Journal of Electronic Testing (2021) 37:243–254
https://doi.org/10.1007/s10836-021-05942-4

Tolerating Soft Errors with Horizontal-Vertical-Diagonal-N-Queen (HVDNQ) Parity

Muhammad Sheikh Sadi[1] · **Sumaiya Sumaiya**[1] · **Mouly Dewan**[1] · **Atikur Rahman**[1]

Received: 15 November 2020 / Accepted: 25 March 2021 / Published online: 3 May 2021

Abstract
A new error detection and correction methodology, defined as Horizontal-Vertical-Diagonal-N-Queen-Parity (HVDNQ), is proposed in this paper. This approach relies on five different types of parities: horizontal parity, vertical parity, forward diagonal parity, backward diagonal parity, and queen parity. This method works on an N X N cell area and can correct multi-bit upsets. The experimental analysis validates the effectiveness of the proposed methodology by comparing its efficiency with existing methodologies. In different varieties of error patterns such as equilateral triangle, pentagon, hexagon etc., the capability of error detection and correction of HVDNQ is much better than existing methods.

Keywords Horizontal Parity · Vertical Parity · Diagonal Parity · N-queen Parity · Soft Error Tolerance

1 Introduction

Systems complexity is increasing day by day with the advancement of modern circuitry. As a result, systems have become more prone to soft errors [1]. Embedded systems, having high complexity, face casualty in this matter. Data are becoming more susceptible to soft errors (i.e., single-event transients - SET) while they are being transferred from sender to receiver. Similarly, data are also becoming more susceptible to soft errors (single-event upsets - SEU) while being stored in memory elements. With progressive innovation, the scaling of complex systems based on circuitry is expanding exponentially. The structure of a transistor on a chip in current dimensions has elements only a few hundred atoms wide. Hence, very little energy is needed to change the state of the transistor and thus causing transient faults in the embedded systems' circuits [2]. Transient errors are highly responsible for bitflips in systems, which is a major reliability issue in real-time systems [3]. In practical applications, such as avionic systems or nuclear power plants, a single bit error can cause disastrous changes [4]. Nonstop downscaling of CMOS innovations has come about in clock frequencies coming to different GHz; supply voltage decreasing below one-volt level, and load capacitances of circuit nodes dropping to femtofarads [5] are expanding the dangers in embedded systems [6]. The risks of soft errors will become more of an issue as the line widths are needed to be reduced more since in that situation a little disturbance in the compact circuit may create a voltage glitch.

Manufacturing companies are adding more complex features to upcoming processors. To reach this goal, circuits are becoming more compact. Additionally, supply voltage is being reduced to minimize power. As a result, the natural resilience of chips to soft errors is continuously decreasing [7].

The exceptionally high level of complexity and the fact that the software and hardware are so intricately linked in an embedded system means is creating the situation where unexpected voltage fluctuation in internal structure can change the system's state at any time [8]. Specifically, systems like pacemaker; unmanned aerial vehicles, electronic hostile environments where a system cannot afford a malfunction; nuclear power plants where a single failure may cause severe destruction are very much concerned about the risks of soft error [5].

Responsible Editor: F. L. Vargas

✉ Muhammad Sheikh Sadi
sadi@cse.kuet.ac.bd

Sumaiya Sumaiya
sumaiya0069@gmail.com

Mouly Dewan
mouly.dewan@gmail.com

Atikur Rahman
Rahman1907511@stud.kuet.ac.bd

[1] Khulna University of Engineering & Technology, Khulna, Bangladesh

Some approaches tried to tolerate soft errors at their best. However, in most of the cases (as far we have reviewed), the system is still at risk in key areas [9]. Further, in the case of hardware and software dual redundancy-based approaches, these methods incur overheads for synchronizing identical threads and performance degradation for redundant components [10]. Hardware-based strategies based on duplication regularly endure from a high area, time, and power overheads [11].

Several error correction coding techniques already exist. Among them, parity codes, N-Dimensional parity code [12], BCH code [13], Golay code [14] are notable. The error correction coverage, information overhead, and complexity of these methods vary from adopting various techniques. Most of the techniques have large information overhead, and/or circuit complexity. Besides these, the error correction capability of those techniques still require more advancements.

Hence, this paper proposes an enhancement of our previous HVDQ [15] and HVD7Q [16], to advance the soft error tolerance strategies. In the present approach, we have used horizontal, vertical, diagonal, and n (here we used a variable rather than a fixed value) queen parity. The NQueen is a problem where N queens on an N × N chessboard are to be placed so that no two queens attack each other. Then generate all possible configurations of queens on board and print a configuration that satisfies the given constraints. So, for a given N, we have generated all possible solutions and each parity for these solutions individually. These parity bits are used with other parity bits (horizontal, vertical and diagonal parity) to detect and correct errors. HVDNQ removes the limitations of the HVDQ method by adding the variation of the queen parity. Here, a high-level detection and correction method has been applied for all solutions of the queen parity. In the queen parity scheme, we have experimented with 7 to 20 queen parities. HVDNQ has covered 5-bit error detection and 14-bit error detection by using 7 queens to 20 queens with horizontal, vertical, and diagonal parity.

The remaining part of the paper is organized as follows. Section 2 discusses related work in brief. The proposed methodology is discussed in Sect. 3, while Sect. 4 depicts a case study to illustrate the proposed methodology. Experimental analysis is shown in Sect. 4. Finally, conclusions are drawn in Sect. 5.

2 Related Work

There are several existing approaches to tolerate soft errors. Few remarkable approaches among them are outlined shortly as follows.

Sakib et al. [15] depicted a modified approach of HVD [21]. They defined their approach as HVDQ which functions in 5 dimensions. A new dimension named as Queen is included with the other 4 dimensions: H, V, FD, and BD. The key idea of their paper was to use 8 queen solutions for error detection and correction. Our previous paper HVD7Q [16] also added five different types of parities. It added seven queen parity with H, V, FD, and BD. However, it could detect and correct up to 4-bit errors only.

Pflanz et al. [17] proposed a 5-bit error correction coding technique. It used parity bits in three directions. All combinations of 5-bit errors are not tolerable by using this method. It also ignored the probability of error in the parity bits.

Horizontal, vertical, and diagonal parity-based approach for soft error tolerance was shown by Shanna and Vijayakumar [18]. However, the proposed system can detect and correct maximum 5-bit errors.

Anne et al. [19] proposed a methodology where information bits are structured to achieve a cube of different layers. The external surfaces are composed of pieces of equality. This method provides a design in a three-dimension structure and by this method, errors of maximum 7 bits can be detected and errors of maximum 2 bits can be corrected.

Aflakian et al. [20] proposed two soft error tolerant approaches. For the first approach, the system can detect maximum 7-bit errors and can correct up to 4-bit errors. This system utilized 4 dimensions such as horizontal, vertical, forward diagonal, and backward diagonal (H, V, FD, and BD). For the second approach, the system created cuboids of data bits and this system is able to detect a maximum of 15 erroneous bits and to correct a maximum of 4 bits.

Kishani et al. [21], in their paper, illustrated an approach that utilized 4 dimensions such as H, V, FD, and BD. This approach can correct maximum 3-bit errors. However, it neglected several error patterns for which it could not detect and correct errors.

