difference, other than m_i and m_i', must be in their parities. The MPG operator must be able to provide this disparity. Then, *Codeword(1)* and *Codeword(2)* will have two different digits, (m_i versus m_i') and (p_k versus p_k'), while the remaining digits, (m_1 to m_{i-1}) and (m_{i+1} to m_k), are similar. According to Fig. 2:

(i) The identical parts, (m_1 to m_{i-1}) and (m_{i+1} to m_k), generate the same parities, $p_{1\_to\_i-1}$ and $p_{i+1\_to\_k}$, in both datawords.
(ii) m_i and m_i' are different. Consequently, $p_i = \text{MPG}(p_{1\_to\_i-1}, m_i)$ and $p_i' = \text{MPG}(p_{1\_to\_i-1}, m_i')$ must return dissimilar values. The reasonable outcome is that $\text{MPG}(a, b) \neq \text{MPG}(a, c)$ where $a, b, c \in \{0, 1, \ldots, r-1\}$ and $b \neq c$. This is the first necessary property for MPG ((1)).
(iii) Then, p_i and p_i' are different. Therefore, $p_k = \text{MPG}(p_i, p_{i+1\_to\_k})$ and $p_k' = \text{MPG}(p_i', p_{i+1\_to\_k})$ must again return dissimilar values in order to produce different final parities. The reasonable outcome is that $\text{MPG}(b, a) \neq \text{MPG}(c, a)$ where $a, b, c \in \{0, 1, \ldots, r-1\}$ and $b \neq c$. This is the second necessary property for MPG ((2)).

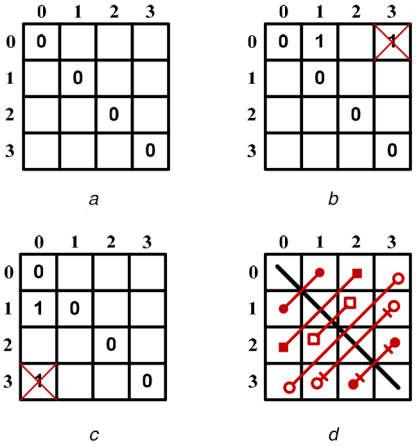

Fig. 3 *MPG table defined by a two-dimensional table*
(a) Visual representation of the third requirement (3), ***(b)*** Visual representation of the first necessary property (1), ***(c)*** Visual representation of the second necessary property (3), ***(d)*** Visual representation of the first optional property (4)

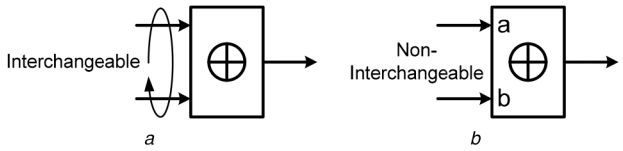

Fig. 4 *MPG gate*
(a) Commutative, ***(b)*** Non-commutative

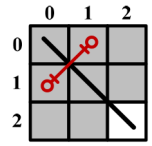

Fig. 5 *MPG operator cannot be commutative in base 3*

The third condition is in fact a requirement based on the assumption shown in Fig. 1*b*. In the receiver side, parity is generated once more. The received and recalculated parities, p' and p'', are compared with each other. Due to the earlier agreement, the operator must return zero in case of their equality. Therefore, we conclude that $\mathrm{MPG}(a, a) = 0$ (for $0 \leq a \leq r-1$). This is the third requirement for our MPG ((3)).

A pairwise operator can visually be expressed by using a two-dimensional table (Fig. 3), where $p_{i,j} = \mathrm{MPG}(i, j)$. MPG tables are exemplified in base 4 although they are extensible to any other radix. The third requirement (3) forces the main diagonal of the matrix to be zero (Fig. 3*a*). The first and second properties ((1) and (2)) imply that no repetitive values are allowed in a row and a column, respectively (Figs. 3*b* and *c*)

$$\mathrm{MPG}(a, b) \neq \mathrm{MPG}(a, c) \tag{1}$$

$$\mathrm{MPG}(b, a) \neq \mathrm{MPG}(c, a) \tag{2}$$

$$\mathrm{MPG}(a, a) = 0 \tag{3}$$

2.2 Optional properties

Thus far, the necessary properties have been settled for the proposed MPG operator. The first and second properties satisfy the minimum Hamming distance for detecting all of the single-digit errors. The third requirement helps the receiver to detect error without ambiguity if the received and recalculated parities are unequal.

There are three other optional properties which might make MPGs more effective. Although MPGs are preferred to be *commutative* (4), it is not a mandatory attribute. If so, input signals are interchangeable (Fig. 4*a*), and it does not matter in what order they are situated. If not, circuit designers must be careful not to

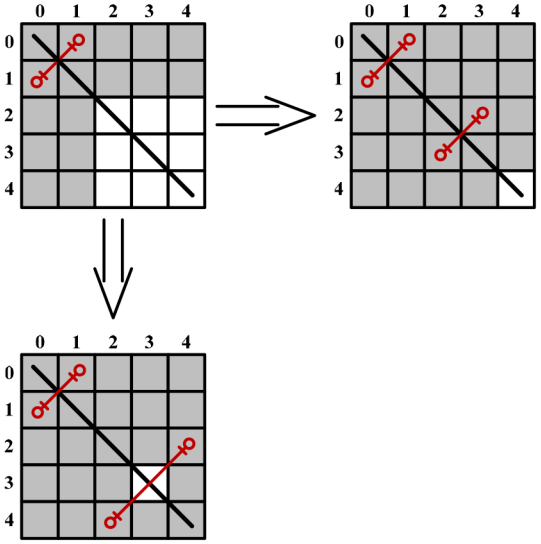

Fig. 6 *MPG operator cannot be commutative in base 5*

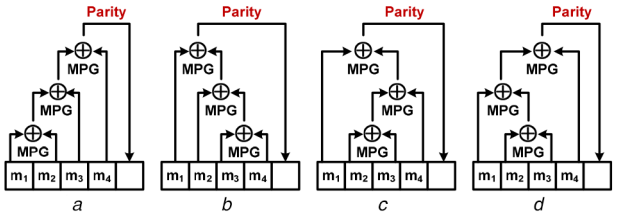

Fig. 7 *MPG placement topologies for a 4-digit dataword*
(a) $\mathrm{MPG}(\mathrm{MPG}(\mathrm{MPG}(m_1,m_2),m_3),m_4)$, ***(b)*** $\mathrm{MPG}(m_1,\mathrm{MPG}(m_2,\mathrm{MPG}(m_3,m_4)))$, ***(c)*** $\mathrm{MPG}(m_1,\mathrm{MPG}(\mathrm{MPG}(m_2,m_3),m_4))$, ***(d)*** $\mathrm{MPG}(\mathrm{MPG}(m_1,\mathrm{MPG}(m_2,m_3)),m_4)$

place MPG gates conversely. For a *non-commutative* MPG (Fig. 4*b*), inputs are not interchangeable. To make an MPG *commutative*, its table must be symmetric (Fig. 3*d*). However, MPG operators cannot be *commutative* in odd bases. This issue is addressed in Figs. 5 and 6 for the bases 3 and 5, respectively. In spite of a little circuit complexity, this limitation does not affect the proper functionality of MPGs in odd bases.

