Design a Merging Technique Circuit to Error Detection and Correction Based on Hamming Code and Checksum Using VHDL

Mohammed Sami Mohammed[1], Hanan Badeea Ahmed[2], Yasir Ghazi Rashid[3], and Adham Hadi Saleh[4]

[1]Department of Computer, University of Diyala, 32001 Diyala, Iraq
[2,4]Department of Electronic Engineering, University of Diyala, 32001 Diyala, Iraq
[3]Department of Electrical Power and Machines, University of Diyala, 32001 Diyala, Iraq

***Correspondence:** adham.hadi@yahoo.com; Tel.: (009647713036552)

ABSTRACT- Particularly when considering the Internet, recent developments in communication networks have resulted in considerable rises in both the amount of information transmitted and the user base. Errors frequently occur during transmission and reception of this spike in data transfer, which includes phone, video, and message connections. This work proposes a novel approach to error repair and detection through the combination of two well-known techniques: checksum and Hamming code. This combined strategy minimizes the drawbacks of each technique while leveraging its advantages. Even with single-bit errors, Checksum techniques can cause delays and decreased bandwidth efficiency because they require retransmitting data. Nevertheless, they are effective at detecting the existence of errors. However, when multiple-bit faults arise, hamming codes are ineffective at identifying and fixing single-bit problems. The suggested system offers complete mistake detection and correction capabilities by integrating these two approaches. To be more precise, it guarantees the detection of single- and multiple-bit errors and permits the rectification of straightforward faults without requiring retransmission. This strategy was put into practice by using VHDL on an FPGA platform to develop and simulate transmitter and receiver circuitry in addition to be applied by Python to demonstrate these results. The effectiveness of the combined strategy in improving data integrity and transmission reliability is demonstrated by the successful integration of checksum techniques and Hamming codes within the FPGA architecture.

Keywords: Hamming code, Checksum, VHDL, Error Detection, and Error Correction.

ARTICLE INFORMATION

Author(s): Mohammed Sami Mohammed, Hanan Badeea Ahmed, Yasir Ghazi Rashid, and Adham Hadi Saleh;

Received: 03/10/2024; **Accepted**: 19/12/2024; **Published**: 30/12/2024;
e-ISSN: 2347-470X;
Paper Id: IJEER 0309-02;
Citation: 10.37391/IJEER.120439
Webpage-link:
https://ijeer.forexjournal.co.in/archive/volume-12/ijeer-120439.html

crossref member
CROSSREF.ORG
THE CITATION LINKING BACKBONE

Publisher's Note: FOREX Publication stays neutral with regard to Jurisdictional claims in Published maps and institutional affiliations.

1. INTRODUCTION

In digital communication, hamming code considering as a potent error detection/correction technique while several of parity bits are added to the original data using this coding approach. Because of its ability to identify single-bit errors with correction, it was helpful in field for data integrity importance. The basic idea behind the Hamming code is to add extra bits or sometimes called parity bits. These bits are added to the original data in order to create distinct patterns that help in error detection. In order for this method to function, parity bits must be embedded into a data sequence so that each parity bit will responsible of about certain sets of binaries counted places. The code can identify faults due to this coverage, while the receiver recalculates the parity bits and compares them to the received parity bits to identify any issues or errors. Utilizing a group of these bits with original data in a predetermined structure would offer a methodical way to guarantee data quality and dependability. Hamming code is highly recommended among articles for its ease of use and effectiveness. It is a good option for many applications since it achieves a balance between the error correction level and the quantity of added bits. Some studies emphasize the trade-offs between complexity and efficiency in order to enhance some fields like wireless communication networks in [1]. It determined developments in energy-efficient error control coding when incorporated into network designs. Also, a review of the most recent techniques for wireless communication networks is provided by this article. In this field authors examined ACM, CDM, FEC and error correction coding across a variety of domains, including LPWAN and IoT technologies. Hamming code, which is utilized in memory systems, communication protocols and data storage is essential for preserving the accuracy of digital data since it demonstrates that data sent and received even when errors are present.

In [2], a new technique for using Galois fields (GF) to improve the security of data delivered via noisy communication channels is presented. Numerous fields of contemporary cryptography, error correction and combinatorial design heavily rely on GF computations, sometimes referred to as finite field arithmetic. The aim of [2] was to use these domains to create an encoding and decoding technique that is more effective and safer. The analysis's findings point to two main benefits of the suggested strategy. First, the technique provides a more reliable solution than traditional encoding techniques like the GOLAY code, improving the security of data transmission over noisy channels. Second, it effectively used the frequency bandwidth on Field-Programmable Gate Array (FPGA), which makes it a viable option for hardware implementations in the real world that need strong security and quick processing speeds. The technical history of the Viterbi decoder and its component parts was thoroughly covered in [3]. Three essential components that are essential to the decoder's architecture were used in the design of the suggested Viterbi decoder: The Branch Metric Unit, the Path Metric Unit and the Survivor Memory Unit. I-sim is utilized for the design synthesis, and Xilinx 14.1 is utilized for the simulation of the suggested Viterbi decoder. Both the simulation and synthesis procedures have been effectively completed by the suggested design without any issues. In order to prevent false alarms, an effective background circuit was created in this study to identify objects in the dynamic or static backdrop as explained in [4]. The suggested background circuit, which was based on FPGA, processes the real-time collected image. It had been noted that this proposed work yields good results in terms of area usage and processing time. Two conditions are taken into account for the circuit's operation in [5], when output data for the other ports will be zero when the first port is in a high condition. On the other hand, the outputs of the remaining ports will be proportionate to the outcomes of the calculation of the incoming data when the first port is in a low condition. The processing mechanism used in [5] was a modulo-2 division parallel circuit based on Xilinx ISE Simulator and compared with the Spartan 3E XC3S500E device implementation. Compared to earlier studies that employed the same CRC technique, the suggested circuit design was simple, produces less noise and uses fewer resources. In particular, 223 4-input LUTs, 114 occupied slices, 72 IOB flip-flops, 114 bonded IOBs, and 1 BUFGMUX were used in this investigation. In [Highly Efficient Security Level Implementation in Radiation-Tolerance FPGA Using a Combination of AES Algorithm and Hamming Code: LST-SW Case], a secure land surface temperature (LST) implementation using a radiation-tolerant Virtex-4QV-FPGA that combines the Advanced Encryption Algorithm (AEA) with Hamming code was presented. This paper implemented the Advanced Encryption Standard (AES) algorithm using an iterative looping technique to improve speed and security. In [6], the findings showed that the suggested hardware design achieves a high throughput of 1854.82 Mbps by using 3319 slices and 2 BRAMs. The outcomes of these analyses validated the efficacy of the cryptosystem with respect to its high-security capabilities and hardware efficiency. In order to improve area efficiency and power consumption, [7] presented a unique method for forecasting intermittent failures in Network-on-Chip (NoC) systems. The Presage Debacle model tracked the traffic flow across the network links in real time to predict these sporadic failures within an application-specific architecture. The model contributed to increased system dependability by anticipating possible link failures during runtime. Designing a hardware chip for a turbo encoder and decoder and assessing its performance were the goals of [8]. An interleave, also known as a permute, separates the two convolutional codes that were concatenated in parallel in the turbo encoder. The two linked decoders then iteratively process the channel's received data to increase system performance and decoding accuracy.

