NIRMA UNIVERSITY

INSTITUTE OF TECHNOLOGY

NAAC ACCREDITED 'A+' GRADE

NIRMA INSTITUTE OF TECHNOLOGY

DEPARTMENT

OF

ELECTRONICS AND COMMUNICATION ENGINEERING

(2EC202)

FIELD PROGRAMMABLE GATE ARRAY - SYSTEM DESIGN

SEMESTER -IV

HAMMING CODE ENCODER AND DECODER

SUBMITTED BY:

KACHA PREM DHARMENDRABHAI

22BEC064

FACULTY:

PROF. AKASH MECWAN

ABSTRACT

This report explains the design and simulation of a Hamming code encoder and decoder using Verilog HDL, with the verification conducted through RTL (Register-Transfer Level) simulation and TTL (Transistor-Transistor Logic) implementation. Hamming codes are widely utilized for error detection and correction in digital communication systems due to their ability to detect and correct single-bit errors and detect some double-bit errors. The encoder module translates input data bits into redundant code words with parity bits, while the decoder module analyses received code words to identify and correct errors. The Verilog HDL implementation enables a detailed description of the encoder and decoder logic, facilitating RTL simulation to verify functionality and performance. Additionally, the design can be synthesized into TTL circuits for hardware realization, ensuring compatibility with real-world digital systems. Through this integrated approach, the paper demonstrates the efficacy of Hamming code encoder and decoder designs, providing a comprehensive framework for error control in digital communication systems.

Keywords

1. Hamming code
2. Parity bit
3. Error detection.
4. Error correction.
5. Linear code
6. Digital communication.

Literature Survey/State of the Art Technology Available

Hamming code stands as a well-established error-correction method extensively explored in scholarly works. Originating from Richard Hamming's work in 1947, it has found broad utility across digital communication, data storage, and computer networks. As a linear code, Hamming code excels in identifying and rectifying single-bit errors, and it possesses the capability to detect double-bit errors as well.

In practical implementations, the encoding and decoding processes of Hamming code are typically realized through the utilization of combinational logic circuits. This approach represents the principles of Boolean logic to efficiently translate input data bits into a redundant code word during the encoding phase. This code word is augmented with parity bits strategically positioned to enable the subsequent detection and correction of errors during the decoding phase.

The encoding process involves mapping the input data bits onto the Hamming code's codeword space, where redundancy is judiciously introduced to facilitate error detection and correction. Parity bits, generated through logical operations on specific subsets of the data bits, serve as markers, enabling the decoder to pinpoint and rectify any discrepancies that may arise during transmission or storage.

Equation of Parity Bit Generation

$$2^p=p+k+1 \qquad ; k = \text{input} \ \& \ p = \text{parity bit}$$

Limitations or Drawbacks of currently available technology

There are few limitations or drawbacks of current technology:

- **Complexity:** Implementing higher-order Hamming codes to detect and correct more errors increases the complexity of the system, requiring more computational resources and making it harder to manage.

- **Overhead:** Hamming codes introduce additional redundant bits to detect and correct errors, which can increase the overall data size and overhead, reducing the efficiency of data transmission or storage.

- **Limited Error Correction Capability:** While Hamming codes are effective at correcting single-bit errors, they may not be able to correct certain types of errors, such as burst errors or errors occurring in multiple bits within the same code word.

- **Sensitivity to Noise:** In noisy communication channels or environments with high levels of interference, Hamming codes may struggle to accurately detect and correct errors, leading to a higher probability of undetected errors slipping through.

Proposed solution/methodology

- **Hybrid Coding Schemes:** Integrating Hamming codes with other error-correction codes, such as Reed-Solomon codes or Turbo codes, in a hybrid coding scheme can enhance error correction capabilities. By leveraging the strengths of multiple codes, hybrid coding schemes can achieve improved error correction performance while minimizing the overhead associated with higher-order Hamming codes.

- **Feedback-Based Encoding:** Introducing feedback mechanisms into the encoding process can help tailor the redundancy introduced by Hamming codes to the specific error patterns observed in the communication channel. By incorporating feedback from the decoder into the encoding process, the system can adaptively adjust the distribution of parity bits to better suit the prevailing error characteristics, enhancing error correction performance.