- Base 3 (Fig. 5): On account of symmetry, values of the matrix elements $p_{0,1}$ and $p_{1,0}$ are the same (Let them be ♀ as indicated in Fig. 5; ♀∈{1,2}). According to the first and second properties ((1) and (2)), there are exactly three ♀s in the whole table because repetitive values are not allowed in rows and columns. Therefore, the third ♀ can only be situated in the element $p_{2,2}$, which is on the main diagonal. Nevertheless, this is against the third requirement (3), based on which $p_{2,2}$ is zero.
- Base 5 (Fig. 6): Values of the matrix elements $p_{0,1}$ and $p_{1,0}$ are the same (Let them be ♀ as in Fig. 6; ♀∈{1,2,3,4}). The next two ♀s can be either in ($p_{2,3}$ and $p_{3,2}$) or ($p_{2,4}$ and $p_{4,2}$). Either way, the remaining ♀ must be situated in an element on the main diagonal, $p_{3,3}$ or $p_{4,4}$. Both outcomes are in contrast with the third requirement (3).

$$\mathrm{MPG}(a, b) = \mathrm{MPG}(b, a) \tag{4}$$

The second optional property is *associativity* (5). Figs. 7 and 8 display different MPG placement topologies for a 4-digit dataword. If an MPG operator is *associative*, it does not make a difference in what order it calculates parity. All of them produce identical results. However, the one in Fig. 8 leads to maximum parallelism and the fastest parity calculation approach. If the MPG operator is *non-associative*, transmitter and receiver must follow the same MPG placement topology. They can still use the topology with the fastest parity computation (Fig. 8) whether or not the MPG operator is *associative*. Therefore, *associativity* does not add any specific value to the MPG operators and can be ignored.

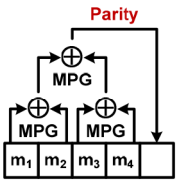

Fig. 8 *MPG placement topology for a 4-digit dataword with maximum parallelism (MPG(MPG(m_1, m_2), MPG(m_3, m_4)))*

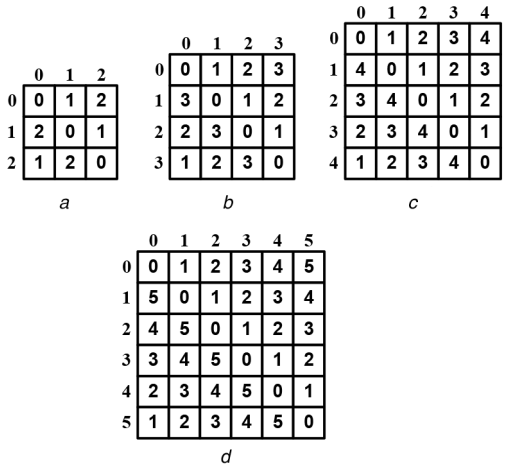

Fig. 9 *MPG operators created by Procedure 1*
(a) $MPG_{(3)}$, ***(b)*** $MPG_{(4)}$, ***(c)*** $MPG_{(5)}$, ***(d)*** $MPG_{(6)}$

The third optional characteristic is *self-reversibility*. MPGs might be *self-reversing* over the columns (6), over the rows (7), or both of them. Nonetheless, this characteristic does not add any specific value to the MPG operators for detecting error. *Self-reversibility* can specifically be exploited for data recovery, and it is particularly useful for error correction [30].

In summary, the optional properties bring about some advantages. Although *commutativity*, *associativity,* and *self-reversibility* are generally preferable, none of them is necessary for the context of error detection. Finally, unlike *commutativity*, the second and third optional properties do not have visual representations within an MPG table

$$\mathrm{MPG}(a, \mathrm{MPG}(b, c)) = \mathrm{MPG}(\mathrm{MPG}(a, b), c) \tag{5}$$

$$\mathrm{MPG}(a, b) = p \Rightarrow \mathrm{MPG}(a, p) = b \tag{6}$$

$$\mathrm{MPG}(a, b) = p \Rightarrow \mathrm{MPG}(p, b) = a \tag{7}$$

3 MPG creation

Filling an MPG table is like solving a Sudoku puzzle; however, with fewer and less complicated rules. Similar to Sudoku, non-repetitive digits are supposed to be inserted in every row and column. On the other hand, there is not any subgrid to be worried about. The third requirement (3) is used to initialise the table (such as Fig. 3*a*). The first and second necessary properties ((1) and (2)) are the cardinal rules, based on which an MPG table is completed. Although it is also possible to consider the first optional property, *commutativity*, as an additional rule in even bases, it is not mandatory.

There are several algorithms such as rule-based [31, 32] and backtracking [32, 33] algorithms to solve a Sudoku table. There are also some other algorithms based on permutations [34], cyclotomic cosets [35], swarm intelligence [36, 37], and genetic algorithm [38]. Nonetheless, MPG creation is much easier than solving Sudoku puzzles since the rules are relatively simpler.

Procedure 1 is an example that provides a row-wise traversing algorithm, which fills the table line by line taking only the necessary properties into account. At first, the main diagonal is initialised by zero (MPG(i, i) = 0). Then, the whole table is filled out by an iterative procedure. The outer and inner loops are

```
\\Table Initialization
For i = 0 to (r–1) {
        MPG(i, i) = 0;
}

\\Table Completion
For i = 0 to (r–1) {
        value = 1;
        For j = (((i+1) mod r)) to (((i+1) mod r)+(r–2)) {
                MPG(i, (j mod r)) = value;
                value++;
        }
}
```

Fig. 10 *Procedure 1: MPG creation in base r*

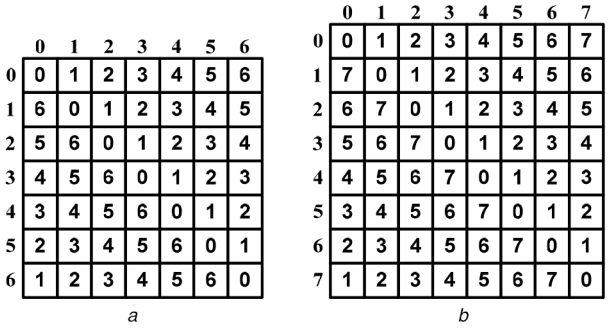

Fig. 11 *MPG operators created by Procedure 1 (continued from Fig. 9)*
(a) $MPG_{(7)}$, ***(b)*** $MPG_{(8)}$

repeated r and $r-1$ times, respectively. They set two variables, i and j, pointing to the blank cells, which are assigned in each row by a *value* increasing from *1* to $r-1$. The assignment starts from the cell next to the one initialised by zero. Fig. 9 shows the $MPG_{(Base)}$ operators which are produced by Procedure 1 (Fig. 10) in bases 3 to 6 ($MPG_{(3)}$ to $MPG_{(6)}$). $MPG_{(7)}$ and $MPG_{(8)}$ are also depicted in Fig. 11. The outcomes are similar to circulant matrices [39]; however, there are several other solutions to create an MPG operator. They have no particular priority over each other as long as the necessary properties are satisfied.