One popular error correction technique used in communication systems to fix mistakes is the multiplication error correction scheme as explained in [9]. To fix the flaws in the on-chip connectivity link authors have been suggested a triplication error correction approach. Every encoded message bit is triple-copied using the triplication error correcting process. The ultimate number of bits in the triplication message is therefore 3n. The use of parity-check to find errors is one of the main characteristics of the Hamming code. Binary operations such as XOR, are used to calculate the parity bits' value regarding the sending data which assures that parity-check set has an even total number of 1s. The same parity-check formulae are used to see if errors are founded when a codeword is received in the other side. In the case of founding error, the parity checks determine the particular bit position of the error. Some modifications and studies utilized BPSK modulation in Additive White Gaussian Noise (AWGN) channels to investigate and compare the error correcting capabilities of Hamming code by comparing it with Bose Chaudhuri Hocquenghem (BCH) code cyclic code as explained in [10]. It presented the general and familiar codeword from each decade of the past century in chronological order. Additionally, when the input signal is a Bernoulli binary sequence, it compares the bit error rates of the output signals of the three codes added to the BPSK modulation communication process. Hamming code operates within a mathematical framework known as the Hamming distance, which measures the minimum number of bit changes required to sending data. The minimum distance for Hamming codes is 3, which means the code can detect up to two-bit errors and correct single-bit errors. This property is basic to its effectiveness in error correction with typically implemented in various formats such as (7,4), (15,11) or (31,26). These notations indicate the number of added bits in the codeword and the number of original data bits. Each format provides a balance between the number of parity bits added and the level of error correction. In [11], the authors provided an example of how compact (7, 4) Hamming encoders and implemented using QCAD designer. It is employed to measure the and realize the circuits characteristics, such as latency, cell area, Clock zone and QCA cost. A low-cost QCA-based (7, 4) Hamming encoder and decoder design is proposed in [12]. The Hamming decoder employs a multilayer structure for its error detector while the Hamming encoder is constructed with a coplanar structure. The main objective was to maximize the dissipation of energy, cost and area. The two components of the Hamming decoder implementation are the error detector and the syndrome calculator. The proposed (7, 4) Hamming encoder circuit reduces counting by 49.47% and 9.52% respectively. In addition, it compared to another research with about 56.54% cost reduction and 11.11% area decreasing. The practical applications of Hamming code have numerous fields including computer memory systems, data storage and network communications. In computer memory, hamming code detects and corrects errors that might occur during data storage or even in retrieval process which ensuring the reliability of software and hardware operations. Hamming is useful to manage memory issues due to its simplicity in encoding/decoding with faster process compared to other techniques such as Reed-Solomon as explained in [13]. Authors presented an error model to control memory issues with a developed technique of Hamming code. Similarly, in network communications, it helps to maintain data integrity over unreliable channels and reducing the need for costly retransmission process. Despite its simplicity, its ability to correct errors makes it a valuable tool in various digital systems. DNA storage technique one of the applications that produces a high error rate, which poses a great challenge for the researchers as in [14]. Error detection and correction should be implemented for such an issues to synthesis DNA or providing free-errors. DNA storage especially for big data need these systems as suggested in this article by comparing two algorithms to measure the system effectiveness. To prevent overloading of the transmission channel and to prove the data reliability as well, an accurate selection of code is required as in [15]. The primary benefit of the suggested approach is that it can be used to perform defect repair and information integrity control for a specific security level with the least amount of duplication. Some authors as in [16] suggested a strategy successfully reduces unnecessary bits of the created code, data improving storage and transmission as a whole. There are applications for this study in a number of fields including network protocols, data storage systems and telecommunications. However, hamming code is only able to detect and deal with a single bit of error; it is unable to handle multiple-bit errors without improvement. For that reason, it was applied when there is little chance of multiple errors happening and can provide simple error correcting features while maintaining data integrity.

In the other hand, there is also a basic technique for error detection in data storage which is called as Checksum. It works by adding up the whole transmission data to generate a single number that is called the Checksum value. The Checksum of the original data is calculated by the sender and sent with the data when it is transmitted from a source to a receiver. The receiving side computes the same Checksum on the received data and compares the outcome with the transmission Checksum. It is considering as no error data if the computed checksum matches with the transmitted value. If there are any differences, it leads that there might have been a transmission error [17]. The complexity of a Checksum evaluation can range from basic addition to more complex algorithms. The Checksum can be computed in its simplest form by adding up all bits of data, then dividing the result by a specified amount like 256. Polynomial division is used by more sophisticated Checksum algorithms such Cyclic Redundancy Check to offer a more reliable error detection ability. The Checksum approach is useful for identifying common error types, like single-bit or brief bursts of errors despite its simplicity. It might miss some kinds of corrupted data, especially when there are several mistakes or when they cancel each other out in somehow of calculations. In [18], authors presented a novel method for error identification and correction in Very Large-Scale Integration systems with an ability of multiple error class version of checksum. Nevertheless, there is no mechanism for error correction built into the Checksum method; it is just utilized practically in several previous researches for error detection. It is limited to indicating the happening of an errors occurring and it cannot specify the location of the error. In [19], authors studied on how effective of modulo 11 container number at error detection. Although the modulo 11 checksum calculation approach works well for identifying errors in container numbers, issues can still happen. Error correction is not an intrinsic feature of the container number code; therefore, a new modulo 13 with improved error correction and detection capabilities is suggested by authors. Additional error-correction techniques are frequently applied in order to cover the limits of Checksums, which are incapable of correcting errors. Error-correcting codes like Hamming or Reed-Solomon codes are utilized with Checksums in some cases when multiple detection is required. Reed-Solomon codes are good at managing multiple errors mistakes, while Hamming codes are better at specifying and correcting single-bit positions. This combination can demonstrate both error detection and correction by adding Checksums to one of these error-correcting ability methods and offering reliable method of required application as in [20], [21] and [22]. By combining Checksums and Hamming code, a reliable error detection and correction system is produced which will utilize of the advantages of both methods. By using redundant bits into the data to create a codeword, hamming code is highly effective in identifying and fixing single-bit faults. This makes it possible for error that might happen during transmission or storage to be automatically corrected. In contrast, a Checksum generates a numerical value depending on the contents of the data that providing straightforward approach to error detection. Checksums and Hamming codes combination offer a number of important advantages like addressing several facets of error control. The possibility of error in transmitted data are going to be decreased because of the error correction capacity of the hamming code. In the meanwhile, the Checksum serves as a prior error detection tool by spotting potential errors occurrences before more involved errors repair is required. Reliability is increased by this dual additional techniques, which aids in recognizing both single-bit and more complicated error patterns. Moreover, the performance can be improved by combining the usage of Checksums and Hamming codes. The Checksum offers a rapid and computationally simple way for early error detection, which can quickly spot data errors. Additionally, this combination offers a more comprehensive error management technique that helps to create a more dependable data transmission where many types of mistakes can occur [23] and [24].

2. PROPOSED METHODS FOR ERROR DETECTION COMBINATION

The suggested system combines Checksum and Hamming code techniques to take advantage of each method's advantages. While single-bit errors are corrected using the Hamming code approach, faults are detected using the checksum method. This hybrid technique lowers the need for retransmissions and increases overall system efficiency by ensuring extensive error detection and correction capabilities. Using an FPGA, the error detection and correction system's architecture were constructed using VHSIC Hardware Description Language (VHDL). The great degree of flexibility and real-time processing provided by this implementation platform option make it appropriate for evaluating and verifying the efficacy of the combined strategy.

2.1 Hamming Code Technique

By using some specific arrangements for additional parity bits, hamming code is designed for the purpose of detection and correction errors. For example, when having 4 bits of original transmission data, Hamming will have three additional parity bits to these data bits. This approach is allowing the receiver to both detect and correct errors that occur during transmission, thus enhancing the reliability of data communication [25]. This technique has some steps that should be explained to make the combination process based on this sequence and overlapping with another technique. Steps of Hamming code are listed as follow:

2.1.1. Specifying Parity Bit Numbers

Transmission data is transformed to binary representation with additional parity bits. For example, if data would be 10101100, Hamming has three more additional bits which calculated through *equation 1*. The additional bits are equal to 4 with these positions in sequence (1, 2, 4, 8) based on the power of 2.

$$2^r \geq m + r + 1 \tag{1}$$

Where m is the original bit numbers, r is required number to specify additional parity bits.

2.1.2. Rearrange Data

Based on step 1, the position of these bits are (1, 2, 4, 8) and the new representation of required transmission data would be as follow: $H_1\ H_2\ D_1\ H_3\ D_2\ D_3\ D_4\ H_4\ D_5\ D_6\ D_7\ D_8$. The data will be for the selected bits as $H_1\ H_2$ 1 H_3 010 $H_4$1100, which should be calculated in the next step.