- **Adaptive Code Rate:** Implementing adaptive code rate schemes where the coding rate dynamically adjusts based on the channel conditions can help mitigate the trade-off between error detection and correction. In this approach, the system can switch between higher and lower code rates depending on the observed error rates, optimizing performance based on the prevailing conditions.

Block Diagram

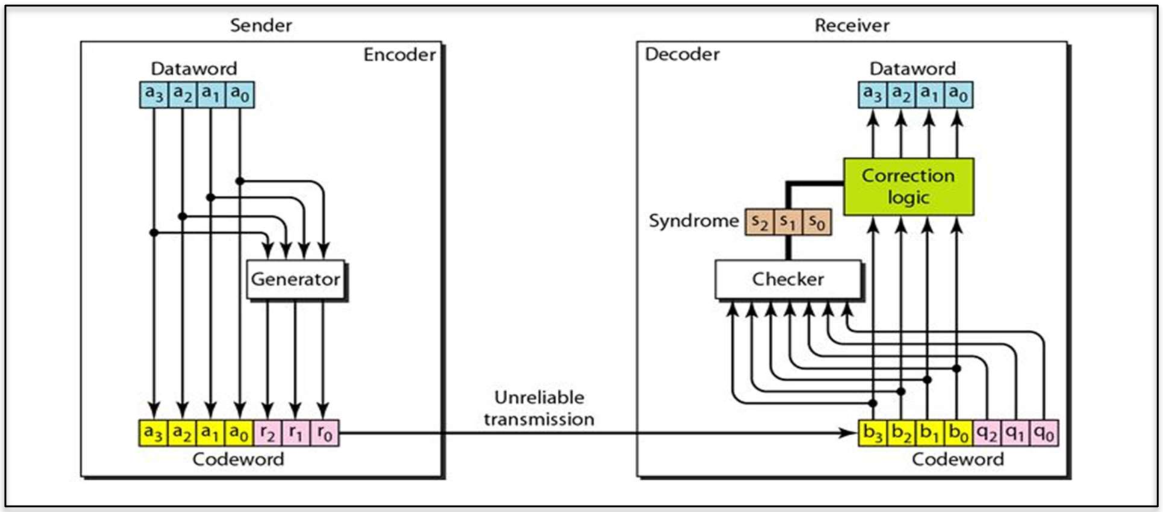
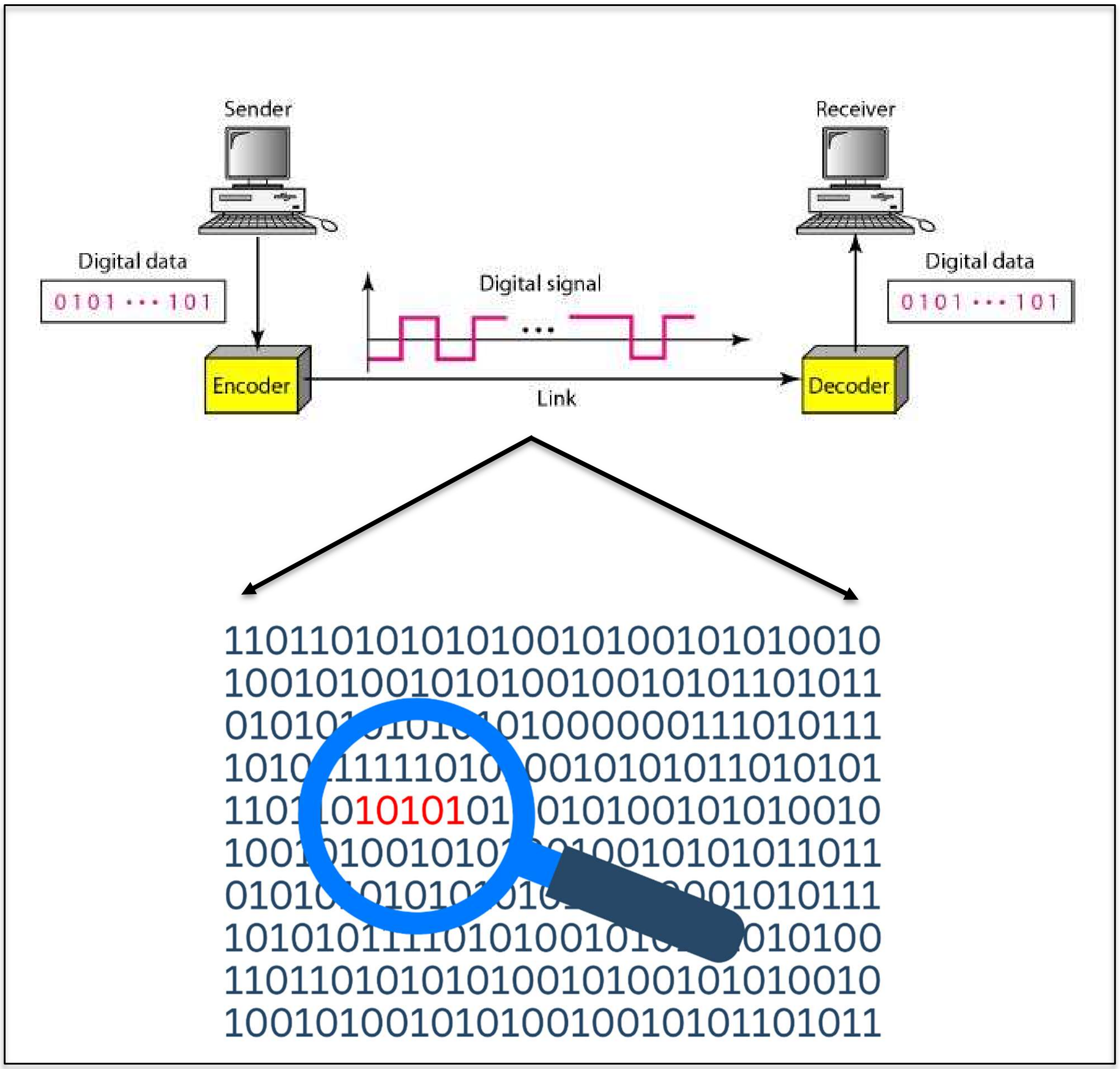