4 Error detection capability

As described before, the MPG operators can reveal single-digit errors. This ability is exemplified in Fig. 12*a* in base 4. At first, the transmitter calculates parity for a 5-digit dataword based on the $MPG_{(4)}$ operator given in Fig. 9*b*. Then, one of the digits changes from '2' to '3' during data transmission over a noisy channel. Afterwards, the receiver recalculates parity, $p'' = 0$, and compares it with the received one, $p' = 3$. Since they are different, the incorrect delivery of data is successfully detected in this scenario.

The MPG operators are even more powerful. Although not all of the error patterns are detectable, MPGs can detect incorrect data delivery if more than one digit flips. Two other scenarios are exemplified in Figs. 12*b* and *c*. In the first one (Fig. 12*b*), the receiver detects the incorrect delivery correctly. In the second example (Fig. 12*c*), however, the receiver fails to reveal error occurrence due to the fact that the received and recalculated parities are similar ($p' = p''$). In this section, the probability of successful error detection is analytically calculated and experimentally measured.

4.1 Error detection probability

Error detection probability for up to five errors is calculated in this subsection. It is assumed that errors can occur independently of one another among all of the digits of a codeword. The probability calculations are based on (8)–(10). According to the first necessary property (1), MPG(a, b) and MPG(a, c) are certainly dissimilar. In other words, the probability of their equivalence is zero (8). In the

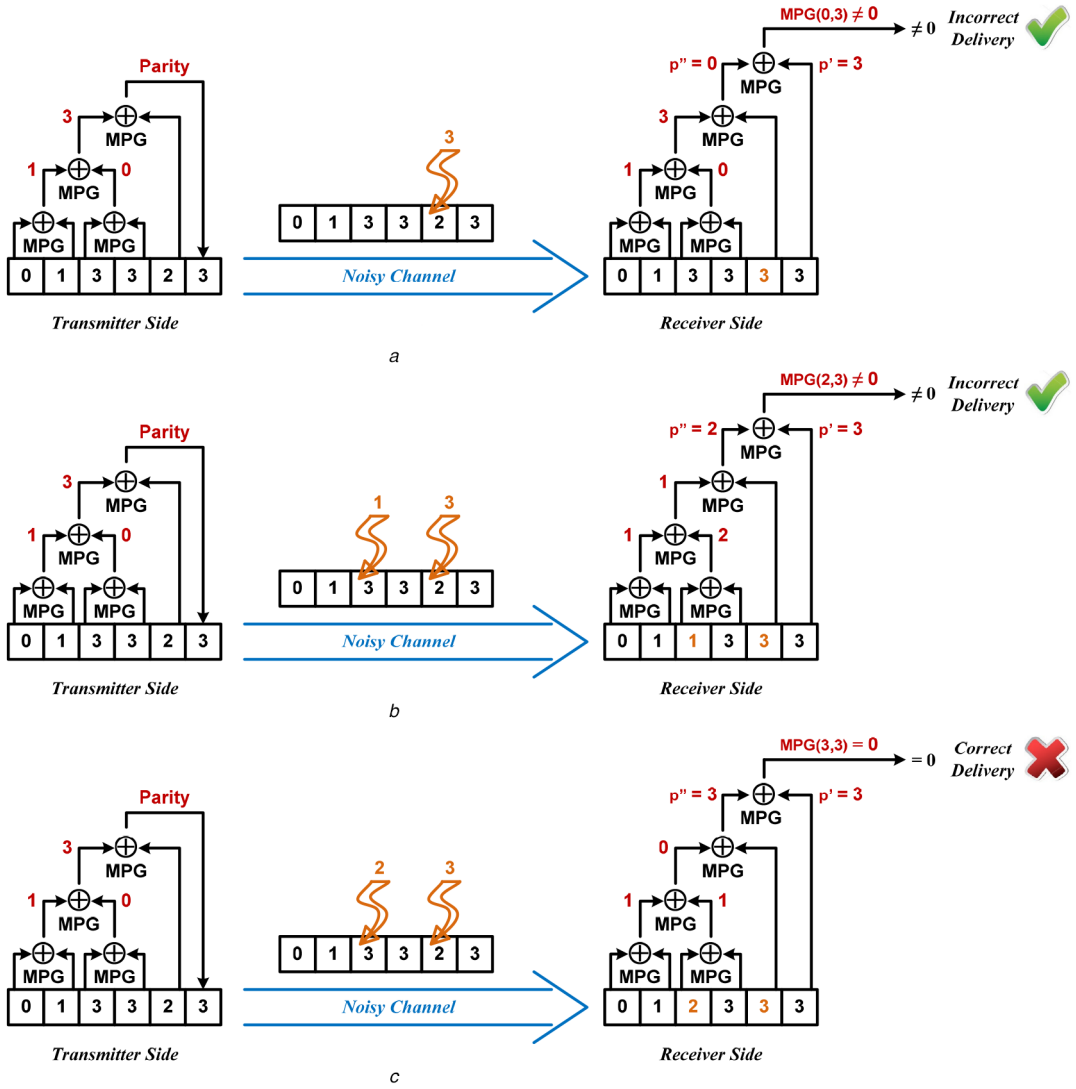

Fig. 12 *Examples of parity-check process*
(a) One error is detected, ***(b)*** Two errors are detected, ***(c)*** Two errors are not detected

same manner, according to the second necessary property (2), MPG(b, a) and MPG(c, a) never generate identical parities (9).