2.1.3. Parity bits' calculations

The calculation of 4 additional parity bits can be done based on each Hamming code basic approach. For example, for the first Hamming code H1, it will be evaluated through the XOR of all position numbers that has 1 in their first bits as given in *equation 2*.

$$H_1 \oplus D_1 \oplus D_2 \oplus D_4 \oplus D_5 \oplus D_7 = 0.$$

Based on equalizing these values with zero and regarding the XOR operation H_1 will be 1 to satisfy the required equation. According to this procedure, the rest Hamming bits (H_2, H_3, H_4) are equal to (000) and the original data will be as follow: - (101001101100). This would be the final transmission data based on Hamming code technique with its additional parity four bits.

2.1.4. Receiver Verification Process

After sending these data to the destination, the XOR operation will be re-count again to demonstrate the error-free data. For the previous example, the recalculation process of *equation 2* which belongs to H_1 would be zero. This value represents the free-error case. However, if there is an error occurred during transmission operation then the results wouldn't be zero. The error position will be calculated according to Hamming checking bits, for example if $H_1H_2H_3H_4$ equal to 1000, that means the bit error is in 8 position[26][27].

2.2 Checksum Technique

Checksum involves the calculations of a specific value from the original data with a determinate procedure. This value will be added to the original data before sending it, while the receiver performs the same calculation and compares the resulting checksum with the transmitted data. With matching data, transmission is considering as free-error and if not, it indicates that errors may have occurred during transmission. This method is in general utilized in various applications, including file transfers and network communications [28][29]. However, it generally does not offer error correction capabilities like Hamming code. checksum steps can be listed briefly as followed:

2.2.1. Transforming Data

Data will be read as a decimal number, for example having this value 11001010 will be 202 in Decimal.

2.2.2. Checksum Generation

Checksum generated using the highest possible value based on the power of 2 that should have covered the required sending data. In this example, it would be 255-202=53 and it represented the Checksum value.

2.2.3. Receiver Verification Process

After sending these data to the destination, the summation between received data and available Checksum value in the receiving side would be zero. This value represents the free-error receiving, while any different result would be an error occurrence and need another technique for error position specifications and correction. The Checksum technique involves calculating a value based on the original data by sending this value along with the data. Then recalculating the checksum upon receipt to check for errors. This approach ensures that data can be quickly checked and making it an effective tool for error detection in several system fields like Hamming code. This similarity in usage and simplicity process between these techniques would be helpful to combine them together [30][31].

2.3 Hamming Code Plus Checksum Combination Technique

By utilizing the advantages of both techniques, the combination of Checksum and Hamming code improves data integrity in digital communication. By introducing redundancy in the form of parity bits, the Hamming code offers error correction capabilities that enable the identification of single-bit errors in the sent data. A checksum, on the other hand, provides an extra degree of error detection by calculating a value based on the complete data block. This would help in locating any errors that might happen during transmission. When combined, these features demonstrate the identification of smaller mistakes as well as the specifications of larger issues to enhance the general dependability of data transmission. This hybrid strategy is especially useful in settings where data accuracy is essential and need more error detection ability at the same time. The steps of combination process would be as followed:

The steps of the proposed algorithm (Hamming + Checksum)

1. Sender Side Flow Chart
   - A. *Start*
   - B. *Divide Input Data (32 bits into 4 groups)*
     - ➢ *Input datain1 (8 bits)*
     - ➢ *Input datain2 (8 bits)*
     - ➢ *Input datain3 (8 bits)*
     - ➢ *Input datain4 (8 bits)*
   - C. *Process Data (Applying Checksum)*
     - ➢ *Apply Checksum to obtain Dataout1, Dataout2, Dataout3, and Dataout4 (each 12 bits)*
     - ➢ *Zer010 and Zero11 are initialized to zero*
   - D. *Process data (Applying Hamming Code)*
     - ➢ *Apply Hamming code to form Dataout5 (17 bits)*
   - E. *Output Data*
     - ➢ *Output Dataout5 (17 bits)*
   - F. *End*

The proposed model steps and sender side diagram are shown in *figure 1*.

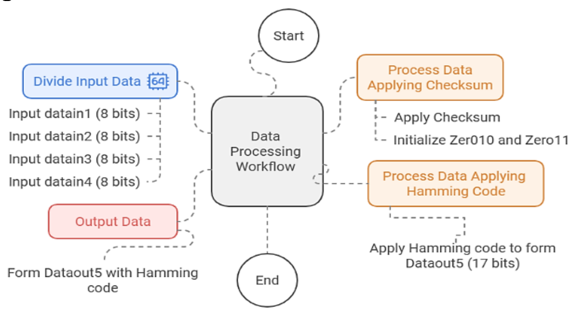

Start
Divide Input Data
Input datain1 (8 bits)
Input datain2 (8 bits)
Input datain3 (8 bits)
Input datain4 (8 bits)
Data Processing Workflow
Process Data Applying Checksum
Apply Checksum
Initialize Zer010 and Zero11
Process Data Applying Hamming Code
Apply Hamming code to form Dataout5 (17 bits)
Output Data
Form Dataout5 with Hamming code
End

Figure 1. The Sender Side Diagram of related 32 bits based on proposed algorithm

2. Receiver Side Flow Chart
   - A. *Start*
   - B. *Receive Data*
     - ➢ *Receive DATAIN5 (17 bits)*
   - C. *Process Data*
     - ➢ *Apply Hamming code to form Dout1, Dout2, Dout3, Dout4 (each 8 bits)*
   - D. *Check Retransmission Signal*
     - ➢ *Check RETRANS*
       - ▪ *If RETRANS is zero, data is correct*
       - ▪ *If RETRANS is non-zero, request retransmission*
   - E. *Output Data*
     - ➢ *Output Dout1 (8 bits)*
     - ➢ *Output Dout2 (8 bits)*
     - ➢ *Output Dout3 (8 bits)*
     - ➢ *Output Dout4 (8 bits)*
   - F. *End*

The proposed model steps and receiver side diagram are shown in *figure 2*.

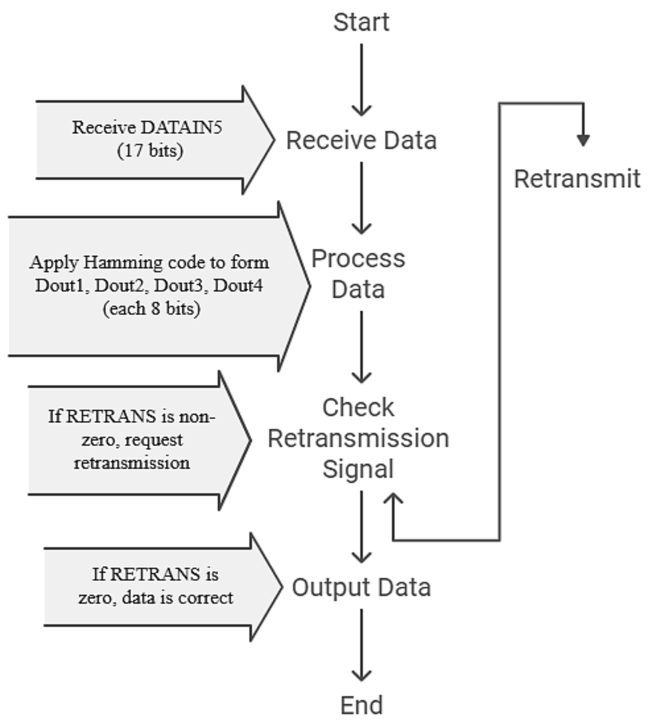

Start
Receive DATAIN5 (17 bits)
Receive Data
Retransmit
Apply Hamming code to form Dout1, Dout2, Dout3, Dout4 (each 8 bits)
Process Data
If RETRANS is non-zero, request retransmission
Check Retransmission Signal
If RETRANS is zero, data is correct
Output Data
End

Figure 2. The Receiver Side Diagram of related 32 bits based on the receiver side

In this proposal work we divided the 32 bits into 4 bytes (four groups of 8 -bits) this proposal method is getting high performance as compared with classic hamming coding method with 32- bits, since the classic method will be detected only one error bit and corrected it. Where the proposed method will have corrected single error bit at 4 bytes which mean corrected 4 bits at the overall 32- bits. As shown in *figure 3*.