Figure1: Block Diagram of Hamming Code

https://www.researchgate.net/figure/Block-diagram-of-LDPC-encoder-and-decoder-Hamming-code-Hamming-codes-are-linear-error_fig4_319102009

Hamming (7,4) Error Detection and Correction



https://bignet.in/blog/1286/digital-to-digital-conversion-in-english

https://hyperskill.org/learn/step/29178

Flow of the code

1. **Hamming Code Generation:** The code generates Hamming (7,4) codes for a 4-bit data input by adding three parity bits. The input data is expanded to 7 bits, with the three parity bits calculated to ensure error detection and correction.

2. **Error Detection:** The code detects single-bit errors in the received data. It calculates syndrome bits (z4, z2, z1) based on the received data and compares them with the parity bits to identify errors.

3. **Error Correction:** The code includes error correction logic to rectify single-bit errors in the received data. Corrected data is provided as the output, ensuring that data integrity is maintained.

4. **Parity Bit Calculation:** Parity bits p1, p2, and p4 are calculated using XOR operations on specific bits of the input data. These parity bits play a crucial role in detecting and correcting errors.

5. **Syndrome Bit Calculation:** Syndrome bits z4, z2, and z1 are computed based on XOR operations between the received data and the calculated parity bits. These syndrome bits are used to identify errors in the received data.

6. **Data Rearrangement:** The code rearranges the data bits for output in the Hamming (7,4) format. The input data restructured to include parity bits at specific positions in the output.