Furthermore, concerning the two-dimensional table of the MPG operator, MPG(a, b) and MPG(c, d), where $a \neq c$ and $b \neq d$, return equal values with $(1/r - 1)$ probability (10). The reason is that $P(\text{MPG}(a, b) = x) = (r/r^2)$ where $x \in \{0,1,\ldots,r-1\}$. Subsequently, $P(\text{MPG}(c, d) = x) = (r - 1/(r-1)^2)$ after the elimination of the ath row and bth column from the MPG table. Eventually, permutation of x is also taken into consideration by the combination of one from r digits

$$P(\text{MPG}(a,b) = \text{MPG}(a,c)) = 0 \tag{8}$$

$$P(\text{MPG}(b,a) = \text{MPG}(c,a)) = 0 \tag{9}$$

$$P(\text{MPG}(a,b) = \text{MPG}(c,d)) = \frac{r}{r^2} \times \frac{r-1}{(r-1)^2} \times \binom{r}{1} = \frac{1}{r-1} \tag{10}$$

Fig. 13 demonstrates two datawords with the same size, which are different in five digits [(m_i versus m_i'), (m_j versus m_j'), (m_w versus m_w'), (m_h versus m_h'), and (m_g versus m_g')]. With reference to Fig. 13, 14 shows a tree diagram where all of the possible outcomes are displayed. The successful and unsuccessful error detection probabilities are indicated by a *Tick* and a *Cross*, respectively. A *Tick* on a level indicates a sequence of events and probabilities from the root to the leaf (*Tick*) which has led to successful error detection. The probabilities on the corresponding branch are multiplied to reach the probability of a *Tick*. Then, the detection probability ($P_{\text{detection}}$) is the sum of probabilities of the *Tick(s)* on a specific level. As a result, $P_{\text{detection}}$ of q errors can be calculated by (11), where n points to the *Ticks* on the target level (level q), and m points to nodes of the corresponding branch from root to the target leaf (n in level q). In fact, n and m characterise q within (11).

The first mismatch between *Dataword(1)* and *Dataword(2)* in Fig. 13 (m_i versus m_i') definitely causes two different parities ($p_i \neq p_i'$). This is the first error, which can certainly be detected (12). The second mismatch might equal the parities again ($p_j = p_j'$) with the probability of $(1/r - 1)$. If so, the two datawords produce similar parities, and it is infeasible to detect an error. If not, different parities are generated once again ($p_j \neq p_j'$), and error occurrence is detectable with the probability shown in (13).

There is an impossible situation in the third level which never happens (Fig. 14). If the second mismatch (m_j versus m_j') generates identical parities ($p_j = p_j'$), the third mismatch (m_w versus m_w') will then produce different parities exclusively ($p_w \neq p_w'$). Therefore, $P(p_w = p_w') = 0$. $P_{\text{detection}}$ for three to five errors can also be calculated by (14)–(16)

$$P_{\text{detection}}(q\text{Error}(s)) = \sum_{n=1}^{\text{No. ticks in level } q} \prod_{m=1}^{q} P^{\text{Node on level } m \text{ leading to n th tick on level } q} \tag{11}$$

$$P_{\text{detection}}(1\text{Error}) = 1 \tag{12}$$

$$P_{\text{detection}}(2\text{Errors}) = 1 - \frac{1}{r-1} \tag{13}$$

$$P_{\text{detection}}(3\text{Errors}) = \left(1 - \frac{1}{r-1}\right)^2 + \frac{1}{r-1} \tag{14}$$

$$P_{\text{detection}}(4\text{Errors}) = \left(1 - \frac{1}{r-1}\right)^3 + \frac{2}{r-1}\left(1 - \frac{1}{r-1}\right) \tag{15}$$

$$P_{\text{detection}}(5\text{Errors}) = \left(1 - \frac{1}{r-1}\right)^4 + \frac{3}{r-1}\left(1 - \frac{1}{r-1}\right)^2 + \left(\frac{1}{r-1}\right)^2 \tag{16}$$

Fig. 15 displays detection probability versus the number of errors in different radixes. It shows that the presented MPG operators are generally capable of detecting errors with high probability. In addition, detection probability increases in higher radixes. For instance, errors are detected with ~85% probability in base 8. Another noticeable observation is that it is less likely to detect two errors compared to when there are more, e.g. three errors. Finally, (12)–(16) show that detection probability is independent of the size of dataword.

The error detection probabilities presented in (13)–(16) are directly derived from (11) and Fig. 14. Fig. 14 is a binary tree representing all of the successful and unsuccessful error detection scenarios and probabilities for up to five errors. The binary tree can be expanded for a larger number of errors. Equation (11) is simply correct because $\sum_{n=1}^{\text{No. ticks in level } q} \prod_{m=1}^{q} P + \sum_{n=1}^{\text{No. crosses in level } q} \prod_{m=1}^{q} P = 1$. In other words, for every level we have $P_{\text{SuccessfulDetection}} = 1 - P_{\text{UnsuccessfulDetection}}$. We will also verify the equations by means of a formal verification method in the next subsection.

4.2 Formal verification

A model checking formal verification method is used here to verify the probability equations (13)–(16). A labelled transition system (LTS) is utilised for behavioural modelling of the MPG operator in base 3 ($\text{MPG}_{(3)}$). The state space exploration for the behavioural

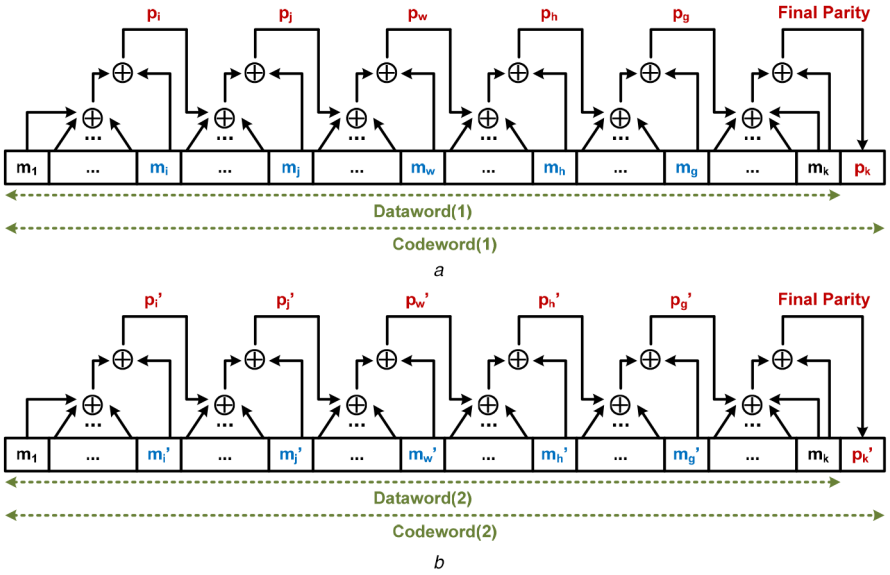

Fig. 13 *Two datawords with five mismatches*
(a) Dataword(1), ***(b)*** Dataword(2)