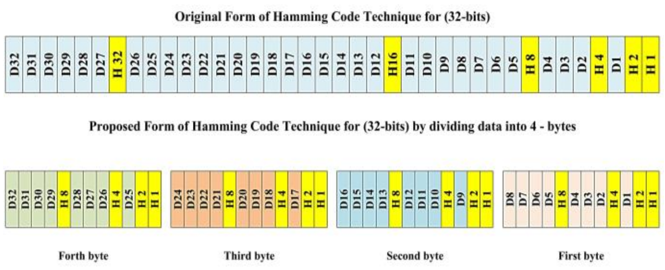

Original Form of Hamming Code Technique for (32-bits)
D32 D31 D30 D29 D28 D27 H 32 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16 D15 D14 D13 D12 H16 D11 D10 D9 D8 D7 D6 D5 H 8 D4 D3 D2 H 4 D1 H 2 H 1
Proposed Form of Hamming Code Technique for (32-bits) by dividing data into 4 - bytes
D32 D31 D30 D29 H 8 D28 D27 D26 H 4 D25 H 2 H 1
D24 D23 D22 D21 H 8 D20 D19 D18 H 4 D17 H 2 H 1
D16 D15 D14 D13 H 8 D12 D11 D10 H 4 D9 H 2 H 1
D8 D7 D6 D5 H 8 D4 D3 D2 H 4 D1 H 2 H 1
Forth byte
Third byte
Second byte
First byte

Figure 3. Hamming proposal technique

In the other hand if there are many errors getting at one byte the error corrected by using check sum so the data will be corrected by retransmitted. The proposal error detection and correction is shown in *figure 4*.

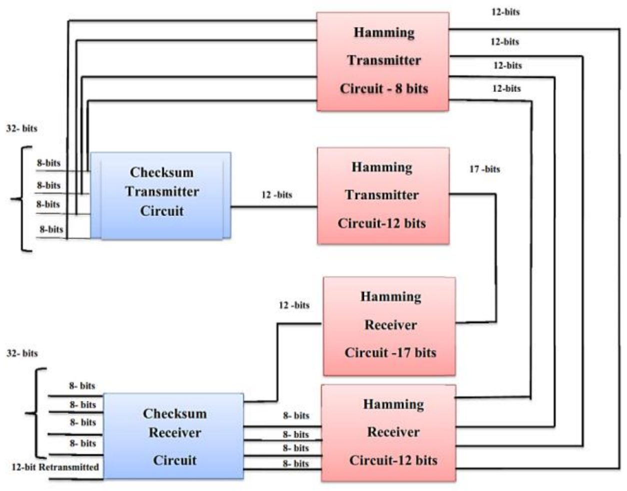

Hamming Transmitter Circuit - 8 bits
12-bits
12-bits
12-bits
12-bits
32- bits
8-bits
8-bits
8-bits
8-bits
Checksum Transmitter Circuit
12 -bits
Hamming Transmitter Circuit-12 bits
17 -bits
Hamming Receiver Circuit -17 bits
12 -bits
32- bits
8- bits
8- bits
8- bits
8- bits
12-bit Retransmitted
Checksum Receiver Circuit
8- bits
8- bits
8- bits
8- bits
Hamming Receiver Circuit-12 bits

Figure 4. The proposed system design

These steps were applied for an example which is shown in *figure 3*, six steps were applied and obtaining the same results at sending and receiving side as well.

3. HARDWARE DESIGN AND SIMULATION RESULTS

The privilege of the combined Hamming code and Checksum approach was proved by the VHDL construction of the transmitter and receiver circuits. Simple errors did not need retransmission because the system correctly identified and fixed errors, including single-bit and multiple-bit errors. The transmitter circuit bock diagram is shown in *figure 5*. Where the pin description of the transmitter circuit is explaining in *table 1*.

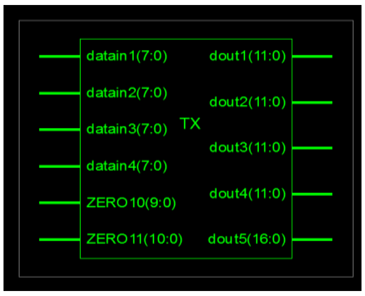


Figure 5. The proposed transmitted circuit

Table 1: IP-v4 selection for the proposed method

Input Pins	Description	Output Pins	Description
Datain1	Input data /one byte (8- bits)	Dataout1	Output transmitted data (12- bits)
Datain2	Input data / one byte (8- bits)	Dataout2	Output transmitted data (12- bits)
Datain3	Input data /one byte (8- bits)	Dataout3	Output transmitted data (12- bits)
Datain4	Input data / one byte (8- bits)	Dataout4	Output transmitted data (12- bits)
Zer010	Rest pins for initial value which always equal to zero	Dataout5	Output transmitted data (17- bits)
Zero11	Rest pins for initial value Which always equal to zero	0	0

With multiple examples as illustrated in *figure 6* and *figure 7*, this system has undergone validation and testing to guarantee the effectiveness and resilience of the applied system. The desired message is represented by the initial 1A 1A 1A 1A 1A input data to the transmitter circuit. Nevertheless, the transmitter outputs 92A 92A 92A 92A B38 following processing (perhaps involving error detection and repair procedures). The last B38 indicates that extra information, like a parity bit or checksum, was added to the message for error-checking. The receiver can confirm the accuracy of the data they have received thanks to this redundancy. The identical data (92A 92A 92A 92A B38) is subsequently sent to the recipient, who processes it to recover the original message. The receiver confirms that the data was successfully recovered by producing 1A 1A 1A 1A after decoding, which corresponds to the original input data from the transmitter. This proves the system's ability to identify and fix any possible transmission faults, guaranteeing an error-free final output. Error correction techniques and error detection codes, such as B38, guarantee dependable communication even when there is interference or transmission noise.

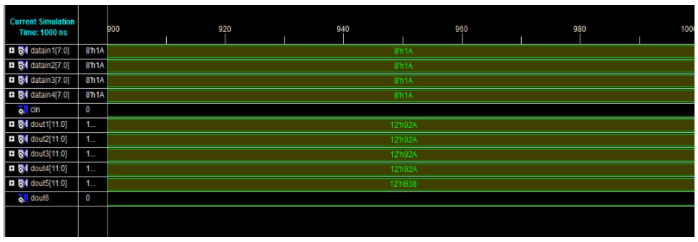
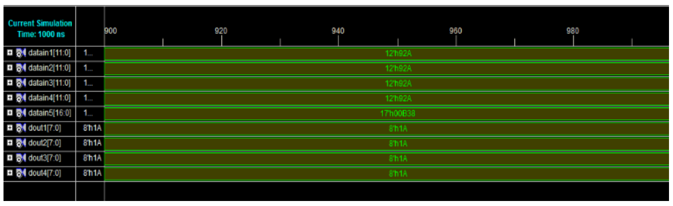


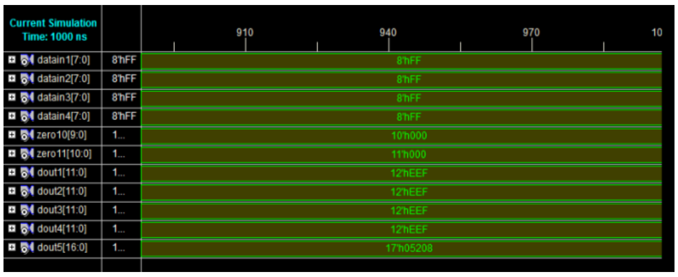
Figure 6. The proposed simulation results based on the proposed method and for input data 1A 1A 1A 1A at the transmitter side



```
Current Simulation
Time: 1000 ns        900    920    940    960    980
datain1[11:0]   1...       12'h92A
datain2[11:0]   1...       12'h92A
datain3[11:0]   1...       12'h92A
datain4[11:0]   1...       12'h92A
datain5[16:0]   1...       17'h00B38
dout1[7:0]      8'h1A      8'h1A
dout2[7:0]      8'h1A      8'h1A
dout3[7:0]      8'h1A      8'h1A
dout4[7:0]      8'h1A      8'h1A
```

