Contents

1 Introduction

The Hamming Code is an error-detecting and error-correcting code used in telecommunications, computer systems, and data storage. It was developed in the late 1940s by Richard W. Hamming, an American mathematician and computer scientist, while working at Bell Labs. Hamming was frustrated with the frequent errors in data transmission and storage, particularly in early computer systems, which often required manual error correction. To address this, he devised a mathematical method that could automatically detect and correct errors, leading to the development of the Hamming Code. [1]

1.1 Background Information

The Hamming Code is based on binary linear error-correcting codes and is particularly known for its ability to detect and correct single-bit errors in transmitted data. It uses redundant bits, known as parity bits, to create a code that can check for errors and identify which bit is incorrect if an error is detected. The original Hamming Code (referred to as (7,4) Hamming Code) uses seven bits, where four are data bits, and three are parity bits.

The code operates using a method called Hamming Distance, which is the number of bit positions in which two code words differ. For error detection and correction, the Hamming Code relies on a minimum Hamming Distance of three between any two code words, allowing it to detect up to two-bit errors and correct one-bit errors.
Key Principles

1. **Parity Bit Calculation**:
   Parity bits are added at specific positions in the code word. Each parity bit covers a certain set of data bits and ensures that the number of 1s in that subset is even (even parity) or odd (odd parity).
2. **Error Detection and Correction**:
   When a code word is received, parity checks are performed. If the parity doesn't match, it indicates an error, and the position of the error can be determined using a binary index derived from the parity checks.

1.2 Current Trends and Application

While technology has evolved significantly since the Hamming Code's invention, it remains a foundational concept in digital error correction. Its simplicity and efficiency in correcting single-bit errors make it popular in applications where minimal redundancy and simplicity are key. Some modern applications and trends include:

1. **Low-Power, Embedded Systems**:
   : The Hamming Code is used in memory systems like static RAM (SRAM) in low-power devices, such as IoT devices and sensors, where power efficiency and low data redundancy are crucial.
2. **Network Communications**:
   Although advanced codes like LDPC (Low-Density Parity-Check) codes and Turbo Codes are more common in modern high-speed networks, the Hamming Code's simplicity still makes it useful for error detection in simpler protocols and small-scale networks.

2 Literature Survey

A 15-bit Hamming Code, such as (15,11) Hamming Code, was used to encode 11 data bits into 15 bits by adding four parity bits. This allows for single-bit error correction and double-bit error detection. Implementing the Hamming code encoding and correction between two communicating Micro controller.[2]

2.1 Mathematical principles

Theorem 1 (Hamming code)**.** *Hamming codes are a class of binary linear code. For each integer*

$$r \geq 2$$

there is a code-word with block length

$$n = 2 * r - 1$$

message length

$$k = 2 * r - r - 1$$

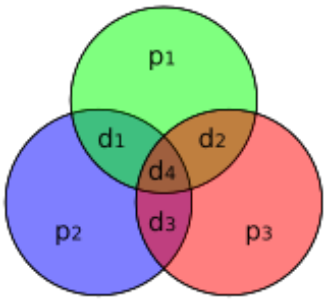


Figure 1: The Hamming(7,4) code (with r = 3)

2.2 Implementation Steps

- Software implementation

  Writing program to encode data into hamming code and decode the same. The program is ran and in the due process random bit flips in data is forced which is corrected using hamming code and hence making data Error Free.

- Implementation between two communicating Arduino

  Establishing Reliable Communication link between two Arduinos. of which one playing the role of a transmitter and

- Hardware Implementation

  Designing the the logic block diagram of hamming encoder on and Decoder and realizing the same using logic ICs and connecting them

3 OBJECTIVE-I : Understanding and Designing the Hamming Code Algorithm

3.1 Code Layout

- Encode 8-bit data into a 12-bit code word.
- Compute parity bits to ensure error detection and correction.
- Include error detection and single-bit error correction for the receiving Arduino.
- **Bit positions:**

The 12-bit Hamming code is structured as follows:

Table 1: Hamming (12,8) Code Layout

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Bit Type	D8	D7	D6	D5	P8	D4	D3	D2	P4	D1	P2	P1

D1 to D8 are the 8 data bits.
P1, P2, P4, and P8 are the 4 parity bits.

3.2 Designing the Hamming Code

- Step 1: Determine the number of parity bits. Given a data bit sequence of length m, the number of parity bits r is determined such that:

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

- Step 2: Position the parity bits Place the parity bits at positions that are powers of 2 (i.e., positions 1, 2, 4, 8, etc.). The remaining positions are reserved for data bits.
- Step 3: Calculate parity bit values, The value of each parity bit is calculated based on specific bits in the data sequence. For each parity bit, cover bits are determined using binary representation. For example, parity bit P_1 covers bits at positions where the least significant bit of their binary representation is 1.