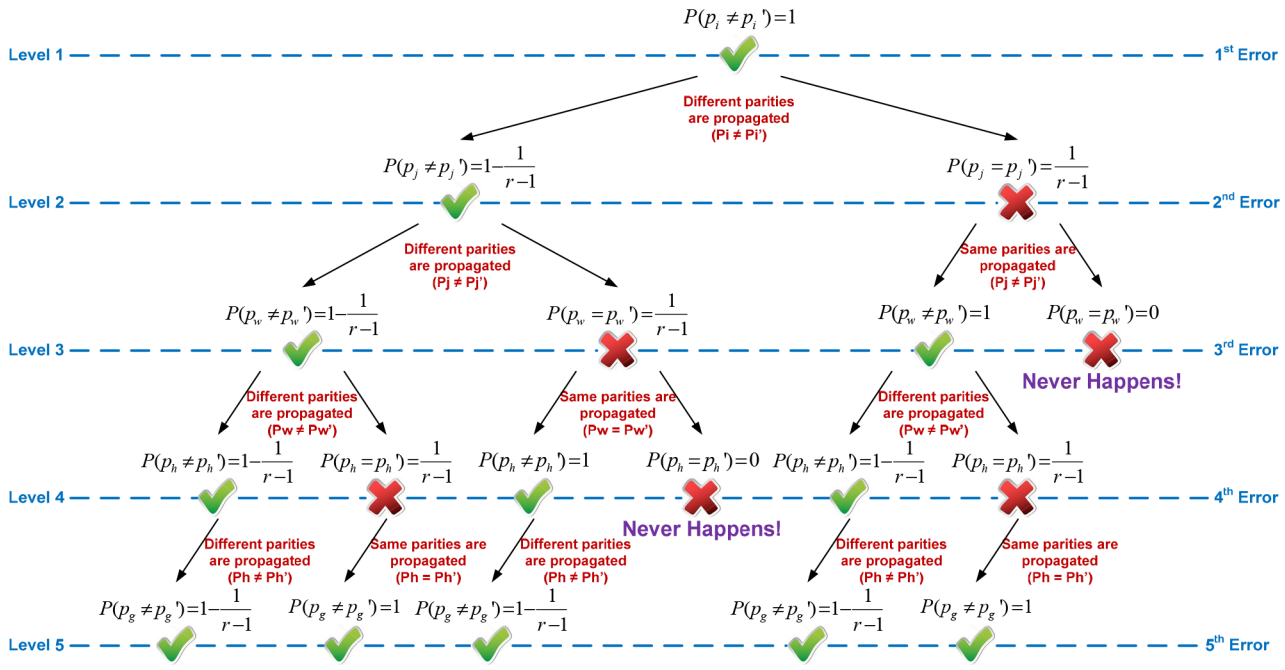

Fig. 14 *Tree diagram of the possible outcomes for up to five errors and their detection probabilities (with reference to Fig. 13)*

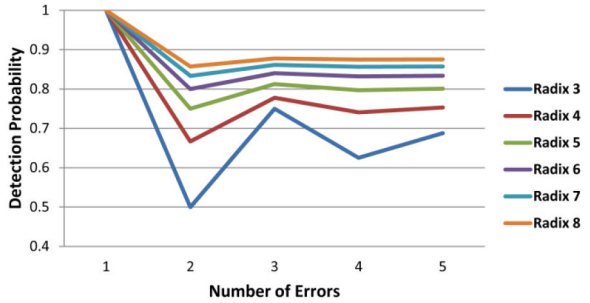

Fig. 15 *$P_{detection}$ versus number of errors in different radixes; obtained from our analytical results (12)–(16)*

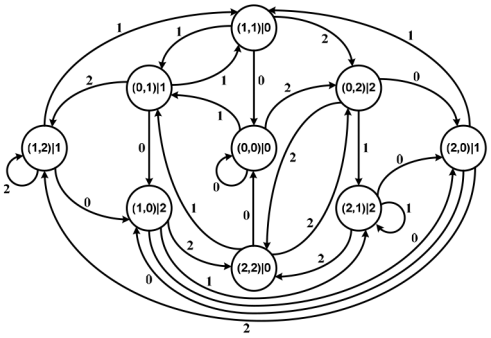

Fig. 16 *LTS-based structure for the MPG operator model in base 3*

model is depicted in Fig. 16. Base 3 is exemplified here for the sake of simplicity; however, LTS models for MPGs in higher radixes are also achieved. The definition for formal specification model is as follows.

Definition 1: The parity error detection model in base 3 is mapped on a specific LTS, which is represented by a 4-tuple [40]: $MPG_{(3)} = (\mathcal{S}, s, E, \mathcal{R})$, where:

- $\mathcal{S}$ is a set of nine states, $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9\}$. A state represents a record of $MPG_{(3)}$ truth table (Table 1) with the structure of $(x,y) \mid MPG_{(3)}(x, y)$, where x and y are the input arguments for the MPG operator. All of the states are shown in Table 1.
- The initial state is s ($s \in \mathcal{S}$). Note that there is not a specific initial state, and the state machine can start from an arbitrary state depending on the first two digits of dataword/codeword.
- All of the events are denoted by E. An event is in fact the next unprocessed digit. Thus, $E \in \{0, 1, 2\}$. In addition, an event shows a transition relation between two states.
- A transition relation, $\mathcal{R}$, is presented by the existing states and events: $\mathcal{R} \subseteq \mathcal{S} \times E \times \mathcal{S}$. The relation $s_1 \xrightarrow{e} s_2$ ($s_1, s_2 \in \mathcal{S}$ and $e \in E$) is applied to the state that $(s_1, e, s_2) \in \mathcal{R}$. A transition relation between two states and the related event is defined with $(x_1, y_1) \mid MPG_{(3)}(x_1, y_1) \xrightarrow{y2} (MPG_{(3)}(x_1, y_1), y_2) \mid MPG_{(3)}(MPG_{(3)}(x_1, y_1), y_2)$. It actually forms a recursive function where the generated parity in a state becomes the first input argument of the next state.

After specifying the formal concepts in the LTS structure, the behavioural model is translated into a symbolic model verifier (SMV) code to verify the system correctness in the NuSMV model checker [41]. The flowchart of Fig. 17 demonstrates the code sequence diagram. At first, a random dataword is generated. Then, errors are deliberately inserted in some accidental positions. Finally, p' and p'' are compared with each other.

In the NuSMV, flatten hierarchical method generates a state space graph for the MVL model to analyse the error detection reachability (EDR). Fig. 18 demonstrates EDR versus the number of errors in different radixes. Each run contains 10000 128-bit

Table 1 States and events of $MPG_{(3)}$ table model

x	y	$MPG_{(3)}(x, y)$	State
0	0	0	$S_1 = (0,0)\|0$
0	1	1	$S_2 = (0,1)\|1$
0	2	2	$S_3 = (0,2)\|2$
1	0	2	$S_4 = (1,0)\|2$
1	1	0	$S_5 = (1,1)\|0$
1	2	1	$S_6 = (1,2)\|1$
2	0	1	$S_7 = (2,0)\|1$
2	1	2	$S_8 = (2,1)\|2$
2	2	0	$S_9 = (2,2)\|0$