Figure 7. The proposed simulation results based on the proposed method and for input data 1A 1A 1A 1A at the receiver side

The transmitter Time simulation is shown using ISE and Python in *figure 8 (a)* and *(b)*, respectively.



```
Current Simulation
Time: 1000 ns        910    940    970    10
datain1[7:0]    8'hFF      8'hFF
datain2[7:0]    8'hFF      8'hFF
datain3[7:0]    8'hFF      8'hFF
datain4[7:0]    8'hFF      8'hFF
zero10[9:0]     1...       10'h000
zero11[10:0]    1...       11'h000
dout1[11:0]     1...       12'hEEF
dout2[11:0]     1...       12'hEEF
dout3[11:0]     1...       12'hEEF
dout4[11:0]     1...       12'hEEF
dout5[16:0]     1...       17'h05208
```

Figure 8. (a) Time simulation of the transmitter circuit /data ISE program

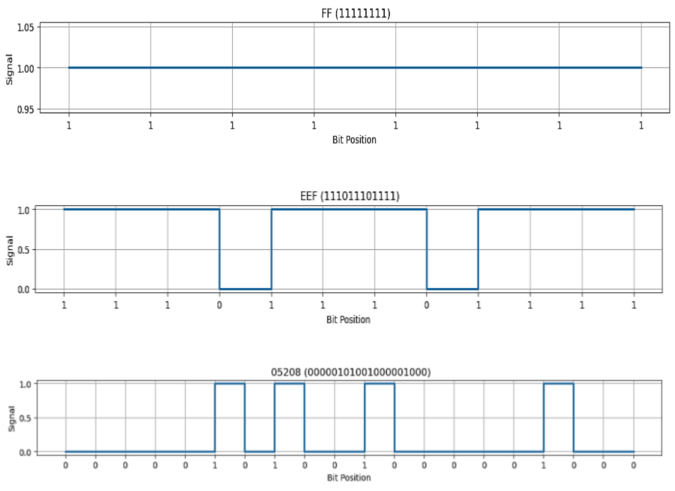


Figure 8. (b) Time simulation of the transmitter circuit data /python program

A summary of the use of slice logic in an FPGA design is given in the accompanying *table 2*. With an emphasis on slice registers, LUTs, route-thrus, occupied slices and IOBs, the metrics in the table show how the FPGA uses its resources. The design uses one, for a 1% usage rate of the 19,200 possible slice registers. This suggests that the design uses the slice registers that has a limited effect on this specific resource. With 3 route-thrus used out of 38,400 available, the design utilizes 1% of the available space. The limited employment of route-thrus, which link various logic units within the FPGA, suggests that the design has less internal routing complexity. This low utilization rate indicates a well-thought-out design that maximizes the use of slice resources. Furthermore, the design contains unused LUTs equal to zero, which implies that even when slices are used, a large percentage of them are still unused, especially when it comes to flip-flops. In addition, out of the 78 available pairs, there is just one fully utilized LUT-FF pair representing a 1% usage rate for these particular pairs. The limited number of completely utilized LUT-FF pairings highlights the effective resource utilization of the architecture even more. Also, out of the 220 available, the design uses 118 bonded IOBs, or a 53% usage rate. This shows a moderate to high IOB resource utilization, indicating that the design is utilizing a sizable amount of the FPGA's I/O capabilities. The table shows a design that uses a modest amount of IOBs and very little slice registers, LUTs, route-thrus and occupied slices.

Table 2. Design summery of transmitter proposed system

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	1	19,200	1%
Number of Slice LUTs	78	19,200	1%
Number used as logic	78	19,200	1%
Number of route-thrus	3	38,400	1%
Number of occupied Slices	43	4,800	1%
Number with an unused Flip Flop	77	78	98%
Number with an unused LUT	0	78	0%
Number of fully used LUT-FF pairs	1	78	1%
Number of bonded IOBs	118	220	53%

The input and output pin description of the receiver circuit show in *table 3*, and the design summary of the receiver circuit is show in *table 4*. *Figure 9* shows the used percentage of logical devices based on the total available ratio. The results show how an FPGA design uses logic resources. With only one slice register used in this instance out of 19,200 possible slice registers, the design had made little use of register resources. All 78 of the slice LUTs that were being used are also being used for logic. This indicates that just a small portion of the 19,200 slice LUTs available were been used for the current architecture. Furthermore, out of the 38,400 available route-thru elements, the design used only three of them to route signals between logic blocks, indicating that routing resources were not widely utilized. Out of the 4,800 slices that are accessible, 43 slices had been taken by the design. This shows that just a small portion of the FPGA's slice resources have been utilized by the current setup, with the majority still being available. All things considered, these findings imply that the architecture uses resources quite effectively, leaving plenty of room for more reasoning, routing or optimization. The design is not pushing the boundaries of the FPGA's capabilities, as evidenced by the limited consumption of slice registers, LUTs, and routing resources.

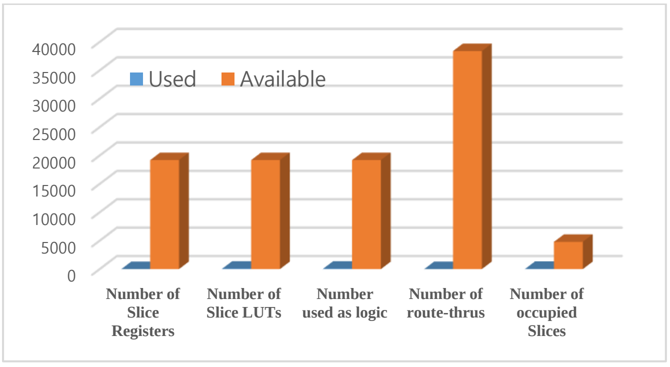


Figure 9. Logical devices ratio based on the total available number of Slice in the sending side

While *figure 10* shows the Logical devices ratio based on the total available Flip Flop and LUTs. An in-depth examination of the use of flip-flops, LUTs and IOBs in the FPGA architecture was provided by the results in *table 2* and shown by this figure. As shown in *figure 10*, 77 of the 78 accessible LUTs were not fully utilized, meaning that most LUTs are either not used at all or are only used partially. This implies that the LUT resources have a sizable amount of unused capacity that might been used for more logic if necessary. The fact that there was one fully utilized LUT is noteworthy since it indicates that at least one LUT was being properly optimized for its intended purpose. Since all 78 of the flip-flops in the design are being used, the data indicate that none were left unused. This suggests that flip-flops have been used effectively, with each one actively contributing to the design, perhaps for data synchronization or storage. Although the results did not explicitly state how many LUT-FF pairs were utilized, the use of both LUTs and flip-flops indicates that the design was balancing the use of logic and storage components. Finally, 118 of the 220 bonded IOBs on the FPGA were in use, which indicates that roughly 53% of the I/O pins had utilized. A sizable percentage of IOBs are thus still open for growing external relationships.