Matrix Code is another popular error-tolerant method that is mostly memory protection oriented [22]. In this method, at first total data bits are divided into a matrix form followed by the application of hamming code for the generation of check bits. Through matrix codes, single and double bit errors can be detected and corrected. However, for double bit errors in the same column, the method cannot detect and correct errors.

Silva et al. [23] proposed an extended matrix region selection code (EMRSC) which increased correction capability and reduces the number of generated redundant bits by using data matrix regions. This method also generates a tradeoff between area and power overheads and reliability degree. However, this method could correct around 60% errors considered in all scenarios.

3 The Proposed HVDNQ Methodology

Our proposed methodology is defined as HVDNQ where N is varying from 7 to 20. In the N Queen problem, "N" is the number of chess queens on an N×N chessboard as described in Sect. 1. The number of queen i.e. N is an integer number among 7 to 20 to detect and correct the erroneous bits.

It is comprised of several steps such as Calculate Horizontal and Vertical Parity, Calculate Forward and Backward Diagonal Parity, Calculate Queen Parity, Finding Mismatched Parity Bits, Mark Candidate Bits, Refine Candidate Bits, and Flag the Errors and Error Correction. The flow diagram of the proposed HVDNQ methodology is illustrated in Fig. 1, where the necessary steps are visualized. In here, we have chosen a typical virtual example of information which might be erroneous while Sender is sending data to Receiver and how these errorrs are being detected and corrected using the proposed method.

The proposed methodology works in H, V, FD, BD, and N Queen parities. It calculates parity in all 5 different types of parities and compares sent parity with received parity to find mismatched parity (regenerate parity bits). The candidate bits are generated from these parity bits and finally these are refined to flag errors. The details of these steps are outlined shortly as follows.

The horizontal and vertical parity can be calculated by choosing any method from the Even/Odd Parity generation shown in Fig. 2. The detailed process to generate these parity bits is shown by using pseudocode as illustrated in Fig. 3.

3.1 Forward and Backward Parity Calculation

Diagonal parity bits are calculated from one corner of a matrix to another corner and we calculated the number of inclining parity bits. There are two sorts of diagonal parity calculations to be carried out. One is forward parity calculation, and another one is backward parity calculation.

The forward diagonals take the direction from the top right to bottom left and the backward diagonals are from top left to bottom right. For forward diagonal parity, as shown in Fig. 2, the 1st forward diagonal has only one cell as [0], [0], the 2nd diagonal has a two cells as [0], [1] and [1], [0]. The Nth forward diagonal has a starting position of [0], [N-1] and in such way the next bit will be [row + 1], [column-1] i.e. [1], [N-2]. This process will be continued until column < 0 or row + 1 > row_size. For backward parity calculation, the 1st diagonal has a starting position of [0], [N-1] i.e. [0], [col_position-1]. Starting bit for Nth diagonal is [0], [0]. The next bit will be [row + 1], [col + 1]. The process will continue until row + 1 > row_size or col + 1 > col_size. In Fig. 2, the forward diagonal parity bits are denoted by D1, D2, D3… D9 and the backward diagonal parity bits are represented by D'1, D'2, D'3… D'9.