  Example Consider a 4-bit data sequence: `1011`.

  1. Calculate the number of parity bits needed:

$$2^r \geq 4 + r + 1 \Longrightarrow r = 3 \tag{2}$$

  2. Insert parity bits at positions 1, 2, and 4: `P1, P2, 1, P4, 0, 1, 1`.
  3. Calculate values for each parity bit:
     - P_1 covers positions 1, 3, 5, 7.
     - P_2 covers positions 2, 3, 6, 7.
     - P_4 covers positions 4, 5, 6, 7.

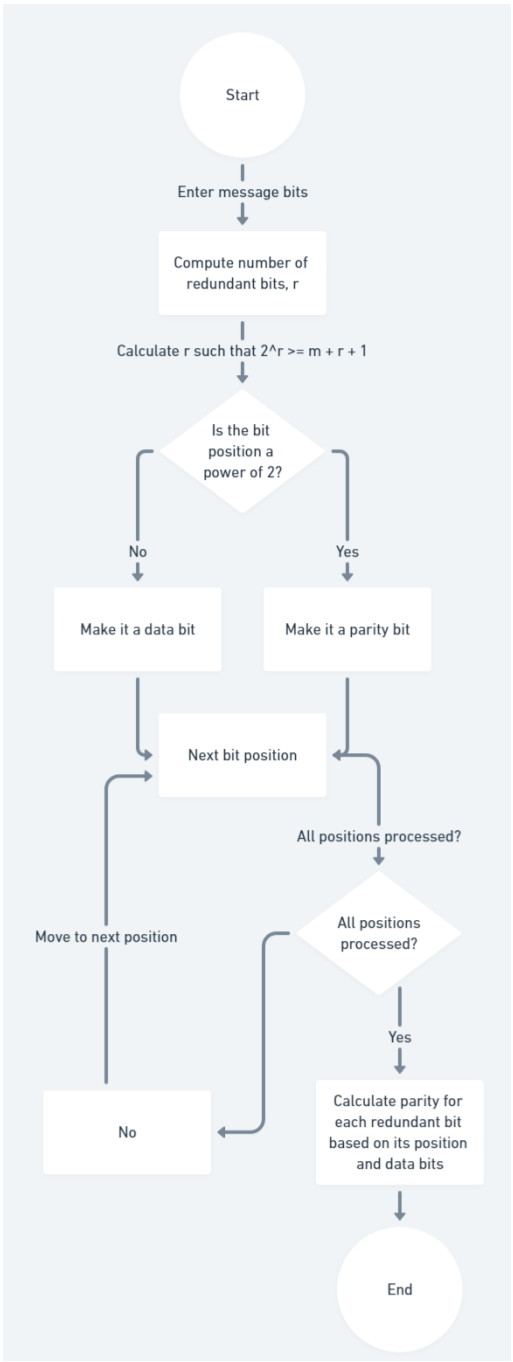
  A (12,8) Hamming code uses 8 data bits and 4 parity bits to form a 12-bit code word. The parity bits are strategically placed in positions that are powers of two (1, 2, 4, and 8), while the data bits are placed in the remaining positions.[3]