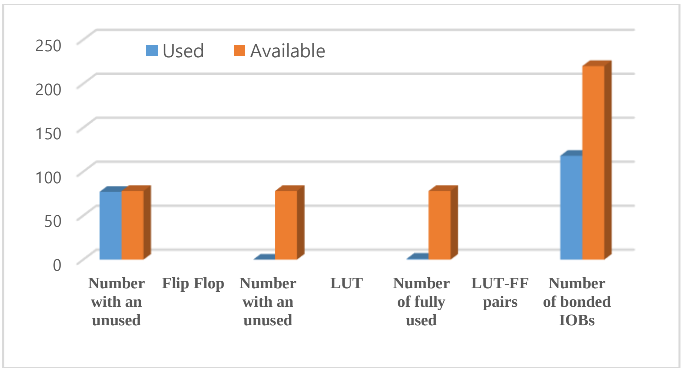


Figure 10. Logical devices ratio based on the total available Flip Flop, LUTs and IOBs/ sending side

Table 3. The receiver pins description

Input Pins	Description	Output Pins	Description
DATAIN1	Input data /(12-bits)	Dout1	Output received / (7- bits)
DATAIN2	Input data / (12-bits)	Dout2	Output received / (7- bits)
DATAIN3	Input data / (12-bits)	Dout3	Output received / (7- bits)
DATAIN4	Input data / (12-bits)	Dout4	Output received / (7- bits)
DATAIN5	Input data / (17-bits)	RETRANS	Output signal if it has zero value the received data is correct else data need to be retransmitted
ZERO10	Rest pins for initial value which always equal to zero	/	/
ZERO11	Rest pins for initial value which always equal to zero	/	/

Table 4. Design summary of the receiver circuit 3

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	44	19200	0%
Number of Slice LUTs	143	19200	0%
Number of fully used Bit Slices	44	143	30%
Number of bonded IOBs	131	220	59%

Several resource metrics were used to assess the FPGA implementation's logic usage for the suggested security technique as explained more for the receiving side in Figure 11. In particular, 44 of the 19,200 Slice Registers that are available on the FPGA are utilized. This suggests that the available slice registers—the FPGA's fundamental storage components—are being used sparingly. With regard to Slice LUTs (Look-Up Tables), 143 of the 19,200 possible LUTs are used, indicating a comparatively low logic gate resource use. Additionally, 44 of the 143 available Bit Slices are fully utilized, indicating that the FPGA's bit-level resources are likewise effectively used. Lastly, the method utilized input/output resources to connect the FPGA to external components or interfaces, using 131 of the 220 available bonded I/O blocks (IOBs). The suggested approach was effective in terms of FPGA resource usage, as evidenced by the generally low resource utilization, which left plenty of room for future scalability or extra features.

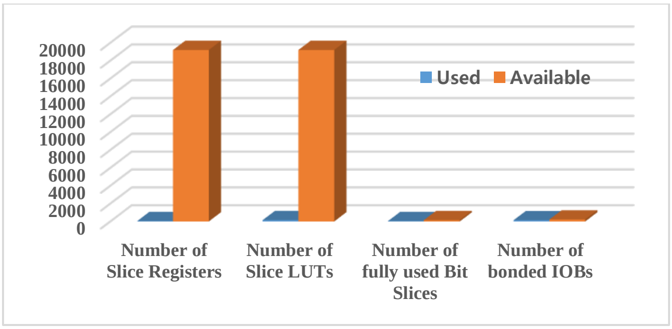
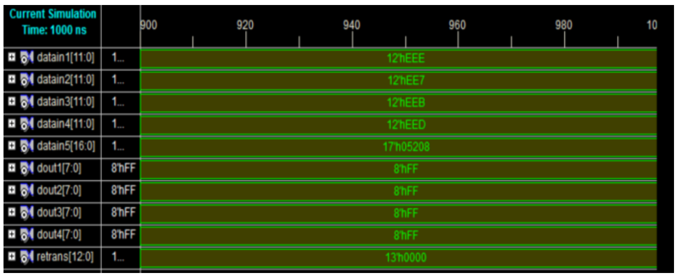


Figure 11. Logical devices ratio based on the total available Flip Flop, LUTs and IOBs/ Receiving side

Table 5. The Results of proposed system

Transmitter Circuit		Receiver Circuit				
Divide data into 4 Groups	Transmitting Output Data	Receiver Input Data	Receiver Output Data	Single Error Detection And Correction	Burst Error Detection And Correction	Output Data Status
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
	05208	05208	000			✓
FF	EEF	EEE	FF	✓		
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
	05208	05208	000			✓
FF	EEF	EEE	FF	✓		
FF	EEF	EE7	FF	✓		
FF	EEF	EEF	FF			
FF	EEF	EEF	FF			
	05208	05208	000			✓
FF	EEF	EEE	FF	✓		
FF	EEF	EE7	FF	✓		
FF	EEF	EEB	FF	✓		
FF	EEF	EEF	FF			
	05208	05208	000			✓
FF	EEF	EEE	FF	✓		
FF	EEF	EE7	FF	✓		
FF	EEF	EEB	FF	✓		
FF	EEF	EED	FF	✓		
	05208	05208	000			✓
FF	EEF	AB1	00			
FF	EEF	EE7	FF			
FF	EEF	CDE	6E			
FF	EEF	AFD	00			
	05208	05208	28F		✓	✓



```
Current Simulation
Time: 1000 ns         900    920    940    960    980    10
datain1[11:0]   1...  12'hEEE
datain2[11:0]   1...  12'hEE7
datain3[11:0]   1...  12'hEEB
datain4[11:0]   1...  12'hEED
datain5[16:0]   1...  17'h05208
dout1[7:0]     8'hFF  8'hFF
dout2[7:0]     8'hFF  8'hFF
dout3[7:0]     8'hFF  8'hFF
dout4[7:0]     8'hFF  8'hFF
retrans[12:0]   1...  13'h0000
```

Figure 12. Time simulation of the receiver circuit data based on ISE program for data example 1

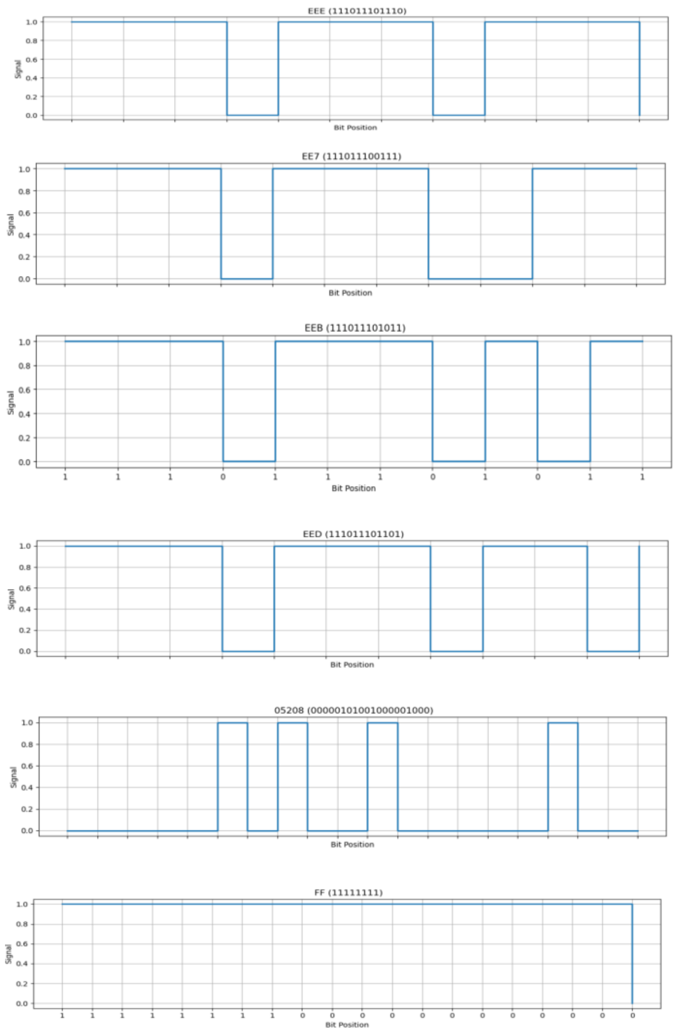
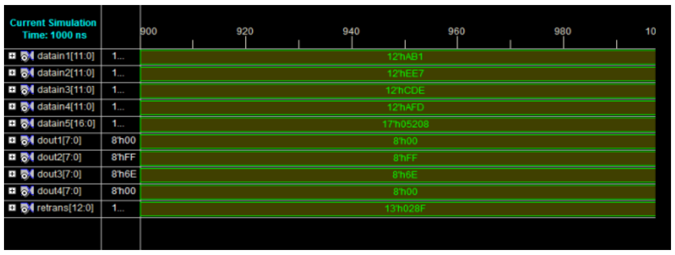