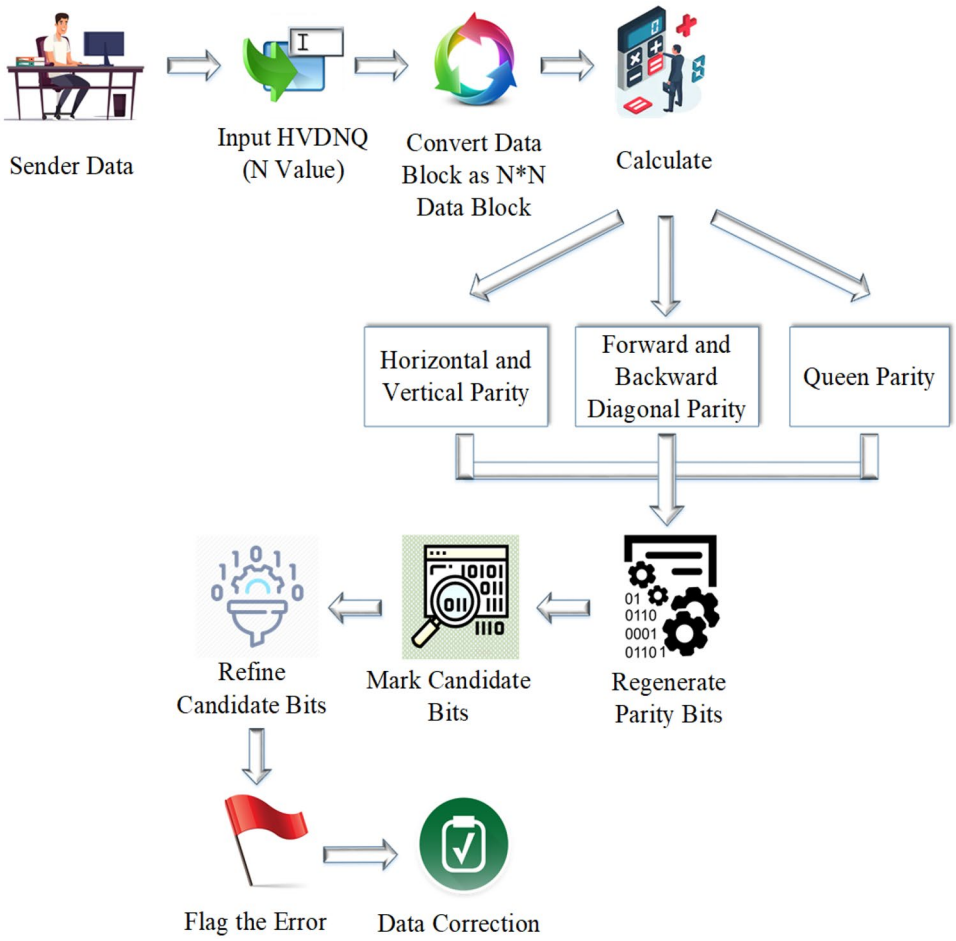


Fig. 1 The flow diagram of the proposed methodology

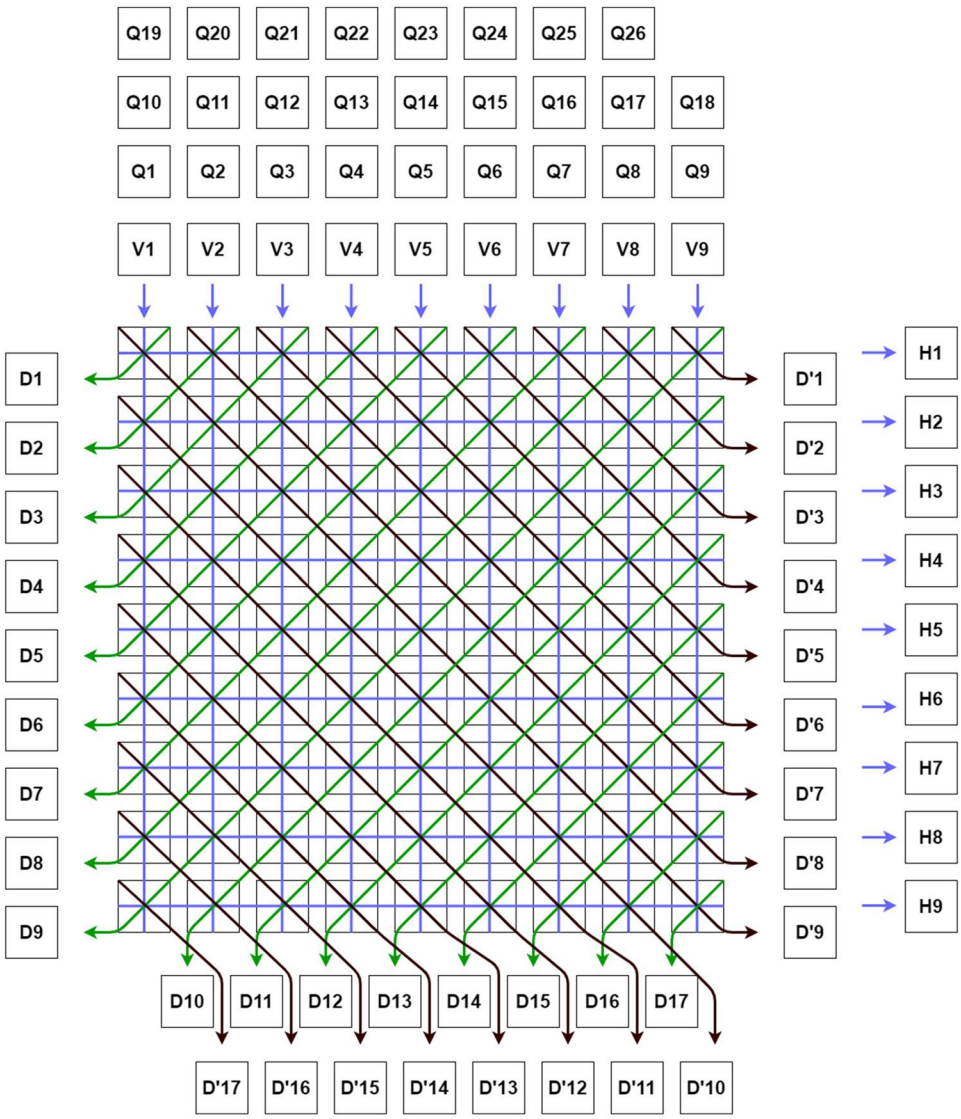


Fig. 2 An architecture of the proposed HVDNQ method (Here, N = 9)

3.2 Queen Parity Calculation

The Queen parity has been used here adapting the N Queen problem. In the classical N Queen problem, no two queens can share the same row, column, or diagonal which gives us a set of unique solutions. These generated unique solutions play an important role in our methodology. The N Queen problem uses a N X N matrix, where N is the number of Queens to be placed in the matrix. The Queen matrix matches with our dataset, which is another reason to use this problem.

During the conventional parity calculation of even/odd Parity, it is often seen that the parity fails to give the actual result in both the sender and receiver. For example, in case of Hamming code, if errors occur in two parity bits of the sender's message, then the system cannot detect error at the receiver end. There are similar types of problems arise in several existing techniques as well.

To increase the error correction coverage, the 5th parity called the Queen parity has been added. When we calculate even parity in vertical and horizontal lines, we see

Step 1: Initialize Row, Column, and Count to 0.
Step 2: Repeat
Step 3: Add value to count
Step 4: Increment row and column
Step 5: If (n modulus 2 ==1)
Set parity bit to 1
Step 6: Else
Set parity bit to 0
Step 7: Until row <=row_size and column <= column_size
Step 7: Endif
Step 8: End.

Fig. 3 Pseudocode for Parity Calculation

that if two bits are changed within the same dimension, the parity is not changed. For illustration, let us take a data word in a row (horizontal dimension) 10,100,110 and the calculated parity, in this case, is 0. Due to changes in two bits, we may get the data word at the receiver side as 10,100,000. Here, we see that with the change of two bits the parity is unaltered. It is possible to flip two or more bits but at the same time parity is not changed; at that point, we cannot identify the erroneous data bits.

For the calculation of queen parity bits, every unique solution for N queen will take place. Unique solutions are required for the calculation of Queen Parity. From the unique Queen solutions, we may find distinct solutions. For example, Let N Queen's problem has M distinct solutions. Among the distinct solutions, we can select P unique solutions which are based on the asymmetric rotation of 90 degrees, 180 degrees, or 270 degrees in their respective matrices. Thus, we can obtain unique solutions. Here, Q (1, 2, 3, 4, 5, 6, 7……0.26) are the 26 unique solutions of the 9 Queen problem. Similarly, if we consider the 7 Queen problem, among the 40 distinct solutions then we shall have 7 unique solutions. For the 9 queen problem, the unique solutions are shown in Fig. 4.

- Q1:(1,2),(2,4),(3,7),(4,3),(5,8),(6,6),(7,1),(8,5),(9,9)
- Q2:(1,2),(2,4),(3,9),(4,7),(5,5),(6,3),(7,1),(8,6),(9,8)
- Q3:(1,2),(2,5),(3,7),(4,4),(5,1),(6,3),(7,9),(8,6),(9,8)
- Q4:(1,3),(2,5),(3,2),(4,9),(5,1),(6,4),(7,7),(8,8),(9,6)
- Q5:(1,4),(2,1),(3,3),(4,6),(5,9),(6,2),(7,8),(8,5),(9,7)
- Q6:(1,5),(2,1),(3,8),(4,6),(5,3),(6,7),(7,2),(8,4),(9,9)
- Q7:(1,5),(2,2),(3,4),(4,9),(5,7),(6,3),(7,1),(8,6),(9,8)
- Q8:(1,6),(2,1),(3,5),(4,2),(5,9),(6,7),(7,4),(8,8),(9,3)
- Q9:(1,6),(2,1),(3,5),(4,7),(5,9),(6,3),(7,8),(8,2),(9,4)
- Q10:(1,6),(2,2),(3,5),(4,7),(5,9),(6,3),(7,8),(8,4),(9,1)
- Q11:(1,6),(2,2),(3,5),(4,7),(5,9),(6,4),(7,8),(8,1),(9,3)
- Q12:(1,7),(2,1),(3,6),(4,2),(5,5),(6,8),(7,4),(8,9),(9,3)
- Q13:(1,7),(2,1),(3,6),(4,9),(5,2),(6,4),(7,8),(8,3),(9,5)
- Q14:(1,8),(2,1),(3,4),(4,6),(5,3),(6,9),(7,7),(8,5),(9,2)
- Q15:(1,8),(2,2),(3,5),(4,7),(5,1),(6,4),(7,6),(8,8),(9,3)
- Q16:(1,1),(2,3),(3,6),(4,8),(5,2),(6,4),(7,9),(8,)7,(9,5)
- Q17:(1,2),(2,4),(3,1),(4,7),(5,9),(6,6),(7,3),(8,5),(9,8)
- Q18:(1,2),(2,4),(3,7),(4,1),(5,3),(6,9),(7,6),(8,8),(9,5)
- Q19:(1,8),(2,2),(3,9),(4,6),(5,3),(6,1),(7,4),(8,7),(9,5)
- Q20:(1,9),(2,2),(3,5),(4,7),(5,1),(6,3),(7,8),(8,6),(9,4)
- Q21:(1,1),(2,6),(3,8),(4,5),(5,2),(6,4),(7,9),(8,7),(9,3)
- Q22:(1,1),(2,7),(3,4),(4,6),(5,9),(6,2),(7,5),(8,3),(9,8)
- Q23:(1,1),(2,8),(3,5),(4,3),(5,6),(6,9),(7,2),(8,4),(9,7)
- Q24:(1,2),(2,9),(3,6),(4,3),(5,5),(6,8),(7,1),(8,4),(9,7)
- Q25:(1,2),(2,3),(3,7),(4,1),(5,8),(6,5),(7,9),(8,4),(9,6)
- Q26:(1,1),(2,5),(3,7),(4,9),(5,4),(6,2),(7,8),(8,6),(9,3)

