International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 03 | March - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

VLSI-optimized Two-Bit Error Detection and One-Bit Error Correction for Enhanced Data Integrity

Dr.R.K.Navandar[1], Kalyani Awatade[2], Gauri Ghadi[3], Poonam Gaikwad[4]

[1]Dr. R. K. Navandar, Dept. Of ENTC, JSPM's JSCOE, Pune, Maharashtra, India
[2]Kalyani Awatade , Dept. Of ENTC, JSPM's JSCOE, Pune, Maharashtra, India
[3]Gauri Ghadi , Dept. Of ENTC, JSPM's JSCOE, Pune, Maharashtra, India
[3]Poonam Gaikwad , Dept. Of ENTC, JSPM's JSCOE, Pune, Maharashtra, India

Abstract : This project presents a new VLSI-optimized solution for enhancing data integrity through 2-bit error detection and 1-bit error correction. It consist some limitations in existing approaches our project aimed to advance the field by introducing an efficient algorithm in Very Large Scale Integration (VLSI) system. We have worked on studying the reason that causes errors in digital transmission process and receiving of data and developed a code in Very High Speed Integrated Circuit Hardware Descriptive Language (VHDL) that optimizes error detection and correction processes, addressing challenges in contemporary data storage and communication systems.

Xilinx ISE Design Suite 14.7 was used for performing the implementation of the code and observing the relevant output in order to know where the error bit was. Our project aimed to detect upto 2 bits and correct 1 bit. Simulation results demonstrated that the proposed approach is capable of achieving high reliability digital systems such as flight critical systems in airplanes, space shuttles, and missile electronics systems.

Key Words: Error detection, error correction, VLSI, digital systems, data storage.

1. INTRODUCTION

In In the rich tapestry of digital communication and data transmission, the relentless pursuit of enhancing accuracy and reliability has spurred profound advancements in error detection and correction methodologies. This research endeavors to contribute to this narrative by delving into the historical context of prior studies and elucidating our hypothesis supported by an overview of the anticipated results.

Our exploration into the realms of hamming code linear block code FPGA (Field-Programmable Gate Array) Architecture and VLSI (Very Large Scale Integration Language) has set the stage for a novel approach in 2-bit error detection and 1-bit error correction.

Error detection and correction is widely used in many application fields especially in communication systems, satellite and space communications, network communications, cellular telephone networks, and any other of digital data communication. In addition, it is used in computing applications, data compression, and system coding. In noisy communication system the data transmission from transmitter to receiver suffer from errors. To overcome this problem and get error free data, there are number of error detection and correction techniques can be used. Linear block code (LBC) is one of the most common used error detection and correction methods. Hamming Code is a special case of LBC error detection and correction codes which is used to detect single or double bit errors and correct single bit errors that occur within data when it is transmitted from one device to another [1]. In traditional communication systems, the implemented hardware of error detection and correction part is designed to deals with fixed number of information data (bits) and has no ability to be reprogramed easily to meet other requirements of different communication system. To overcome this problem a flexible hardware system can be used such as Field programmable Gated Array (FPGA). Hamming code system based on FPGA is utilized in this work. Many sub-systems can implemented to consist the overall system hardware, each sub-systems run with its own program and need to be executed correctly, as well as, whenever data is stored or transmitted, there are chances that at least one or more bits will be an incorrect value. The transmission systems are exposing to get bits error values in either the instruction or data causing undesirable crashes or other system failures. Therefore, utilizing Hamming Code inside an embedded system is considered with high priority in modern industrial fields.

2. METHODOLOGY –

The key to the Single bit error correction is the use of extra parity bits to allow the identification of a single error. Two methods (even parity, odd parity) for generating redundancy bits that Hamming code need it. Number of redundancy bits are generated (Check bits) is calculate according to (1). This redundancy bits depend on the number of information data bits .

$(2r) \geq (k + r + 1)$ (1)

where k is the data bits length, and r is the bits to find the check bits that will add to dataThe operation of hamming code extended can be summarize as following: a. Mark all bit positions that are powers of two as parity bits (positions 1, 2, 4, 8, 16, 32, 64, etc.).

b. All other bit positions are for the data to be encoded (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.).

c. The last bit is added for Parity bit.

d. Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips.

- Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1, 3, 5, 7, 9, 11, 13, 15,...)

- Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2, 3, 6, 7, 10, 11, 14, 15, ...)

- Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc. (4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, ...)

- Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15, 24-31, 40-47, ...) and so on for the position 16 and 32 …etc. e. Set a Parity bit to 1 if the total number of ones in the positions it checks is odd (XOR operation between all bits) . Set a parity bit to 0 if the total number of ones in the positions it checks is even. f. To test packet data received, the XOR is applied on all bits to determine if there is any error in Parity bit; Parity bit is extracted; same algorithm is applied on rest bits to generate hamming code. g. If hamming code is zero and Parity bit is zero, then there are no error in received packet data. If hamming code is not zero and Parity bit is one, then there is one error in packet data and correction is capable by invert the bit location that pointed by hamming code value. If hamming code is not zero and Parity bit is zero, then there are two errors or even errors in packet data and cannot be corrected.