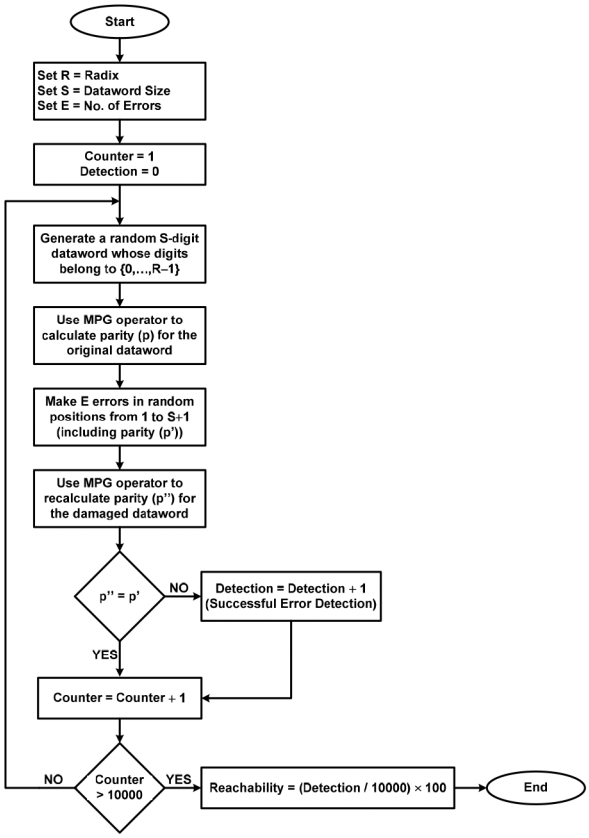

Fig. 17 *SMV code sequence diagram*

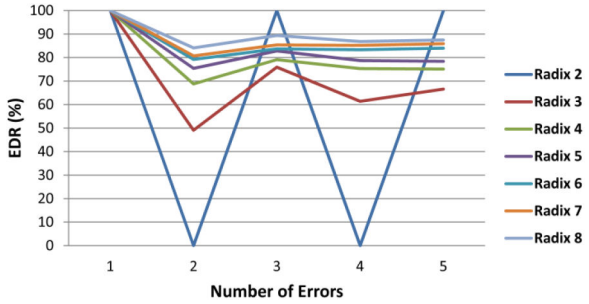

Fig. 18 *EDR (%) versus number of errors in different radixes; obtained from our experimental results*

random data string samples. The experimental results in Fig. 18 confirm our analytical results in Fig. 15. Fig. 18 also provides a comparison between binary and MPGs. Binary XOR can always detect an odd number of errors (EDR = 100%), but it can never detect an even number of errors (EDR = 0%).

The same analysis is also carried out for datawords with different sizes (8-, 16-, 32-, 64-, and 128-bit datawords). The experimental results are displayed in Fig. 19. The curves are almost

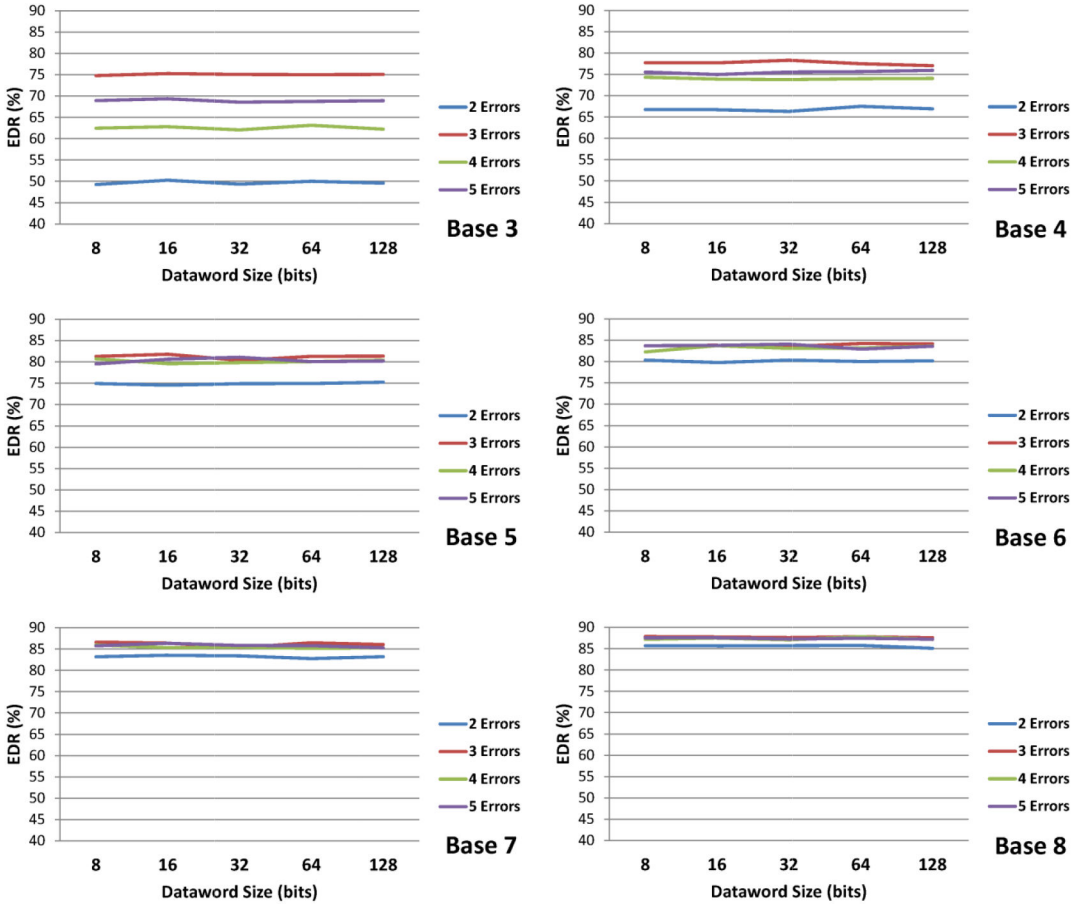

Fig. 19 *EDR (%) versus dataword size (8, 16, 32, 64, and 128 bits) and number of errors; obtained from our experimental results*

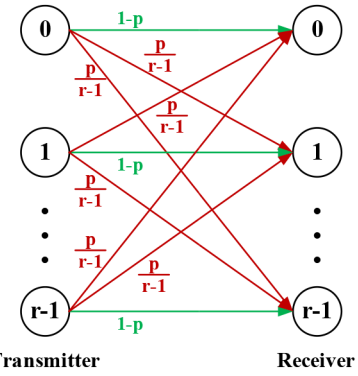

Fig. 20 *NBSC model in radix r with probability of error p [43]*

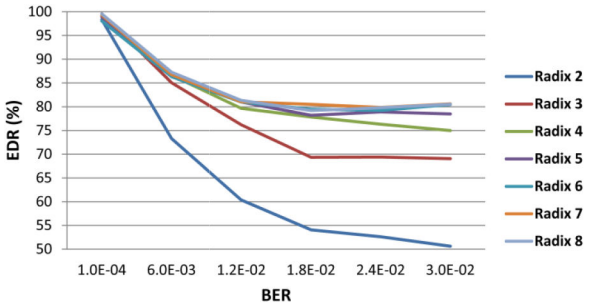

Fig. 21 *EDR (%) versus BER; obtained from our simulation results over a NBSC*

straight lines, implying that error detection probability is independent of dataword size. The negligible fluctuations are due to the probabilistic nature of the proposed MPGs. For instance, as exemplified before in Fig. 12, a particular pattern of two errors is successfully detected (Fig. 12*b*), while another one with the same number of errors is not (Fig. 12*c*). Therefore, EDRs for different dataword sizes with a fixed number of errors are not exactly alike, but very close to each other due to the *law of large numbers*. We have considered a large number of trials, 10,000 dataword samples, so that the obtained results become close to the expected values. Furthermore, EDR values in Figs. 18 and 19 (simulation results) entirely match with EDR values in Fig. 15 (analytical results).