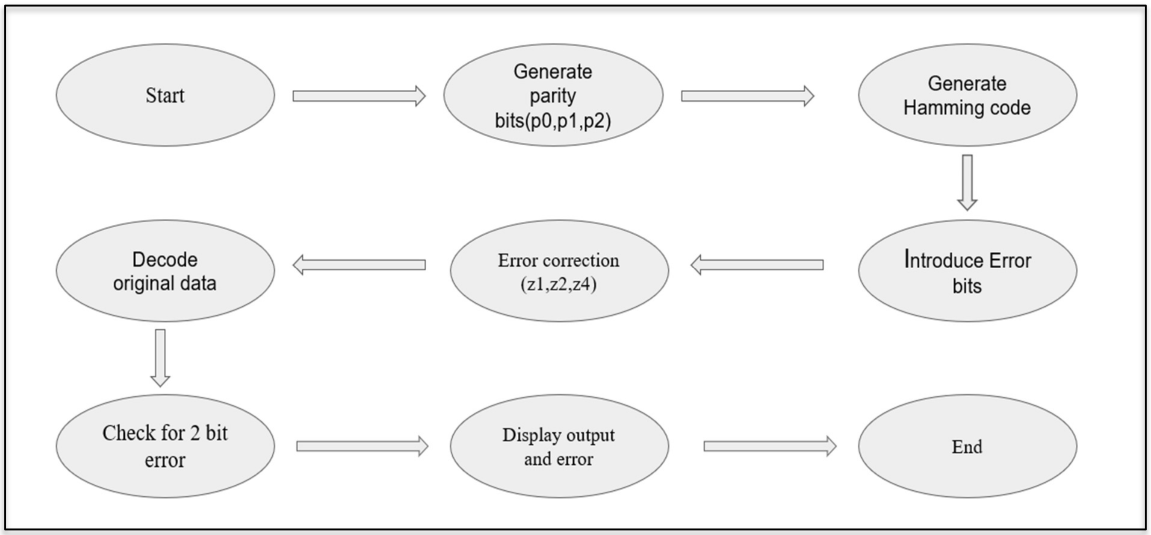


Figure2: Flow of Code

RTL VIEW

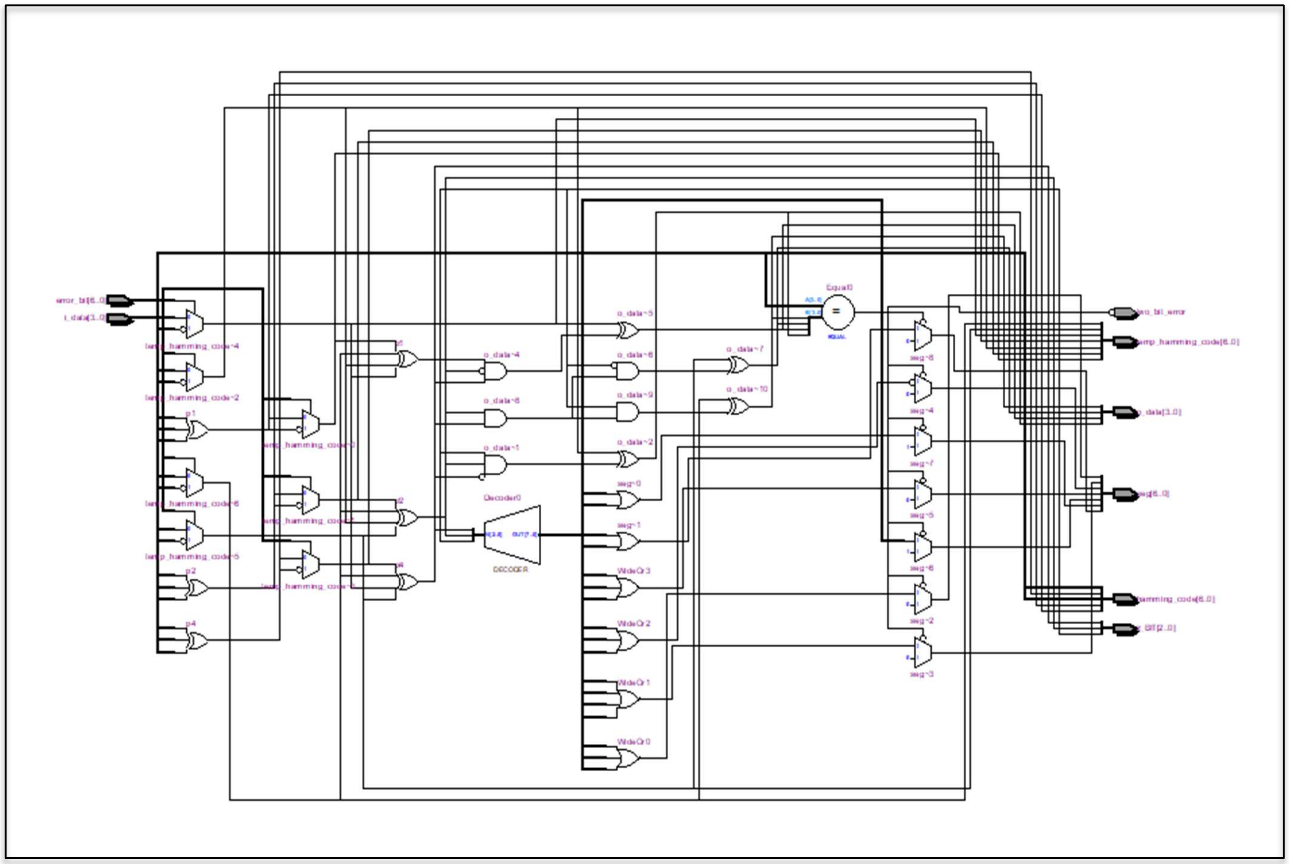
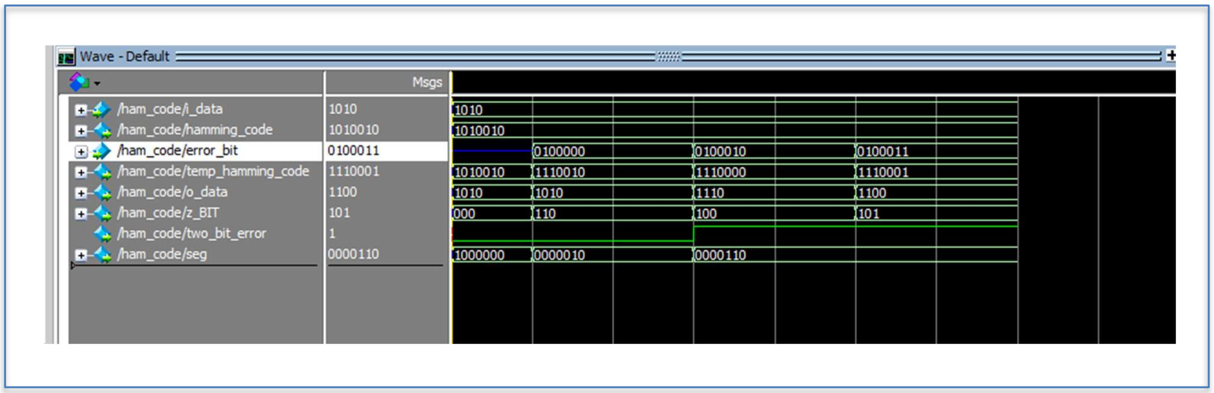


Figure3: RTL View

RTL WAVEFORM



```
Wave - Default
                                Msgs
/ham_code/i_data                1010       1010
/ham_code/hamming_code          1010010    1010010
/ham_code/error_bit             0100011              0100000   0100010   0100011
/ham_code/temp_hamming_code     1110001    1010010   1110010   1110000   1110001
/ham_code/o_data                1100       1010      1010      1110      1100
/ham_code/z_BIT                 101        000       110       100       101
/ham_code/two_bit_error         1
/ham_code/seg                   0000110    1000000   0000010   0000110
```