Fig. 4 The unique solutions of the 9 queen problem

3.3 Finding Mismatched Parity Bits

As data is generated from sender to receiver, the main parity bits of the sender and receiver should be checked. Before sending a data block, the parity is calculated on the sender's side. After receiving the data block from the receiver's side, the same parity calculation is carried out. If there is a dissimilarity between received parity and sent parity, then the mismatched parity bits are detected. Figure 5 shows the pseudocode of detecting mismatched parity bits.

3.4 Candidate Bits' Refining, Error Detection and Correction

Till this point, we have calculated H, V, FD, BD, and queen parity from the given dataword. Then syndrome parity bits are determined by using mismatched parity bits among sent and received parity. Then the candidate bits are calculated. We have chosen the crossing points of three lines to form candidate bits. So, when we take a crossing point of three lines, we mark this bit as candidate bit.

Once the erroneous bits are recognized, these can be corrected by flipping the values of the bits. The detail idea about how the proposed method works can be obtained from the following case study.

4 A Case Study with HVD9Q

To illustrate the proposed methodology, an example of HVD9Q has been shown in this section. HVDNQ is capable to detect and correct multi-bit errors. The number of errors it can correct depends on the number of Queen used in HVDNQ. HVD7Q and HVD8Q can detect and correct up to 5 bits; HVD9Q can detect and correct up to 7 bits of error; HVD10Q, HVD11Q, HVD12Q can detect and correct up to 8 bits of errors; HVD13Q, HVD14Q

Step 1: Initialize a data block
Step 2: Count number of 1 in each row
Step 3: If (Count modulus 2 ==1)
Horizontal parity <- 1
Step 4: Else
Horizontal/Vertical parity <- 0.
Step 5: Repeats step 2, step 3, and step 4 for vertical, and diagonal parity.
Step 6: If (sender parity != receiver parity)
error bit -> 1
Step 7: Endif
Step 8: End

Fig. 5 Pseudocode of detecting mismatched parity bits

can detect and correct up to 9 erroneous bits; while HVD15Q, HVD16Q, HVD17Q can detect and correct up to 10 bits of errors; and HVD18Q, HVD19Q, and HVD20Q can detect and correct 12 and 14 bits of error respectively. In this experiment, we have innjected all possible combinations of faults to evaluate the error correction coverage effectively.

The steps of implementation of HVD9Q has been shown as follows.

4.1 Forming the Architecture of HVD9Q

Fig. 6 illustrates a Codeword Architecture of HVD9Q with 7-bit errors. As shown in this figure, there is an architecture of a 155-bit codeword where it has 81 data bits and 74 check bits. Since in this example (HVD9Q), N = 9, there are 17 diagonals in this architecture. If N is 10, then this value will be 19. In Fig 7, we have highlighted the mistaken parity bits. The parity bits of horizontal, vertical, double diagonal, and Queen parity for HVD9Q have been shown in these architectures. Regeneration of parity bits helps to generate candidate bits. Candidate bits formation procedure is already illustrated in Sect. 3.4. From the candidate bits, we proceed to detect the actual erroneous bits.

4.2 Mark All Candidate Bits

Each parity bit belongs to either one of: : horizontal parity, vertical parity, forward diagonal parity, backward diagonal parity, and queen parity. We have chosen the crossing points of three lines to form candidate bits. In Fig 8, there are 33 parity mismatches. So, when we take a crossing point of three lines, we mark this bit as candidate bit and the refining process is illustrated in Sect. 4.3.

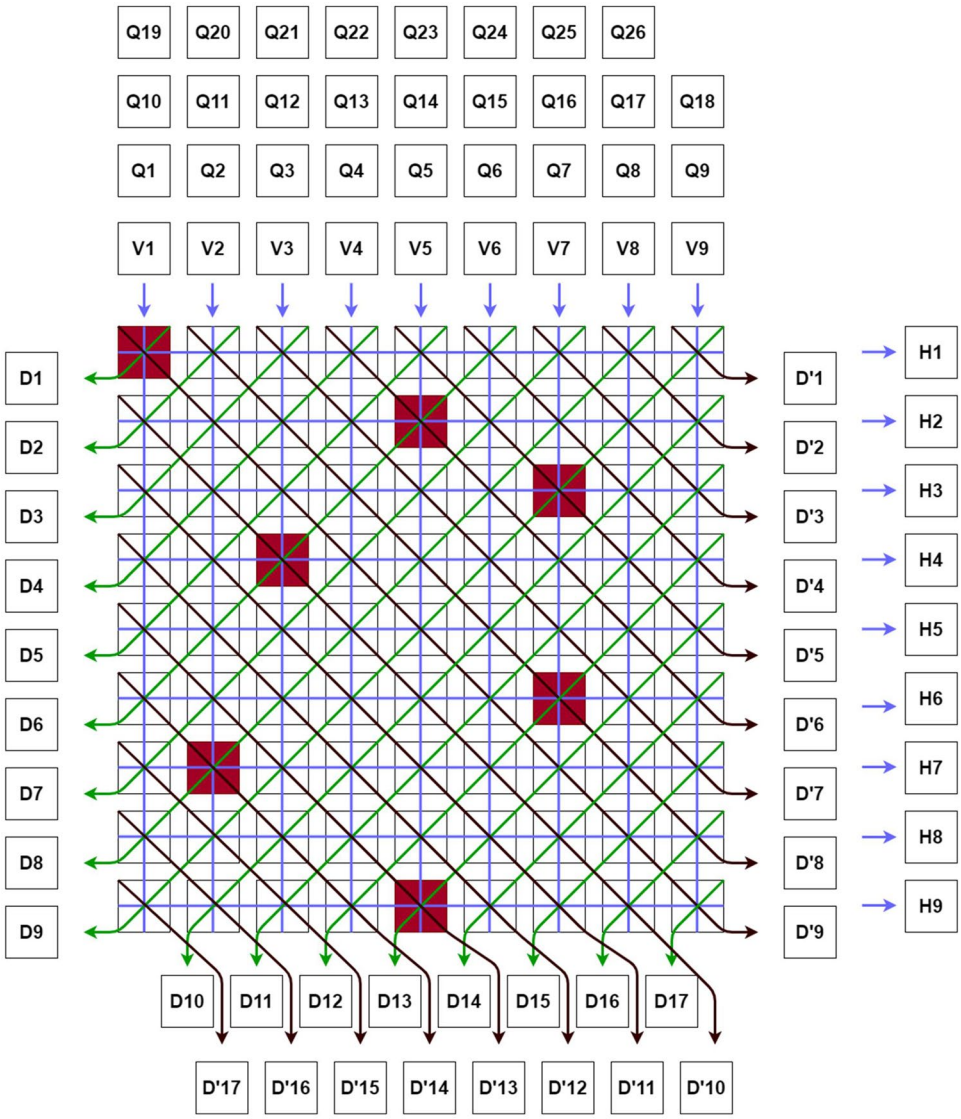


Fig. 6 Codeword Architecture of HVD9Q with 7-bit errors

The addresses of candidate bit are stored in a array. The candidate bits are illustrated in Fig. 8 for the error patterns as shown in Fig. 6.