Eventually, Monte Carlo simulations are taken into consideration to test and model the presented MPG operators over a non-binary symmetric channel (NBSC) [42, 43], where errors are independent and every bit transmitted has a fixed probability of error p. It is a natural probabilistic extension of the binary symmetric channel [42]. The error probability model of an NBSC in radix r is shown in Fig. 20 [43]. The conditional probabilities of such a channel are demonstrated in (17) and (18), where transmitter (T) sends a digit (m), and receiver (R) gets either a right (m) or wrong (m') digit

$$P[R = m | T = m] = 1 - p \tag{17}$$

$$P[R = m' \Big| T = m] = \frac{p}{r-1} \tag{18}$$

The percentage of EDR versus BER is depicted in Fig. 21, in which BER and p as defined in Fig. 20 are equivalent. As it is clear, the proposed MPGs provide higher EDRs than binary XOR. In lower BERs, mostly single-digit errors happen, which are always detectable in base 2. However, in higher BERs, where there is a 50–50 chance of an odd and even number of errors, EDR for binary XOR approaches 50%.

5 Conclusion and future works

Some parity-generating operators, which are *non-commutative*, *non-associative*, and *non-self-reversing*, have been given in this paper. None of these characteristics is essential to create an MPG operator in the context of error detection. Instead, there are three

other necessary properties and requirements for the creation of MPGs. Afterwards, some MPG operators in bases 3 to 8 have been presented based on the idea of Sudoku puzzles. However, MPG tables are not unique, and one can define several other MPG operators by considering permutations and other Sudoku-like solving algorithms. This gives circuit designers a chance to reach the most efficient design in terms of hardware implementation. MPG operators can be realised in transistor level [20] or with look-up tables (LUTs) [44].

We have made the following observations based on our theoretical analyses and simulation results:

(i) The proposed MPGs can successfully detect all of the single-digit errors, and even a larger number of errors with high probability ($\geq 50\%$).
(ii) Detection probability increases in higher radixes. Nevertheless, high-order MPGs need more complex encoding and decoding procedures. Larger LUTs or more complicated circuits are required to implement MPGs in higher bases.
(iii) Detection probability does not necessarily decrease as the number of errors increases. In spite of its apparent irregularity, the same thing actually happens in binary logic as well. For example, binary XOR can detect three errors, but not two. Error detection in MVL is a matter of probability. With similarity to binary logic, three errors in MVL are detected with higher probability than two.
(iv) Binary SPC can detect all of the errors caused by odd numbers of error bits. However, the MPG counterparts do not have the same capability with 100% detection probability. On the other hand, binary XOR fails to reveal incorrect data delivery when an even number of errors occurs in a data stream. This time, MPGs gain the upper hand and can detect an error with more than 50% probability. Generally, MPG operators are more powerful in detecting errors than binary XOR, especially in channels with high BERs.
(v) Detection probability is independent of the size of dataword.

Finally, we point out some open problems and possible future works regarding the topic of this paper:

(i) The proposed MPG operators can be realised in transistor level or with LUTs, and their implementation complexities can also be calculated. On the one hand, MPGs provide higher error detection probabilities than binary XOR. On the other hand, it is evident that more transistors and larger LUTs are needed to realise MPGs.
(ii) It is possible to define different MPG tables in a specific radix for the purpose of error detection. Which MPG table is more appropriate for circuit realisation is still an open question.
(iii) The given MPGs are in fact equivalent to binary parity generator in higher radixes. However, their relation to GF arithmetic is another open question. It will be interesting to study whether or not the proposed MPGs are actually the XOR version of a GF.
(iv) The analogy of MVL parity-check matrices (PCMs) as well as their Hamming weight spectrums to their binary counterpart can further be investigated. It would be interesting to create a whole class of error detection/correction for MVL data based on the proposed MPGs.
(v) The presented MPG operators can be applied to the formulation of PCM of NB-LDPC. It is also possible to create MDPC codes for the purpose of error correction.

6 Acknowledgment

The authors thank the editorial team and anonymous reviewers whose comments and suggestions helped them to improve the quality of their paper.