Figure4: RTL Simulation

TTL VIEW

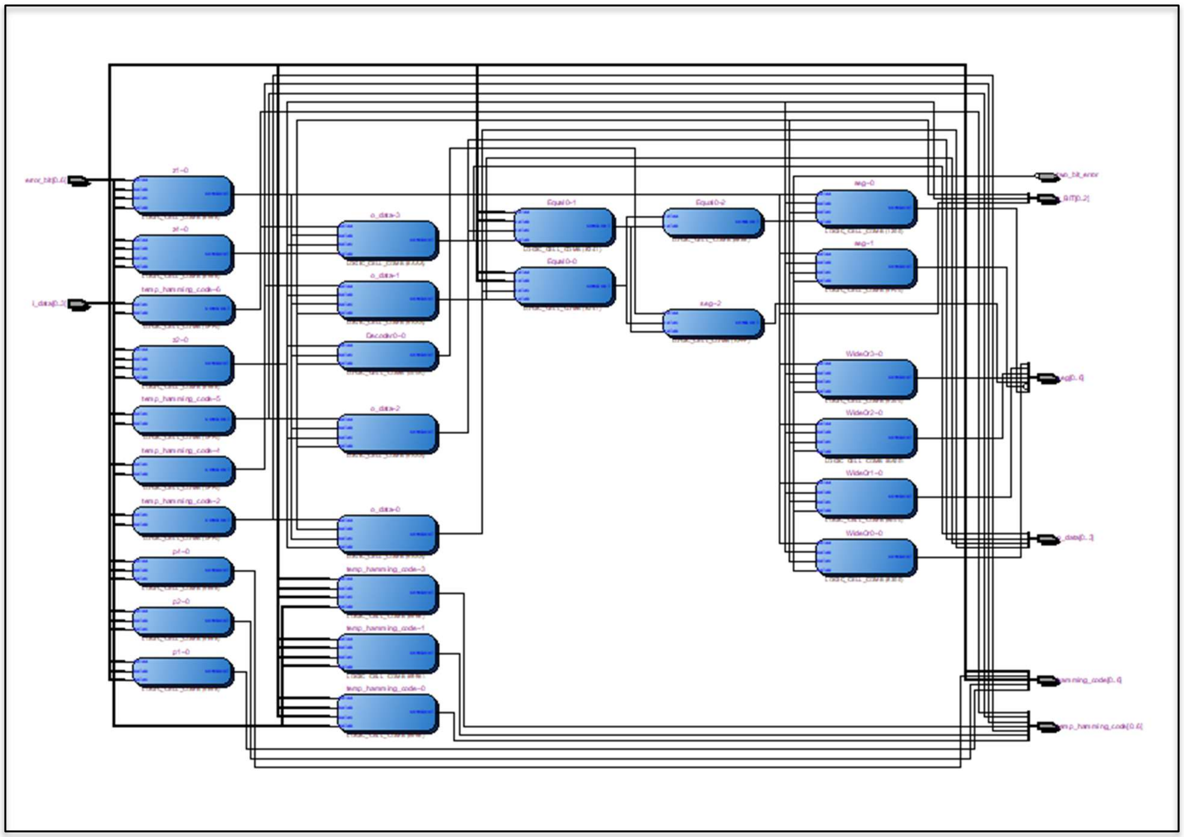


Figure5: TTL View

Future Scopes:

Hamming Codes are widely used in various applications where error detection and correction are crucial for maintaining data integrity and reliability. Some of the wide-ranging applications of Hamming Codes include:

- **Digital Communication Systems:** Hamming Codes are commonly employed in digital communication systems such as wired and wireless networks, satellite communication, and optical fibre communication. They help ensure that data transmitted over these channels is received accurately, even in the presence of noise and interference..

- **Digital Television and Radio Broadcasting:** In digital broadcasting, Hamming Codes ensure the accuracy of audio and video data transmission, delivering high-quality content to viewers and listeners.

- **Medical Imaging:** In medical imaging systems like X-ray, MRI, and CT scanners, Hamming Codes are employed to reduce the impact of noise and artifacts, ensuring accurate medical diagnoses.

CONCLUSION

Through this project, I learned about the concept of Hamming code encoder and decoder and its applications. The Verilog code implements Hamming (7,4) error detection and correction, enhancing data reliability in digital systems. It demonstrates the capacity to detect and rectify single-bit errors, while efficiently identifying two-bit errors. This code underscores its importance in error-tolerant applications.

Despite their effectiveness, Hamming codes do possess limitations, such as increased complexity with higher-order implementations, overhead due to additional redundant bits, and sensitivity to certain error patterns and noise levels. However, through proposed solutions like feedback-based encoding and adaptive code rate schemes, these limitations can be mitigated to some extent, enhancing the overall performance and reliability of Hamming code-based systems. Thus, Hamming codes continue to be a fundamental tool in ensuring data integrity and reliability in various digital systems.