Figure 13. Time simulation of the receiver circuit data based on Python program for data example 1



```
Current Simulation
Time: 1000 ns         900    920    940    960    980    10
datain1[11:0]   1...  12'hAB1
datain2[11:0]   1...  12'hEE7
datain3[11:0]   1...  12'hCDE
datain4[11:0]   1...  12'hAFD
datain5[16:0]   1...  17'h05208
dout1[7:0]     8'h00  8'h00
dout2[7:0]     8'hFF  8'hFF
dout3[7:0]     8'h6E  8'h6E
dout4[7:0]     8'h00  8'h00
retrans[12:0]   1...  13'h028F
```

Figure 14. Time simulation of the receiver circuit with burst error at different byte based on ISE Program for data example 2

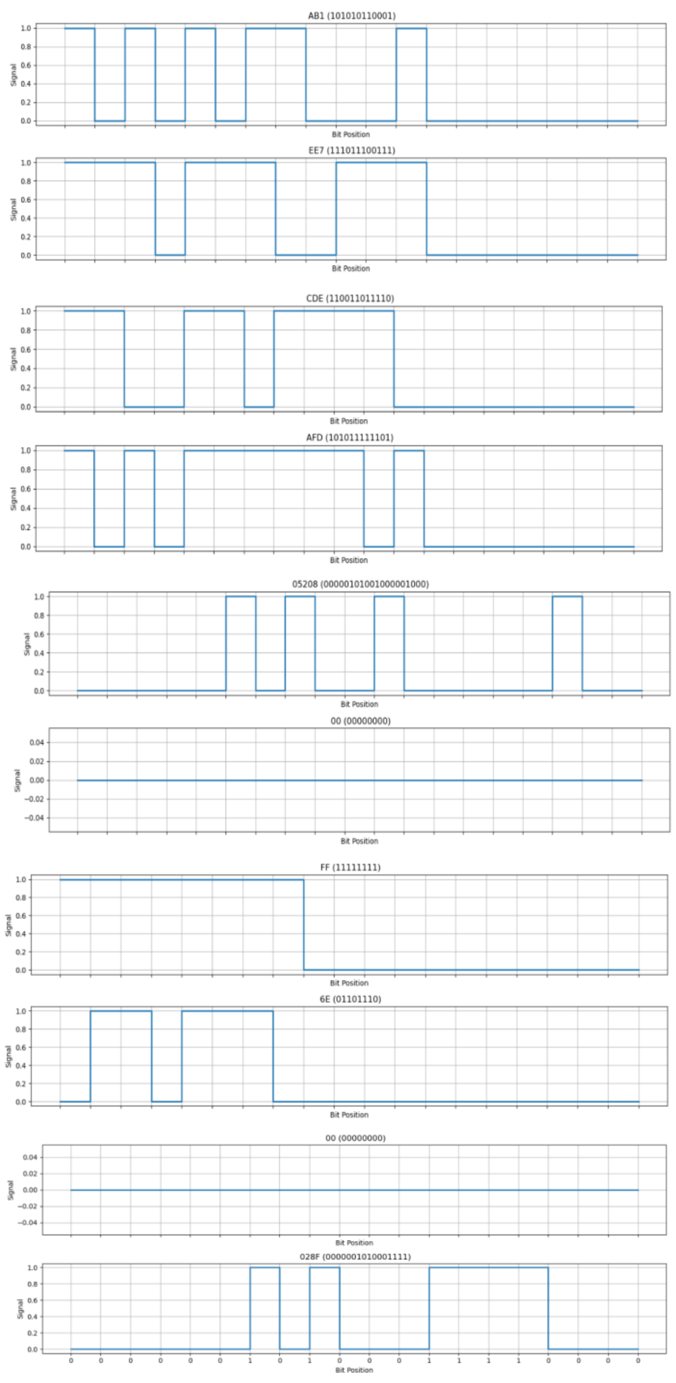


Figure 15. Time simulation of the receiver circuit with burst error at different byte based on Python Program for data example 2

When compared to earlier approaches as shown in *table 6*, the suggested algorithm exhibits a number of significant resource efficiency benefits. Specifically, it effectively reduces the usage of slice registers and LUTs, using only 44 of the 19,200 available slice registers and 143 of the 19,200 available slice LUTs. Compared to the benchmarks offered by other approaches, as those in references [2] and [3], where significantly larger percentages of resources are used, this compares favorably. Furthermore, the suggested approach employs a very small number of completely utilized bit slices—44 out of 143—suggesting minimal waste and effective resource utilization. The algorithm uses 131 of the 220 available IOBs, which is better optimized than the examples in references [4] and [7]. The bonded IOBs also demonstrate an improvement. The suggested algorithm's potential for effective implementation in FPGA or other hardware platforms is demonstrated by its low resource consumption and balanced distribution across available resources, which makes it appropriate for applications where resource constraints are a crucial consideration.

Table 6. Comparison table over previous methods and based on different fields

References	Slice Registers		Slice LUTs		Fully used Bit Slices		Bonded IOBs	
	Used	Available	Used	Available	Used	Available	Used	Available
[2]	192	728400	568	364200	154	606	168	600
[3]	32	11440	75	5720	28	79	141	102
[4]	4364	126800	9326	63400	1257	12433	19	210
[5]	114	8672	223	17344	/	/	114	250
[6]	3319	24576	3090	49152	/	/	325	640
[7]	2101	93120	3523	46560	3525	3525	36	240
[8]	75	150	120	125	87	103	64	96
Proposed	44	19200	143	19200	44	143	131	220

4. CONCLUSION

In communication systems, the combination of checksum techniques and Hamming codes offers a reliable solution for error detection and repair. Data integrity and transmission reliability are improved by the integrated approach, which addresses the shortcomings of each individual method. This technique has been successfully tested and implemented using VHDL on an FPGA platform, demonstrating its practical uses in contemporary communication systems. Additional applications and refinements of this hybrid error management system may be investigated in future study. Many different fields, such as computer memory, embedded systems, satellite communications, modems, shielding cables, connectors and plasma cameras use Hamming codes for error detection and repair. Even though they are widely used, hamming codes have a few drawbacks such in high-noise conditions. It became less effective even though they are excellent at identifying single-bit errors and do not necessitate retransmission. Hamming codes' main benefit is their ability to quickly detect and fix single-bit faults; yet, in situations where there are several errors or in transmission media with a lot of noise, their effectiveness is minimized. On the other hand, the Checksum method detects faults that are both single-bit and multiple-bit, providing a thorough approach to error detection. The delay resulting from having to retransmit complete data sets even in cases where a single-bit error is found is the main disadvantage of checksums. This study suggests an innovative integrated strategy to leverage each method's benefits while addressing its inherent limitations. This technique improves error detection and correction by combining checksums and Hamming codes. In particular, rather than applying Hamming codes to bigger data blocks, like 32 bits, the research presents a bit fragmentation technique where data is separated into smaller segments, such 8-bit groups. The error repair probability increases from 3.125% to 12.5% as a result of this fragmentation. The suggested system achieves over 98% error detection and correction accuracy by combining these techniques. By successfully designing and implementing the integrated approach in VHDL and deploying it on an FPGA Virtex 5 platform, it has been shown to be effective in enhancing the performance and reliability of communication systems.

REFERENCES

[1] M. M. Ali, S. J. Hashim, M. A. Chaudhary, G. Ferré, F. Z. Rokhani, and Z. Ahmad, “A Reviewing Approach to Analyze the Advancements of Error Detection and Correction Codes in Channel Coding with Emphasis on LPWAN and IoT Systems,” IEEE Access, vol. 11, pp. 127077–127097, 2023, doi: 10.1109/ACCESS.2023.3331417.