4.3 Candidate Bits' Refining, Error Detection, and Correction

Candidate bits do not reflect that all of these should be erroneous. For refining candidate bits, the criteria used in this paper is: if in any direction (horizontal, vertical, double diagonal, or Queen), there is any candidate bit with corresponding syndrome bit set to 0, then the candidate bit is discarded. If the corresponding syndrome bit is set to 1, then it is kept. Let us look at some examples of refining candidate bits for 9 Queen parity, on a 9 x 9 data grid. In Figure 9, for the position - (1, 1), both the H1 and V1 are set. In order to refine it, we need to make sure that the forward diagonal, backward diagonal, and the Queen parity bits are set as well. Hence, the parity bits for (1, 1) point are H1, V1, D1, D'9, Q21, and Q26 which are all set. Hence, this is an error.

For the position (1,2): H1 and V2 have a cross-section but the double diagonal parity and the Queen parity are not set. As a result, this is not an error and we have discarded it. Similarly, for (H1, V3), (H1, V4), (H1, V5), (H1, V6), (H1, V7), (H1, V8) on the 1st row, we have eliminated these candidate bits as these are not erroneous. To refine the candidate bits of the second row, horizontal and vertical parity: H2 and V1, H2 and V2, H2, and V3 are set, but the diagonals' and queen parity are not set. Hence, these are discarded from the set of candidate bits. However, in the case of the intersection point - (2, 5): all parity -H2, V5, D6, D'6, and Q26 - are set. Hence, this is an erroneous bit.

Next, we consider another candidate bit on the third row which is the cross-section of H3 and V8. For this position, all other parity bits in D'4, D10, and Q6 parities are set.

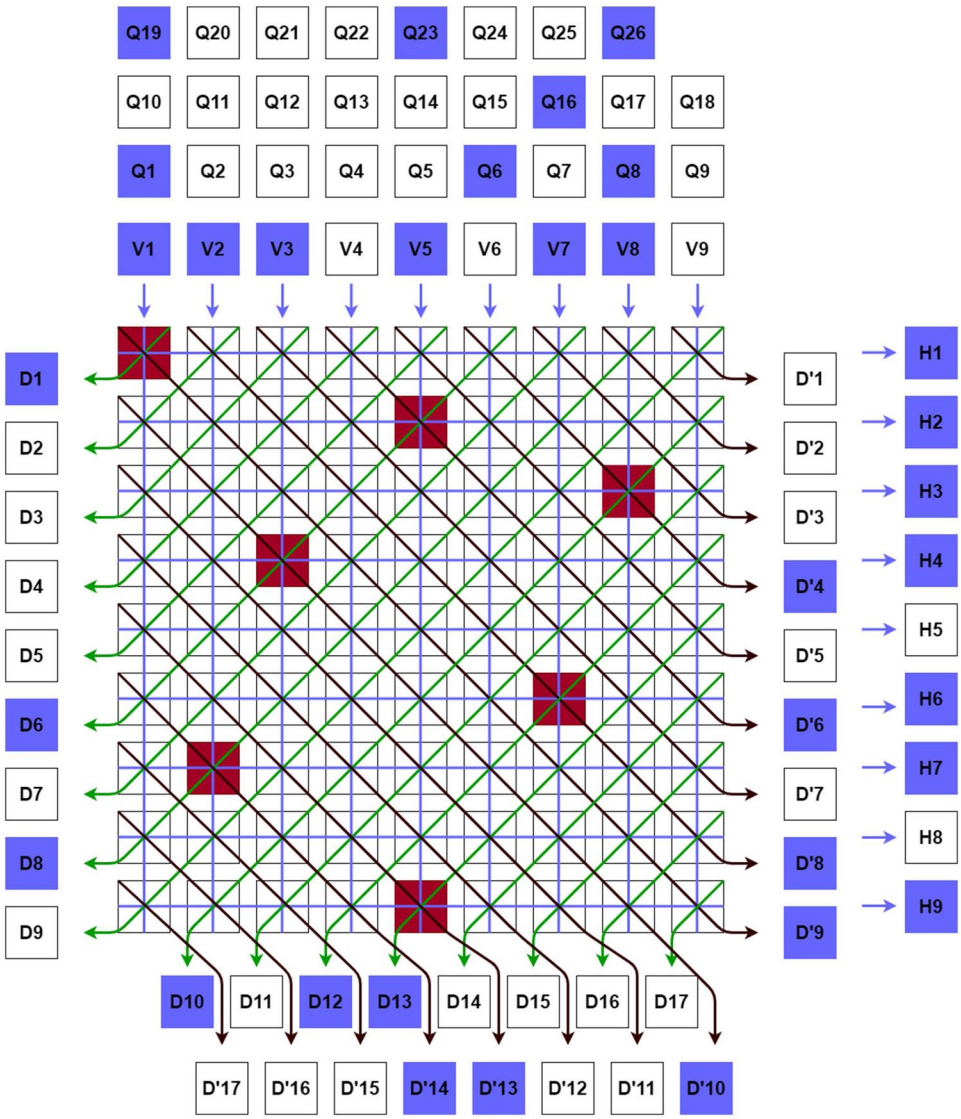


Fig. 7 Codeword with parity bits

As a result, this is an erroneous bit. In such a way, all other cross-sectional bits in other rows are checked whether they have parity set in all five different types of parities. Theoretically, if there is a candidate bit for which no syndrome bits are found in any parity, then the corresponding bit is not erroneous. However, if all five different types of parity bits of a candidate bit are found erroneous, then this bit is erroneous. Thus, the final erroneous bits are detected in (1,1), (2,5), (3,8), (4,3), (6,7), (7,2) and (9,5) positions. Refining the candidate bits are performed at three steps and Fig. 9(a), Fig. 9(b), and Fig. 9(c) respectively show the procedure sequentially. When the erroneous bits are detected, these are corrected by inverting individually.

According to Kishani et al. [21], this rule was proposed for four dimensions. In this paper, we have added one more parity for N queen and it is proved that it improves the error detection and correction capability for five different types of parities. In experimental analysis, we have discussed these results in detail.

