🟢 Hamming Code – Error Detection & Correction

📌 Introduction to Hamming Code

Hamming code is a linear error-detecting and error-correcting code developed by Richard Hamming in 1950. It is used in digital systems to ensure data reliability in communication and memory storage.

✅ Features:

✔️ Detects up to 2-bit errors and corrects 1-bit errors. ✔️ Efficient: Uses a small number of parity bits for error correction. ✔️ Widely used in computer memory (ECC RAM), networking, and wireless communication.

📌 General Formula:

A Hamming code follows the format (n, k), where:

- **n** = Total number of bits (data + parity)
- **k** = Number of data bits
- **r** = Number of parity bits (where **n = k + r**)

📌 Common Hamming Code Variants:

🔹 **(7,4) Hamming Code** → 4-bit data, 3 parity bits, 7-bit codeword. 🔹 **(15,11) Hamming Code** → 11-bit data, 4 parity bits, 15-bit codeword.

🏗️ Working of (7,4) Hamming Code

Hamming (7,4) encodes 4-bit data into a 7-bit codeword using 3 parity bits.

📌 Bit Arrangement in (7,4) Code

- **4 Data Bits**: D1, D2, D3, D4
- **3 Parity Bits**: P1, P2, P4
- **7-bit Codeword**: (P1, P2, D1, P4, D2, D3, D4)

✅ Encoding Process

Each parity bit is responsible for checking specific bits. The parity bits are calculated as follows:

- **P1 (bit 1) = D1 ⊕ D2 ⊕ D4**
- **P2 (bit 2) = D1 ⊕ D3 ⊕ D4**
- **P4 (bit 4) = D2 ⊕ D3 ⊕ D4**

✅ Example (Encoding 1011 in Hamming Code)

📜 Encoding Structure:

Bit Position	P1	P2	D1	P4	D2	D3	D4
Data Value	?	?	1	?	0	1	1
Parity Calc	D1 ⊕ D2 ⊕ D4	D1 ⊕ D3 ⊕ D4	1	D2 ⊕ D3 ⊕ D4	0	1	1
Final Codeword	0	1	1	1	0	1	1

📌 Encoded 7-bit Codeword: 0111011

📜 Verilog Code for Hamming (7,4) Encoder & Decoder

🔹 Hamming74_Encoder_Decoder.v

```
// Hamming Code Encoder (7,4) in Verilog
module HammingEncoder (
    input  [3:0] data_in,  // 4-bit input data
    output [6:0] code_out  // 7-bit encoded Hamming code
);

    // Parity bit calculations
    assign code_out[0] = data_in[0] ^ data_in[1] ^ data_in[3]; // P1
    assign code_out[1] = data_in[0] ^ data_in[2] ^ data_in[3]; // P2
    assign code_out[2] = data_in[0]; // D1
    assign code_out[3] = data_in[1] ^ data_in[2] ^ data_in[3]; // P4
    assign code_out[4] = data_in[1]; // D2
    assign code_out[5] = data_in[2]; // D3
    assign code_out[6] = data_in[3]; // D4

endmodule

// Hamming Code Decoder (7,4) in Verilog
module HammingDecoder (
    input  [6:0] code_in,  // 7-bit received code
    output [3:0] data_out, // 4-bit corrected output
    output reg   error     // Error detection flag
);
    wire p1, p2, p4;
    wire [2:0] parity;
```

```
    // Parity check bits
    assign p1 = code_in[0] ^ code_in[2] ^ code_in[4] ^ code_in[6];
    assign p2 = code_in[1] ^ code_in[2] ^ code_in[5] ^ code_in[6];
    assign p4 = code_in[3] ^ code_in[4] ^ code_in[5] ^ code_in[6];

    assign parity = {p4, p2, p1};

    always @(*) begin
        error = (parity != 3'b000); // If parity is non-zero, error exists
    end

    // Correct the received code if error exists
    reg [6:0] corrected_code;
    always @(*) begin
        if (error)
            corrected_code = code_in ^ (1 << (parity - 1));
        else
            corrected_code = code_in;
    end

    // Extract original data
    assign data_out = {corrected_code[6], corrected_code[5], corrected_code[4],
corrected_code[2]};

endmodule
```

🔹 tb_Hamming74.v (Testbench)

```
module tb_Hamming74;
    reg [3:0] data_in;
    wire [6:0] code_out;
    wire [3:0] data_out;
    wire error;

    // Instantiate modules
    HammingEncoder enc (.data_in(data_in), .code_out(code_out));
    HammingDecoder dec (.code_in(code_out), .data_out(data_out), .error(error));

    initial begin
        // Test Cases
        data_in = 4'b1011; #10;
        $display("Data: %b, Encoded: %b, Decoded: %b, Error: %b", data_in, code_out,
data_out, error);

        data_in = 4'b1100; #10;
        $display("Data: %b, Encoded: %b, Decoded: %b, Error: %b", data_in, code_out,
data_out, error);
```

```
        data_in = 4'b0110; #10;
        $display("Data: %b, Encoded: %b, Decoded: %b, Error: %b", data_in, code_out,
data_out, error);

        data_in = 4'b0001; #10;
        $display("Data: %b, Encoded: %b, Decoded: %b, Error: %b", data_in, code_out,
data_out, error);

        $stop;
    end
endmodule
```

📌 Applications

- Memory error correction (ECC RAM)
- Wireless communication
- Data transmission protocols

🎯 Conclusion

Hamming codes are essential for reliable data transmission and storage, ensuring data integrity in noisy environments.