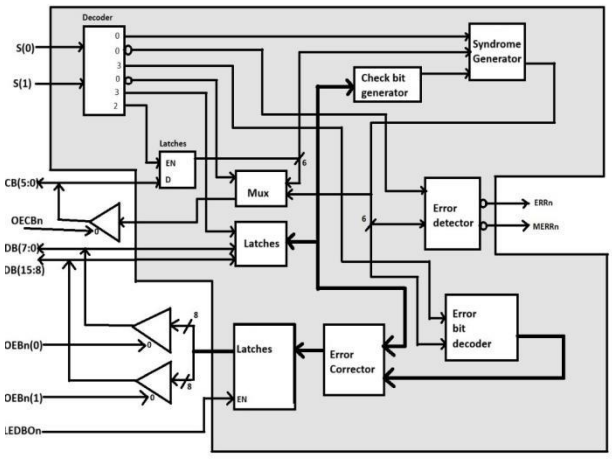

Fig 1. Block diagram

1.

1. **Decoder**:

a) The decoder receives input signals labeled as **S(0)** and **S(1)**
b) Its purpose is to decode these signals, which likely represent encoded data or error-affected data.

2. **Check Bit Generator**:

a) This block generates check bits based on the received data.
b) The check bits are calculated using Hamming code principles (XOR operations on specific data bits).
c) The check bits are essential for error detection and correction.

3. **Syndrome Generator**:

a) The syndrome generator computes the syndrome (error pattern) based on the received data and the calculated check bits.
b) The syndrome provides information about the presence and location of errors.

4. **Latches**:

a) These are memory elements that store intermediate data or signals.
b) They likely hold data during the decoding process.

5. **Mux (Multiplexer)**:

a) The multiplexer selects one of several input signals and routes it to the output.

b) It may be used to choose between different data paths or to select corrected data.

6. **Error Detector**:

a) The error detector examines the received data and the calculated check bits.
b) If an error is detected (mismatch between received and calculated check bits), it triggers an error alert (labeled as **ERRn**).

7. **Error Corrector**:

a) The error corrector uses information from the latches and the error alert to correct detected errors.
b) It modifies the received data to correct single-bit errors.

8. **Error Bit Decoder**:

a) This component ensures that the corrected data is now error-free.
b) It may perform additional checks to validate the correctness of the corrected data.

➢ **SIGNIFICANCE OF HAMMING CODE:**

The selection of check bits (P1, P2, P4, P8, P16, and P32) follows a specific pattern based on their positions as powers of 2. These positions are chosen to ensure efficient error detection and correction. Here's why:

1. **Position as Powers of 2**:
- The check bits are placed at positions corresponding to powers of 2 (1, 2, 4, 8, 16, and 32).
- These positions allow us to cover different subsets of data bits in a systematic way.
- For example, P1 covers all bits with the least significant bit set (bit 1, bit 3, bit 5, etc.).

2. **Parity Calculation**:
- Each check bit is calculated by performing an XOR operation over specific data bits.
- The XOR operation checks for parity (even or odd) among the relevant bits.
- By placing the check bits at powers of 2, we ensure that each data bit participates in the calculation of exactly one check bit.

3. **Efficient Error Detection and Correction**:
- The chosen positions allow us to detect and correct single-bit errors efficiently.
- If an error occurs during transmission, the check bits will reveal the erroneous bit position.For example, if P1 is incorrect, we know that there's an issue with the data bits at positions 1, 3, 5, 7, 9, 11, 13, or 15.

4. **Redundancy and Robustness**:
- The redundancy introduced by the check bits helps in error correction.
- The more check bits we have, the better our ability to detect and correct errors.
- However, we strike a balance between redundancy and efficiency to minimize overhead.

IMPLEMENTATION:

Encoding (Adding Check Bits):

1. Calculates the values of each check bit using XOR operations:

2.

P1 = XOR(D1, D3, D5, D7, D9, D11, D13, D15)

P1 = D1 ⊕ D3 ⊕ D5 ⊕ D7 ⊕ D9 ⊕ D11 ⊕ D13 ⊕ D15 = 0

P2 = XOR(D2, D3, D6, D7, D10, D11, D14, D15)

P2 = D2 ⊕ D3 ⊕ D6 ⊕ D7 ⊕ D10 ⊕ D11 ⊕ D14 ⊕ D15 = 1

P4 = XOR(D4, D5, D6, D7, D12, D13, D14, D15)

P4 = D4 ⊕ D5 ⊕ D6 ⊕ D7 ⊕ D12 ⊕ D13 ⊕ D14 ⊕ D15 = 0

P8 = XOR(D8, D9, D10, D11, D12, D13, D14, D15)