Fig. 9 **(a)** Refining Candidate Bits (First Step). **(b)** Refining Candidate bits (2nd Step). **(c)** Refining Candidate Bits (Final Step) ▶

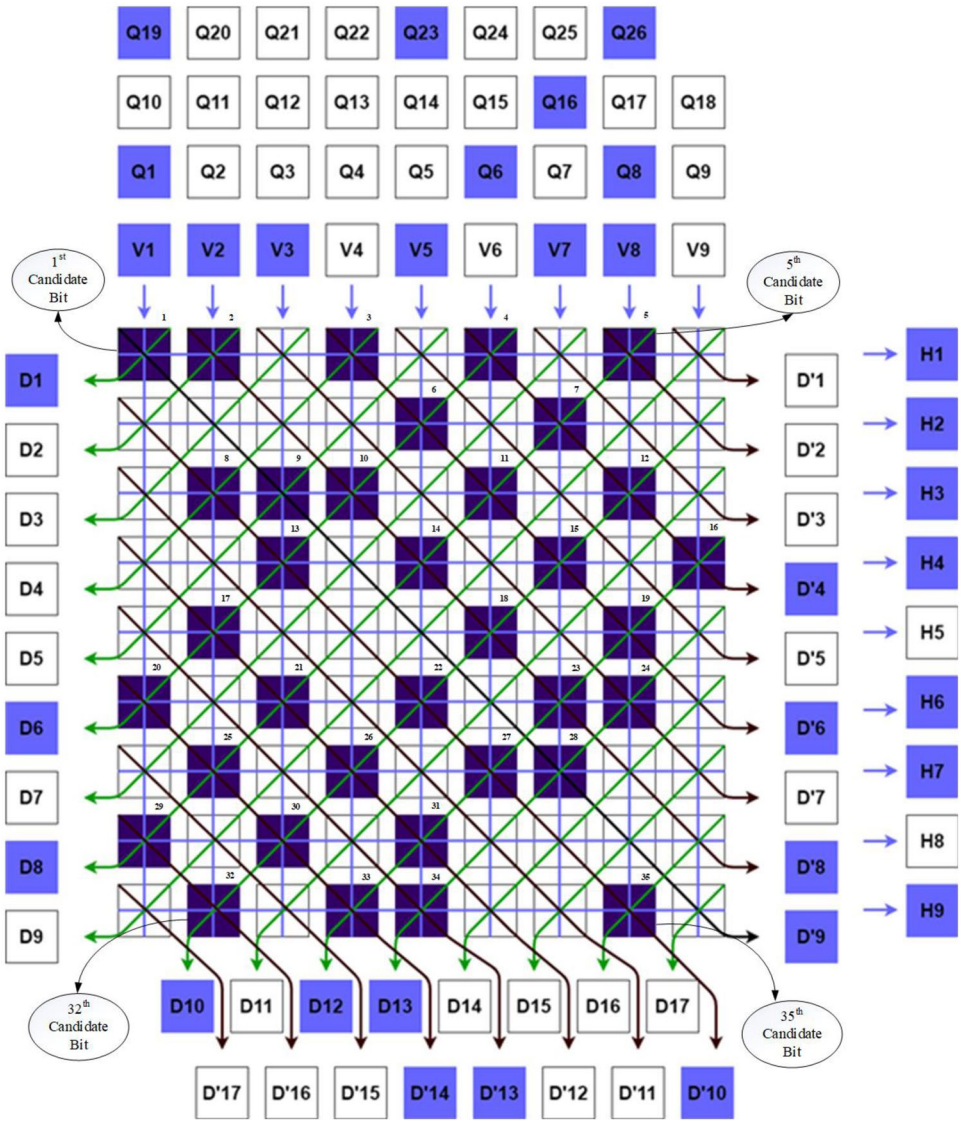
5 Experimental Analysis

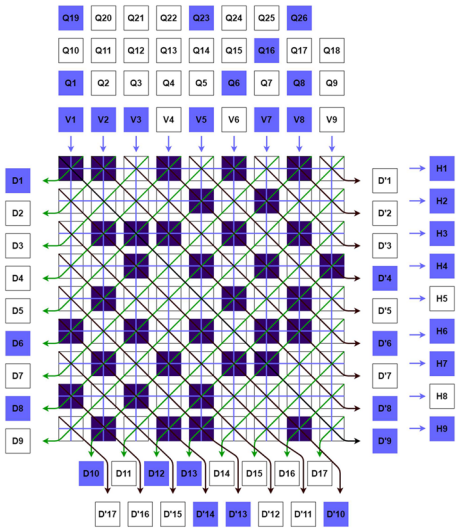
The experimental setup that was used to evaluate the proposed methodology and fault injection are detailed in this section. The experimental results and discussion are also highlighted in this section.

5.1 Experimental Setup

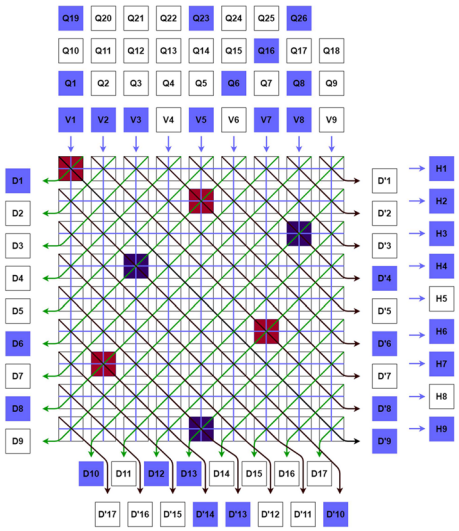
The proposed methodology is executed in Pentium Core i7 processor. The RAM size is 8 Gigabyte, and Turbo C++ language is used for the development of this method.

Fig. 8 Candidate Bits

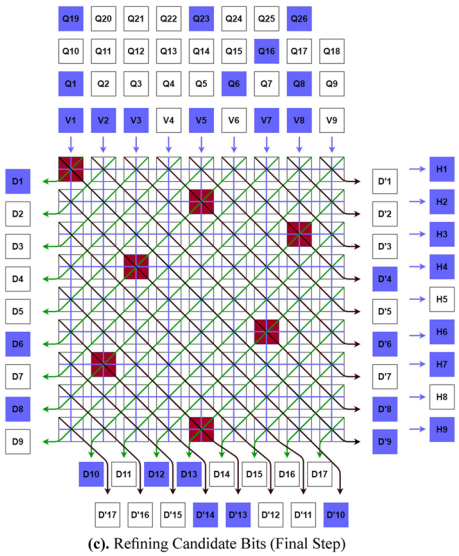




(a). Refining Candidate Bits (First Step)



(b). Refining Candidate bits (2nd Step)



(c). Refining Candidate Bits (Final Step)

5.2 Fault Injection

In this experiment, the error is injected through simulation. Fault injection means bit flipping, which means to change a value from binary '0′ to a binary '1′ and vice versa at any bit's position in the N X N data block. For the N X N data block, to inject one-bit fault, at first we generate all conceivable combinations of one-bit faults and then we took each combination for the injection of faults in the given data block. Similarly, we have generated two, three, four, five, and all combinations' of errors for 7 to 20 Queen solution.