BIBLIOGRAPHY

1. R. W. Hamming, "Error Detecting and Error Correcting Codes," Bell System Technical Journal

2.https://www.researchgate.net/figure/Block-diagram-of-LDPC-encoder-and-decoder-Hamming-code-Hamming-codes-are-linear-error_fig4_319102009

3. https://www.youtube.com/watch?v=h0jloehRKas

4. https://www.youtube.com/watch?v=X8jsijhllIA

5.https://i0.wp.com/krazytech.com/wp- content/uploads/untitled1.jpg?fit=516%2C290&ssl=1

6.https://bignet.in/blog/1286/digital-to-digital-conversion-in-e nglish

7. https://hyperskill.org/learn/step/29178

APPENDIX *(CODE)*

```
module ham_code(
   input      [3:0]  i_data,
   output  reg [6:0]  hamming_code,
   input      [6:0]  error_bit,
   output  reg [6:0]  temp_hamming_code,
   output  reg [3:0]  o_data,
   output  reg [2:0]  z_BIT,
          output  reg two_bit_error,
          output  reg [6:0]  seg
);

//PARITY BIT GENERATION
reg p1, p2, p4;

always@(*)
begin
   p1 = i_data[0] ^ i_data[1] ^ i_data[3];
   p2 = i_data[0] ^ i_data[2] ^ i_data[3];
   p4 = i_data[1] ^ i_data[2] ^ i_data[3];
```

```
end


// HAMMING CODE
// INPUT  I3 I2 I1 -  -  I0 -
// CODE   I3 I2 I1 P2 P1 I0 P0
always@(*)
begin
   hamming_code = {i_data[3:1],p4,i_data[0],p2,p1};
end



always@(*)
begin
   temp_hamming_code = hamming_code;
   if (error_bit[0] == 1) begin
     temp_hamming_code[0] = ~temp_hamming_code[0];
   end
   if (error_bit[1] == 1) begin
     temp_hamming_code[1] = ~temp_hamming_code[1];
   end
   if (error_bit[2] == 1) begin
     temp_hamming_code[2]= ~temp_hamming_code[2];
   end
   if (error_bit[3] == 1) begin
     temp_hamming_code[3] = ~temp_hamming_code[3];
   end
   if (error_bit[4] == 1) begin
     temp_hamming_code[4] = ~temp_hamming_code[4];
   end
   if (error_bit[5] == 1) begin
     temp_hamming_code[5] = ~temp_hamming_code[5];
   end
   if (error_bit[6] == 1) begin
     temp_hamming_code[6] = ~temp_hamming_code[6];
   end
end


// ERROR CORRECTION
reg z4, z2, z1;

always @(*) begin
   z1 = temp_hamming_code[2] ^ temp_hamming_code[4] ^ temp_hamming_code[6] ^
temp_hamming_code[0];
   z2 = temp_hamming_code[2] ^ temp_hamming_code[5] ^ temp_hamming_code[6] ^
temp_hamming_code[1];
   z4 = temp_hamming_code[4] ^ temp_hamming_code[5] ^ temp_hamming_code[6] ^
temp_hamming_code[3];
```

```
end


// ORIGINAL 4-INPUT DATA AS THE OUTPUT
always @(*) begin
        o_data[0] = temp_hamming_code[2] ^ (~z4 & z2 & z1);
        o_data[1] = temp_hamming_code[4] ^ (z4 & ~z2 & z1);
        o_data[2] = temp_hamming_code[5] ^ (z4 & z2 & ~z1);
        o_data[3] = temp_hamming_code[6] ^ (z4 & z2 & z1);
end

always @(*) begin
        if (o_data == i_data)
                        two_bit_error =0;
                        else
                        two_bit_error =1;
end

always @(*) begin
z_BIT ={z4,z2,z1};
end

// ERROR DISPLAY ON SEVEN SEGMENT
always @(*) begin
if (two_bit_error == 1)
begin
seg = 7'b0000110;
end
else begin
        case (z_BIT)
            3'b000: seg = 7'b1000000;
            3'b001: seg = 7'b1111001;
            3'b010: seg = 7'b0100100;
            3'b011: seg = 7'b0110000;
            3'b100: seg = 7'b0011001;
            3'b101: seg = 7'b0010010;
            3'b110: seg = 7'b0000010;
            3'b111: seg = 7'b1111000;
            default:seg = 7'b1111111;
        endcase
end
end
endmodule
```