P8 = D8 ⊕ D9 ⊕ D10 ⊕ D11 ⊕ D12 ⊕ D13 ⊕ D14 ⊕ D15 = 1

P16 = XOR(D16, D17, D18, …, D31)

P16 = D16 (no XOR needed for P16; it's the 16th data bit) = 1

P32 = 0 (not present in the 16-bit data)

Decoding:(Error Detection and Correction):

Received Data (22 bits): D1 to D16 (16 data bits) + P1, P2, P4, P8, P16 (6 check bits)

Received Check Bits: P1 = 0, P2 = 1, P4 = 0, P8 = 1, P16 = 1, P32 = 0 (not present in the 16-bit data)

P1 = XOR(D1, D3, D5, D7, D9, D11, D13, D15) = 0

P2 = XOR(D2, D3, D6, D7, D10, D11, D14, D15) = 1

P4 = XOR(D4, D5, D6, D7, D12, D13, D14, D15) = 0

P8 = XOR(D8, D9, D10, D11, D12, D13, D14, D15) = 1

P16 = D16 (no XOR needed for P16; it's the 16th databit) = 1

Compare the calculated check bits with the received check bits: If any discrepancy is found, identify the erroneous bit position.
Correct the error (if applicable): If only one check bit is incorrect, flip the corresponding data bit.
If more than one check bit is incorrect, the error is uncorrectable.

Therefore, the corrected data bits are as follows:

D1(1),D2(0),D3(0),D4(0),D5(1),D6(0),D7(0),D8(0),

D9(1),D10(0),D11(0),D12(0),D13(0),D14(0),D15(0),

D16(0).

1. RESULTS /DISCUSSION:

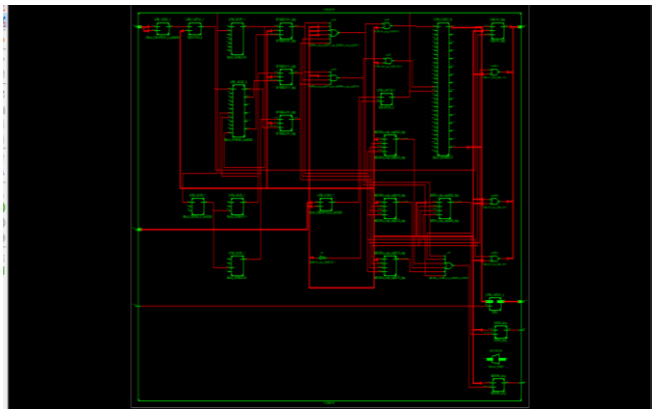

Fig 2. RTL view schematic

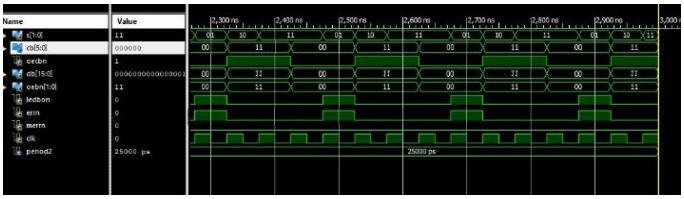

Fig 3. Simulation Waveform

3. CONCLUSION:

In conclusion, our VLSI-based error detection and correction system, leveraging Hamming code, represents a significant step toward ensuring data integrity in digital communication and storage. By combining principles from circuit design, digital communication, and error-correcting codes, we've created a robust solution. However, ongoing research and real-world testing will be crucial to fine-tune the trade-offs between error correction capability, memory overhead, and practical implementation. As technology advances, our commitment to reliable data transmission remains unwavering, safeguarding critical systems across various domains.

4. REFERENCE:

[1] Forouzan BA. Data Communication and Networking. Fourth edition. McGrawHill Ltd. 2007: 267-299.

[2] St. Onge LM, Areibi S. VHDL for Digital Design. Technical Report 2003-01P School of engineering,university of Guelph, Canada. 2003.

[3] Perry DL. VHDL Programming by Example. Fourth Edition. McGraw-Hill Ltd. 2002.

[4] Mohammed ZG, Hamdoon AMA, Aziz MS. Scheduling lecturer system based on FPGA. IEEE International Conference on Advances in Sustainable Engineering and Applications (ICASEA). 2018: 54-58.

[5] Peckol JK. Embedded Systems a Contemporary Design Tool. First Edition. 2007: 597-648.

[6] Mr. Ravindra H. Adekar , Mr. Dhananjay D. Khumane., "Forward Error Correction Techniques Using VLSI" , Sep-2012.

[7] Supraja,C.Kanmani,Dharani.N. ,"VLSI Implementation of Error Detection and Correction Codes for Space Engineering" , 2021.

[8] Vijayakumara Y , Byregowda B. ,"A VLSI Implementation of hamming code algorithm using FPGA architecture." , 2020.

[9] K.C.Chang,"Digital Systems Design with VHDL and Synthesis." , 200.