7 References

[1] Shannon, C.E.: 'A mathematical theory of communication', *Bell Syst. Tech. J.*, 1948, **XXVII**, (3), pp. 379–423
[2] Hamming, R.W.: 'Error detecting and error correcting codes', *Bell Syst. Tech. J.*, 1950, **29**, (2), pp. 147–160
[3] Kamat, R.K., Shinde, S.A., Shelake, V.G.: '*Unleash the system On chip using FGPAs and handel C*' (Springer, Netherlands, 2009)
[4] Gallager, R.: 'Low-density parity-check codes', *IRE Trans. Inf. Theory*, 1962, **8**, (1), pp. 21–28
[5] MacKay, D.J.C., Neal, R.M.: 'Near Shannon limit performance for low density parity check codes', *Electron. Lett.*, 1997, **33**, (6), pp. 457–458
[6] Karmakar, S., Karmakar, S.: 'Improved image processing in quaternary logic'. IEEE National Aerospace and Electronics Conf., Dayton, USA, July 2018, pp. 298–302
[7] Dubrova, E.: 'Multiple-valued logic in VLSI: challenges and opportunities'. Proc. NORCHIP'99, Oslo, Norway, November 1999, pp. 340–350
[8] Chowdhury, A.K.: 'Efficient methods for synthesis of multiple-valued logic'. M.Sc. Thesis, Curtin University, 2014
[9] Gaudet, V.: 'A survey and tutorial on contemporary aspects of multiple-valued logic and its application to microelectronic circuits', *IEEE J. Emerg. Sel. Top. Circuits Syst.*, 2016, **6**, (1), pp. 5–12
[10] Steinbach, B.: '*Problems and new solutions in the boolean domain*' (Cambridge Scholars Publishing, UK, 2016)
[11] Lin, S., Kim, Y.-B., Lombardi, F.: 'CNTFET-based design of ternary logic gates and arithmetic circuits', *IEEE Trans. Nanotechnol.*, 2011, **10**, (2), pp. 217–225
[12] Mirzaee, R.F., Nikoubin, T., Navi, K.*, et al.*: 'Differential cascode voltage switch (DCVS) strategies by CNTFET technology for standard ternary logic', *Microelectron. J.*, 2013, **44**, (12), pp. 1238–1250
[13] Moaiyeri, M.H., Mirzaee, R.F., Doostaregan, A.*, et al.*: 'A universal method for designing low-power carbon nanotube FET-based multiple-valued logic circuits', *IET Comput. Digit. Tech.*, 2013, **7**, (4), pp. 167–181
[14] Rakos, B.: 'Multiple-valued computing by dipole-dipole coupled proteins', *Int. J. Circuit Theory Appl.*, 2019, **47**, (8), pp. 1357–1369
[15] Stark, M.: 'Two bits per cell ROM'. Proc. COMPCON, Washington, DC, USA, January 1981, pp. 209–212
[16] Moradi, M., Mirzaee, R.F., Navi, K.: 'Ternary versus binary multiplication with current-mode CNTFET-based K-valued converters'. 46th Int. Symp. Multiple-Valued Logic, Sapporo, Japan, May 2016, pp. 17–22
[17] Millas, R.J., Weakley, T.L.: 'Multi-valued logic parity'. IBM, Singapore, Singapore, 1989, p. 1
[18] Lablans, P., Township, M.: 'Multi-valued check symbol calculation in error detection and correction', US Patent, 2007, Patent no. US 2007/0258516 A1
[19] Porat, D.I: 'Three-valued digital systems', *Proc. IEE*, 1969, **116**, (6), pp. 947–954
[20] Mirzaee, R.F., Daliri, M.S., Navi, K.*, et al.*: 'A single parity-check digit for one trit error detection in ternary communication systems: gate-level and transistor-level designs', *J. Multiple-Valued Logic Soft Comput.*, 2017, **29**, pp. 303–326
[21] Murai, K., Usui, M.: 'Error code correction devices having a galois arithmetic unit', US Patent, 1988, Patent no. US5020060A
[22] Chen, C.L.: 'Non-binary error correcting code for correcting single symbol errors and detecting double bit errors', US Patent, 2017, Patent no. WO2018218125A1
[23] Ghayoor, F.: 'Non-binary compound codes based on single parity-check codes', Ph.D. Thesis, University of KwaZulu-Natal, 2013
[24] Davey, M.C., MacKay, D.J.C.: 'Low density parity check codes over GF(q)'. Information Theory Workshop, Killarney, Ireland, June 1998, pp. 70–71
[25] Abdmouleh, A.: 'Non-binary LDPC codes associated to high order modulations', Ph.D. Thesis, University of Bretagne Sud, 2017
[26] Song, S., Tian, J., Lin, J.*, et al.*: 'A novel low-complexity joint coding and decoding algorithm for NB-LDPC codes'. IEEE Int. Symp. Circuits and Systems, Sapporo, Japan, May 2019, pp. 1–5
[27] Stark, M., Bauch, G., Lewandowsky, J.*, et al.*: 'Decoding of non-binary LDPC codes using the information bottleneck method'. IEEE Int. Conf. Communications, Shanghai, China, May 2019, pp. 1–6
[28] Birkhoff, G., Mac Lane, S.: '*A survey of modern algebra*' (CRC Press, UK, 1998)
[29] Robinson, D.J.S.: '*An Introduction to abstract algebra*' (Walter de Gruyter, Germany, 2003)
[30] Lablans, P., Township, M.: 'Symbol error correction by error detection and logic based symbol reconstruction', US Patent, 2011, Patent no. US 8,046,661 B2
[31] Saxena, R., Jain, M., Yaqub, S.M.: 'Sudoku g solving approa through parallel processing'. Proc. 2nd Int. Conf. Computational Intelligence and Informatics, Hyderabad, India, 2018, vol. 712, pp. 447–455
[32] Jussien, N.: '*A to Z of Sudoku*' (Wiley, USA, 2007)
[33] Berggren, P.: 'A study of Sudoku solving algorithms', B.Sc. Thesis, KTH University, 2012
[34] Jana, S., Maji, A.K., Pal, R.K.: 'A novel Sudoku solving technique using column based permutation'. Int. Simp. Advanced Computing and Communication, Silchar, India, September 2015, pp. 71–77
[35] Najafian, M., Tadayon, M.H., Esmaeili, M.: 'Construction of strongly mutually distinct Sudoku tables and solid Sudoku cubes by cyclotomic cosets', *IEEE Trans. Games*, 2020, **12**, (1), pp. 54–62
[36] Mantere, T.: 'Improved ant colony genetic algorithm hybrid for Sudoku solving'. 3rd World Congress on Information and Communication Technologies, Hanoi, Vietnam, December 2013, pp. 274–279
[37] Pacurib, J.A., Seno, G.M.M., Yusiong, J.P.T.: 'Solving Sudoku puzzles using improved artificial bee colony algorithm'. 4th Int. Conf. Innovative Computing, Information and Control, Kaohsiung, Taiwan, December 2009, pp. 885–888
[38] Chel, H., Mylavarapu, D., Sharma, D.: 'A novel multistage genetic algorithm approach for solving Sudoku puzzle'. Int. Conf. Electrical, Electronics, and Optimizing Techniques, Chennai, India, March 2016, pp. 808–813
[39] Tanner, R.M., Sridhara, D., Sridharan, A.*, et al.*: 'LDPC block and convolutional codes based on circulant matrices', *IEEE Trans. Inf. Theory*, 2004, **50**, (12), pp. 2966–2984

[40] Souri, A., Rahmani, A.M., Navimipour, N.J., *et al.*: 'A symbolic model checking approach in formal verification of distributed systems', *Human-Centric Comput. Inf. Sci.*, 2019, **9**, (1), pp. 1–27
[41] Cimatti, A., Clarke, E., Ginuchiglia, F., *et al.*: 'NuSMV: A new symbolic model checker', *Int. J. Softw. Tools for Technol. Transfer*, 2000, **2**, (4), pp. 410–425
[42] Bleichenbacher, D., Kiayias, A., Yung, M.: 'Decoding of interleaved reed solomon codes over noisy data'. Int. Colloquium Automata, Languages, and Programming, Berlin, Germany, June 2003, pp. 97–108
[43] Zrelli, Y., Gautier, R., Rannou, E., *et al.*: 'Blind identification of code word length for non-binary error-correcting codes in noisy transmission', *EURASIP J. Wirel. Commun. Netw.*, 2015, **2015**, (43), pp. 1–16
[44] Cunha, R., Boudinov, H., Carro, L.: 'Quaternary look-up tables using voltage-mode CMOS logic design'. 37th Int. Symp. Multiple-Valued Logic, Oslo, Norway, May 2007, pp. 56–61