5.3 Error Detction and Correction Capacity

To measure error detection coverage of HVDNQ, we used the N X N data block. Error combinations of the chosen data block are generated using mathematical combinational formula among the data bits. The erroneous bit is detected and corrected as the solution of HVDNQ differs on N's value. In this paper, N's value lies between 7 to 20.

The code rate is an important parameter to evaluate the performance of a fault-tolerant technique. It is the ratio between the data word and codeword. Hence, if n is the size of the dataword and m is the size of the codeword then the code rate of this coding scheme will be n/m. In Fig. 10, the code rate of HVD7Q to HVD12Q is shown to draw the comparison among the varieties of HVDNQ.

Bit overhead, which is the ratio between check bit and data word; Number of erroneous bits that a coding method can correct; and accuracy, which shows the error correction capability of a method, are three vital parameters to flag the performance of an error correction coding method. Table 1 shows the comparison between the proposed HVDNQ and several existing methods concerning these three parameters. For this experiment, we have considered all combinations of errors and maximum eror correction capacity of a method means how many erroneous bits can be corrected at best by that method. It can be observed from this table that HVDNQ performs better than Golay, BCH, HVD, and HVDD concerning bit overhead and the number of the corrected erroneous bits. For HVD7Q, the accuracy is found better than other existing methods but the accuracy moves downward with the increasing value of N in HVDNQ.

Different types of error patterns are shown in Fig. 11. In Table 2, the detailed investigation report, where the capabilities of different methods are shown concerning different error patterns, has been appeared. Here, 'YES' means the method can detect or correct error and 'NO' means the method is not able for so. It is viewed from Table 2 that the proposed HVDNQ performs better than other methods concerning the capability of error detection and correction in different varieties of error patterns.

Table 1 The Comparison between the Proposed HVDNQ and Several Existing Methods

Different Error Correction Methodologies	Size of the data- word	Max Error Correction Capacity (bit)	Number of bits in a code-word	Bit over-head (%)	Accuracy (%)	Memory Requirements (MB)	Execution Time (s)
Golay(23,12,7)	11	3	23	109.1	89.76%	4.0	2891
BCH(31,16,7)	15	3	31	106.6	76.9%	5.0	2446
HVDD(64)	64	3	91	42.19	42.19%	6.7	3540
HVD7Q(49)	49	5	96	95.91	96.2%	7.3	4887
HVD8Q(64)	64	5	124	93.75	70.5%	7.1	4556
HVD9Q(81)	81	7	155	91.35	68.4%	7.4	3776
HVD10Q(100)	100	8	190	90.00	65.2%	6.9	4553
HVD11Q(121)	121	8	227	87.60	58.78%	6.8	4779
HVD12Q(144)	144	8	268	86.11	55.3%	7.5	4998
HVD13Q(169)	169	9	311	84.02	53.1%	7.4	4671
HVD14Q(196)	196	9	358	82.65	53.09%	7.6	4632
HVD15Q(225)	225	10	407	80.88	52.56%	7.4	4678
HVD16Q(256)	256	10	458	78.90	51.67%	7.2	4876
HVD17Q(289)	289	10	511	76.81	50.1%	7.4	4567
HVD18Q(324)	324	12	564	74.07	50.03%	7.3	4356
HVD19Q(361)	361	14	621	72.02	48.9%	7.4	4667

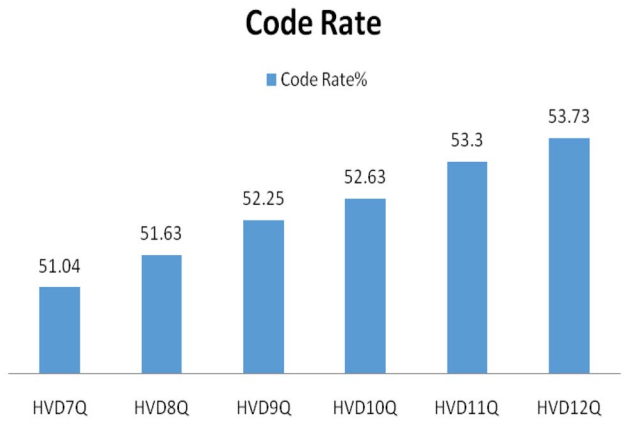


Fig. 10 Comparison of code rates of HVD7Q to HVD12Q

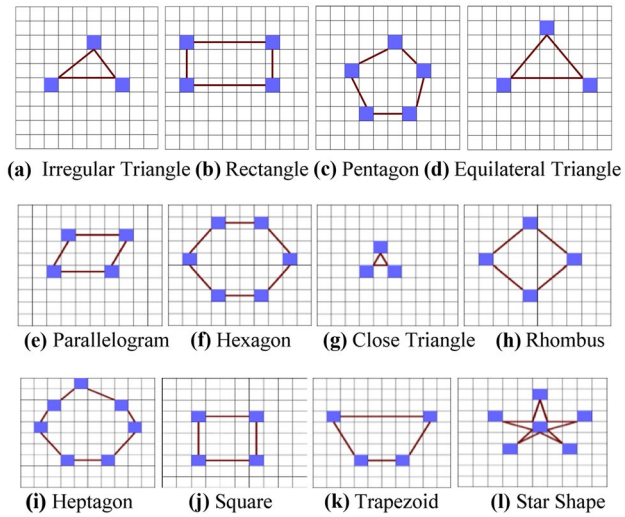


Fig. 11 Different types of error patterns

Table 2 Detecting error patterns by various methods

Error Pattern	Golay	BCH	HVD	HVDD	HVDNQ
Irregular Triangle	Yes	Yes	Yes	Yes	Yes
Equilateral Triangle	Yes	Yes	Yes	No	Yes
Close Triangle	Yes	Yes	Yes	Yes	Yes
Rectangle	No	No	No	No	Yes
Parallelogram	No	No	No	No	Yes
Rhombus	No	No	No	No	Yes
Pentagon	No	No	No	No	Yes
Hexagon	No	No	No	No	Yes
Heptagon	No	No	No	No	Yes
Square	No	No	No	No	Yes
Trapezoid	No	No	No	No	Yes
Star-shaped	No	No	No	No	Yes

6 Conclusion

The proposed HVDNQ method is an efficient error correction coding scheme with a higher error correction coverage. HVDNQ is able to detect and correct 5 to 14-bit errors (the capability varies with the differences of N) where all possible combinations of faults are injected. Subsequently, the proposed HVDNQ delineates the capacity to detect and correct a larger number of errors by encountering information overhead. Hence, it can be concluded that the proposed method is useful for the systems where the systems require high reliability. Further studies can be performed to lower bit overhead by adopting new methodologies, e.g., applying novel compression techniques in the codeword.