[2] M. S. O. Anil Dixit, “Design and implementation of hybrid GALOIS filed encoder & decoder,” J. Comput. Technol., vol. 10, no. 3, pp. 1–6, 2022.

[3] G. Patil and D. Patle, “Design and implementation of an enhanced - using Viterbi Decoder,” IOP Conf. Ser. Mater. Sci. Eng., vol. 12, no. 06, pp. 1–6, 2023, doi: 10.1088/1757-899X/331/1/012009.

[4] N. K R, M. K.S, and S. C M, “Fpga Implementation of Object Detection in Background Modeling Using Gaussian Mixture Model,” Int. J. Trendy Res. Eng. Technol., vol. 06, no. 02, pp. 35–43, 2022, doi: 10.54473/ijtret.2022.6207.

[5] A. P. Dewanty and B. A. Wardijono, “Analysis and Design of CRC-32 IEEE 802.3 Generator for 8 Bit Data Using VHDL,” Kilat, vol. 11, no. 1, pp. 78–87, 2022, doi: 10.33322/kilat.v11i1.1536.

[6] A. EL Makhloufi, S. EL Adib, and N. Raissouni, “Highly Efficient Security Level Implementation in Radiation-Tolerance FPGA Using a Combination of AES Algorithm and Hamming Code: LST-SW Case,” Int. J. Electr. Electron. Eng. Telecommun., vol. 12, no. 4, pp. 223–234, 2023, doi: 10.18178/ijeetc.12.4.223-234.

[7] S. Singh, J. V. R. Ravindra, and B. R. Naik, “Prediction of Intermittent Failure by Presage Debacle Model in Network on Chip,” Int. J. Mod. Educ. Comput. Sci., vol. 14, no. 4, pp. 75–88, 2022, doi: 10.5815/ijcnis.2022.04.06.

[8] A. Devrari and A. Kumar, “Turbo encoder and decoder chip design and FPGA device analysis for communication system,” Int. J. Reconfigurable Embed. Syst., vol. 12, no. 2, pp. 174–185, 2023, doi: 10.11591/ijres.v12.i2.pp174-185.

[9] S. G. Priyadharshini, C. Subramani, and J. Preetha Roselyn, “An IOT based smart metering development for energy management system,” Int. J. Electr. Comput. Eng., vol. 9, no. 4, pp. 3041–3050, 2019, doi: 10.11591/ijece.v9i4.pp3041-3050.

[10] Y. Tao, “Research and Application of Several Error Correction Codes in Communication,” Highlights Sci. Eng. Technol., vol. 53, pp. 49–55, 2023, doi: 10.54097/hset.v53i.9681.

[11] P. Belegehalli Siddaiah, M. Puttaswamy, and N. Kamat, “Compact and Energy Efficient QCA Based Hamming Encoder for Error Detection and Correction,” Adv. Electr. Electron. Eng., vol. 21, no. 2, pp. 120–126, 2023, doi: 10.15598/aeee.v21i2.4794.

[12] P. Megha, B. S. Premananda, and N. Kamat, “Area and energy optimized Hamming encoder and decoder for nano-communication,” vol. 75, no. 3, pp. 229–236, 2024.

[13] D. D. T. Tran Do Hon Nhien, Vo Tan Thanh, Nguyen Thanh Khoa, Nguyen Quoc Thang, Nguyen Van Thanh Loc, Huynh Hoang Ha, Nguyen Ngo Lam, “Application of Hamming Code for Error Control in Memory,” J. Tech. Educ. Sci., no. 71B, pp. 19–28, 2022, doi: 10.54644/jte.71b.2022.1141.

[14] M. Sais, N. Rafalia, and J. Abouchabaka, “DNA technology for big data storage and error detection solutions: Hamming code vs Cyclic Redundancy Check (CRC),” E3S Web Conf., vol. 412, 2023, doi: 10.1051/e3sconf/202341201090.

[15] S. Mitsenko, S. Naumenko, I. Rozlomii, and A. Yarmilko, “Information Protection and Recovery Hamming Codes Based’ Hash Technique,” CEUR Workshop Proc., vol. 3513, pp. 64–77, 2023.

[16] R. Alom, N. Shakib, and M. A. Rahaman, “Enhanced Hamming Codes : Reducing Redundant Bit for Efficient Error and Correction,” in 2023 5th International Conference on Sustainable Technologies for Industry 5.0 (STI), 2024, no. December 2023, pp. 1–7.

[17] X. Wei et al., “ReIPE: Recycling Idle PEs in CNN Accelerator for Vulnerable Filters Soft-Error Detection,” ACM Trans. Archit. Code Optim., 2024, doi: 10.1145/3674909.

[18] T. Manivannan, Y. Basheerbaba, B. A. Kumar, and D. N. Siva, “Advanced VLSI Technique for Error Detection and Correction in Space Systems,” vol. 5, no. 2, pp. 7–18, 2024.

[19] M. P. Kiogora, Loyford Njagi, and Josephine Mutembei, “Errors, error detection and correction efficiency in the container number code,” African J. Sci. Technol. Soc. Sci., vol. 2, no. 2, pp. 93–103, 2024, doi: 10.58506/ajstss.v2i2.166.

[20] A. H. Saleh and M. S. Mohammed, “Enhancing Data Security through Hybrid Error Detection: Combining Cyclic Redundancy Check (CRC) and Checksum Techniques,” no. August, 2024, doi: 10.37391/IJEER.120312.

[21] Louis Narmour, Steven Derrien, and Sanjay Rajopadhye, Automatic Algorithm-Based Fault Detection (AABFD) of Stencil Computations, vol. 1, no. 1. Association for Computing Machinery, 2023.

[22] S. Priyadarshan, H. Nguyen, R. Chouhan, and R. Sekar, “SAFER: Efficient and Error-Tolerant Binary Instrumentation,” 32nd USENIX Secur. Symp. USENIX Secur. 2023, vol. 2, pp. 1451–1468, 2023.

[23] Y. Huang, “Quaternary checksum, redundancy and Hamming code,” no. May, 2022, [Online]. Available: https://github.com/tom123jack321/.

[24] V. Sokolovskyi, E. Zharikov, and S. Telenyk, “Development of the Method of Detecting and Correcting Data Transmission Errors in Iot Systems for Monitoring the State of Objects,” Eastern-European J. Enterp. Technol., vol. 1, no. 9(127), pp. 22–33, 2024, doi: 10.15587/1729-4061.2024.298476.

[25] A. Hadi Saleh, “Design of Hamming Code for 64 Bit Single Error Detection and Correction Using Vhdl,” Diyala J. Eng. Sci., vol. 8, no. 3, pp. 22–37, 2015, doi: 10.24237/djes.2015.08305.

[26] Delphine Mary. P and S. A, “Design and Implementation of Triplication Error Correction Using Hamming Code,” Irish Interdiscip. J. Sci. Res., vol. 07, no. 03, pp. 106–114, 2023, doi: 10.46759/iijsr.2023.7312.

[27] C. Ding, Z. Sun, and Q. Yan, "The Support Designs of Several Families of Lifted Linear Codes," pp. 1–15, 2024, [Online]. Available: http://arxiv.org/abs/2407.15104.

[28] A. O. Hoori, "A Modified 2D-Checksum Error Detecting Method for Data Transmission in Noisy Media," J. Eng., vol. 19, no. 08, pp. 992–998, 2023, doi: 10.31026/j.eng.2013.08.05.

[29] P. Koopman, "An Improved Modular Addition Checksum Algorithm."

[30] P. Zhang, "Polynomial Intermediate Checksum for Integrity under Releasing Unverified Plaintext and Its Application to COPA," Mathematics, vol. 12, no. 7, 2024, doi: 10.3390/math12071011.

[31] H. Pereira et al., "SEGUID v2 : Extending SEGUID checksums for circular , linear , single- and double-stranded biological sequences," 2024.