References

1. Siddiqui MSM, Ruchi S, Van Le L, Yoo T, Chang IJ, Kim TTH (2020) SRAM Radiation Hardening through Self-Refresh Operation and Error Correction. IEEE Trans Device Mater Reliab. https://doi.org/10.1109/TDMR.2020.2994769
2. Junlong Z, Jin S, Xiumin Z, Tongquan W, Mingsong C, Shiyan H, Xiaobo, Sharon H (2018) Resource management for improving soft-error and lifetime reliability of real-time MPSoCs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38(12):2215–2228. https://doi.org/10.1109/TCAD.2018.2883993
3. Sadi et al. (2018) A New Approach to Tolerate Soft Errors by Using Triple Modular Redundancy. J Inf Technol 7(1):1–7
4. Wu J (2016) Energy efficient dual execution mode scheduling for real-time tasks with shared resources. Comput Syst Sci Eng 31(3):239–253. https://doi.org/10.1016/j.future.2015.05.012
5. Nan H, Choi K (2012) High performance, low cost, and robust soft error tolerant latch designs for nanoscale CMOS technology. IEEE Trans Circuits Syst I Regul Pap 59(7):1445–1457. https://doi.org/10.1109/TCSI.2011.2177135
6. Wang F, Agrawal, VD (2008) Soft error rate determination for nanometer CMOS VLSI logic. In Proceedings 40th Southeastern Symposium on System Theory (SSST), pp. 324-328. https://doi.org/10.1109/SSST.2008.4480247
7. Crouzet Y, Collet J, Arlat J (2005) Mitigating soft errors to prevent a hard threat to dependable computing. In Proceedings 11th IEEE International On-Line Testing Symposium. https://doi.org/10.1109/IOLTS.2005.42
8. Muhammad SS, Palash KB, Palash G, Muhammed SR (2013) A new error correction coding approach. J Adv Inf Technol 4:142–147
9. Hentschke R, Marques F, Lima F, Carro L, Susin A, Reis R (2002) Analyzing area and performance penalty of protecting different digital modules with Hamming code and triple modular redundancy. In Proceedings 15th Symposium on Integrated Circuits and Systems Design. https://doi.org/10.1109/SBCCI.2002.1137643
10. Muhammad SS, Myers DG, Cesar OS (2010) Component Criticality Analysis to Minimizing Soft Errors Risk. In International Journal of Computer Systems Science and Engineering, CRL Publishing, 25:5
11. Ferreyra PA, Marques CA, Ferreyra RT, Gaspar JP (2005) Failure map functions and accelerated mean time to failure tests: New approaches for improving the reliability estimation in systems exposed to single-event upsets. IEEE Trans Nucl Sci. 52(1):494–500. https://doi.org/10.1109/TNS.2005.845883
12. Rubinoff M (1961) n-dimensional codes for detecting and correcting multiple errors0. Commun ACM 4(12):545–551. https://doi.org/10.1145/366853.366878
13. Imran M, Al-Ars Z, Gaydadjiev GN (2009) Improving soft error correction capability of 4-d parity codes. In Proceedings 14th IEEE European Test Symposium
14. Dubney GO (2005) IS Reed Decoding the (23, 12, 7) Golay code using bit-error probability estimates. In Proceedings GLOBECOM. IEEE Global Telecommunications Conference. https://doi.org/10.1109/GLOCOM.2005.1577867
15. Sakib A, Muhammad SS, Md. Shamimur R, Jan J (2017) Soft Error Tolerance using HVDQ (Horizontal-Vertical-Diagonal-Queen parity method). International Journal of Computer Systems Science and Engineering, CRL Publishing, 32(1):35–47
16. Sumaiya M, Dewan, Sadi MS (2019) Soft Error Tolerance using Horizontal, Vertical, Diagonal and Seven Queen Parity. In Proceedings IEEE International Conference on Signal Processing, Information, Communication & Systems (SPICSCON), Dhaka, Bangladesh, pp. 114–117. https://doi.org/10.1109/SPICSCON48833.2019.9064971
17. Pflanz M, Walther K, Galke C, Vierhaus HT (2003) On-line techniques for error detection and correction in processor registers with cross-parity check. J Electron Test 19(5):501–510. https://doi.org/10.1023/A:1025165712071
18. Sharma S, Vijayakumar P (2012) An hvd based error detection and correction of soft errors in semiconductor memories used for space applications. In International Conference on Devices, Circuits and Systems (ICDCS). IEEE. https://doi.org/10.1109/ICDCSyst.2012.6188771
19. Anne NB, Thirunavukkarasu U, Latifi S (2004) Three and four-dimensional parity-check codes for correction and detection of multiple errors. In Proceedings International Conference on Information Technology: Coding and Computing (ITCC). https://doi.org/10.1109/ITCC.2004.1286763
20. Aflakian D, Siddiqui T, Khan NA, Aflakia D (2011) Error detection and correction over two-dimensional and two-diagonal model and five-dimensional model. Int J Adv Comput Sci Appl (IJACSA) 2(7)
21. Kishani M, Zarandi HR, Pedram H, Tajary A (2011) HVD: horizontal-vertical-diagonal error detecting and correcting code to protect against with soft errors. Des Autom Embed Syst 15(3–4):289–310
22. Argyrides C, Pradhan DK, Kocak T (2009) Matrix codes for reliable and cost efficient memory chips.IEEE Trans Very Large Scale Integr (VLSI) Syst 19(3)420–428. https://doi.org/10.1109/TVLSI.2009.2036362
23. Silva F, Freitas W, Silveira J, Marcon C, Vargas F (2020) Extended Matrix Region Selection Code: An ECC for adjacent Multiple Cell Upset in Memory Arrays. Microelectron Reliab 106:113582. https://doi.org/10.1016/j.microrel.2020.113582

Muhammad Sheikh Sadi has been working as a Professor in the department of Computer Science and Engineering, Khulna University of Engineering & Technology, Bangladesh since 2013. He was DAAD Research Scholar in the University of Koblenz-Landau, Germany in 2018. He was also a Visiting Research Scholar in Dependable Embedded Systems and Software (DEEDS) Research Group, TU Darmstadt, Germany in 2008. He has completed his PhD research from the Department of Electrical and Computer Engineering, Curtin University, Australia in 2010. His areas of research interests are: Soft Errors Tolerance, Hardware Redundancy for Fault Tolerance, Error Correction Coding Theory, Humanitarian Technology, and Internet of Things (IOT). He has been holding IEEE membership since 2004. He has been upgraded to senior member of IEEE in 2018. He is reviewers of several reputed journals and he has published more than 70 research papers in peer reviewed journals and conferences.

Sumaiya received B.Sc. (Honors) in Computer Science and Engineering degree from Khulna University of Engineering and Technology, Bangladesh in 2020. She has published two papers in reputed IEEE conferences in her area of expertise. Her reaserch interests include Soft Error Tolerance, Fault Tolerance, and Transmission and Encoding.

Mouly Dewan received B.Sc. (Honors) in Computer Science and Engineering degree from Khulna University of Engineering and Technology in 2020. She has published one paper in a reputed IEEE conference in her area of expertise. Her research interests include Soft Error Tolerance and Fault Tolerant Systems.

Mohammad Atikur Rahman received his BSc in Computer Science and Engineering from Patuakhali Science & Technology University. He is studying as an MS student in the Department of Computer Science

and Engineering at Khulna University of Engineering and Technology. He is an Adjunct Faculty at North Western University in Khulna, Bangladesh and a teaching assistant at Khulna University of Engineering and Technology. Atikur has a strong interest in the field of Human-Computer Interaction and Humanitarian Technology. He has authored several papers in IEEE conferences.