4 OBJECTIVE-II : Implementing on software

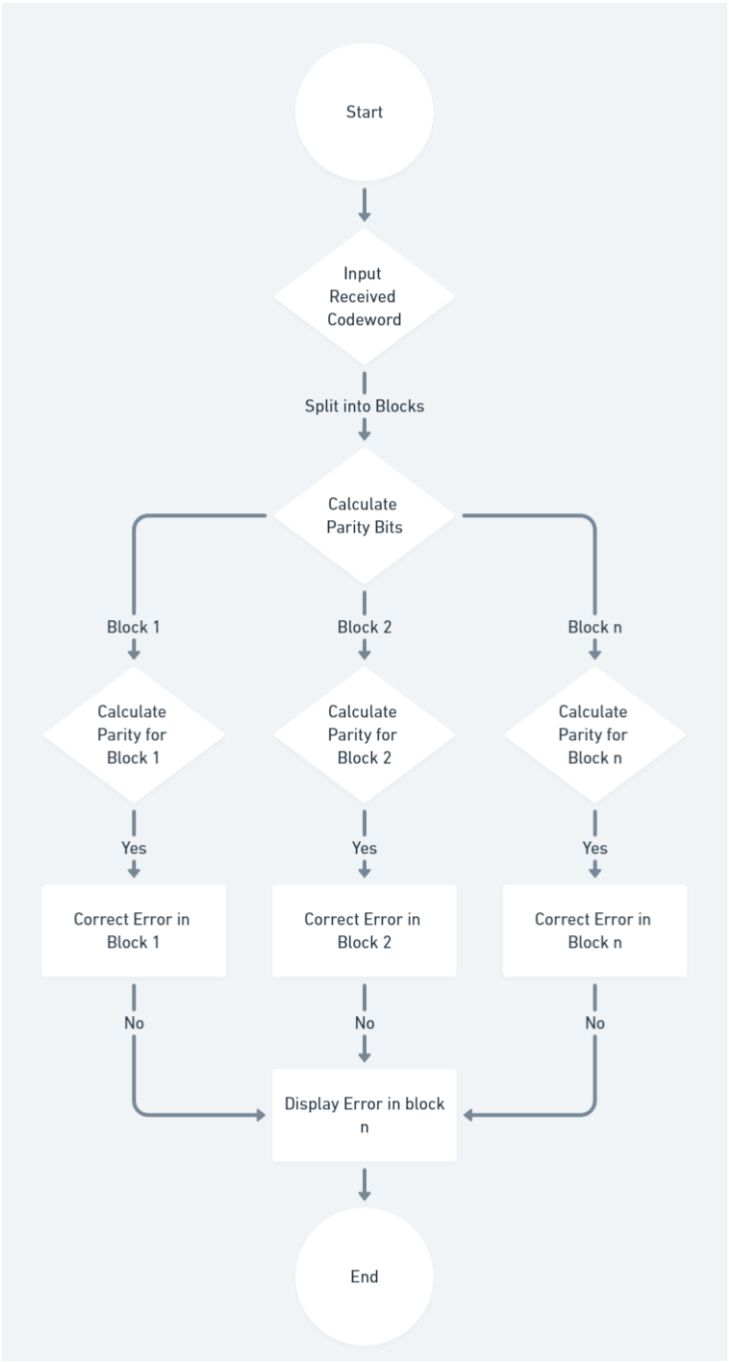
Arduino Sketch for Hamming Code

Below is a simple Arduino sketch to implement a Hamming code encoding and decoding algorithm:[4]

Encoding Algorithm



Decoding Algorithm



5 OBJECTIVE-III : Implementing on Microprocessor

Implement the Hamming (12,8) code encoding and decoding on a microprocessor, such as an Arduino. This implementation will allow real-time error detection and correction in data transmitted between devices.

1. Setting Up the Microprocessor
    - Use an Arduino (or similar micro controller) to send and receive data.
    - Prepare two Arduino boards if you want to simulate the transmission of encoded data between devices.
    - Connect the Arduinos via Serial or I2C communication, which allows them to send and receive data reliably.
2. Programming the Encoder and Decoder
    - Write functions to encode 8-bit data into a 12-bit Hamming code.
    - Write functions to decode the 12-bit Hamming code back into 8-bit data and correct single-bit errors if necessary.
    - Store these functions in header files (`HammingEncoder.h` and `HammingDecoder.h`) for modularity and re-usability.
3. Integrating the Hamming Encoder and Decoder Code
    - Import the encoder and decoder header files into your Arduino sketch.
    - Designate one Arduino as the **transmitter** (encoding and sending data) and the other as the **receiver** (decoding and error-checking the received data).

Tools for Hamming Code Implementation on an Arduino

Tool	**Description**
Arduino Board	The microcontroller board ie, Arduino nano using ATmega32P used for coding and testing the Hamming Code implementation.
Arduino IDE	Software used to write, compile, and upload code to the Arduino board. It provides an environment for developing Hamming code logic in C/C++.
Breadboard	Used for building and testing the circuit without soldering, connecting the Arduino pins with peripherals like LEDs and switches.
Jumper Wires	Used to make connections between the Arduino, breadboard, and other components in the circuit.
LEDs	Can be used as output indicators for testing Hamming code output such as error detection/correction results.
Resistors	Typically used with LEDs to limit current and protect components during testing and debugging.
Oscilloscope	A device used to observe and analyze the electrical signals in the circuit, which is helpful for debugging timing and signal integrity issues.
PS/2 Keyboard	A peripheral device that can be used for input testing and interfacing with the Arduino to send data streams that may require Hamming code error checking.

6 Methodology

6.1 Thought Train Mapping

Figure 2: Methodology Mindmap

6.2 System design Mapping

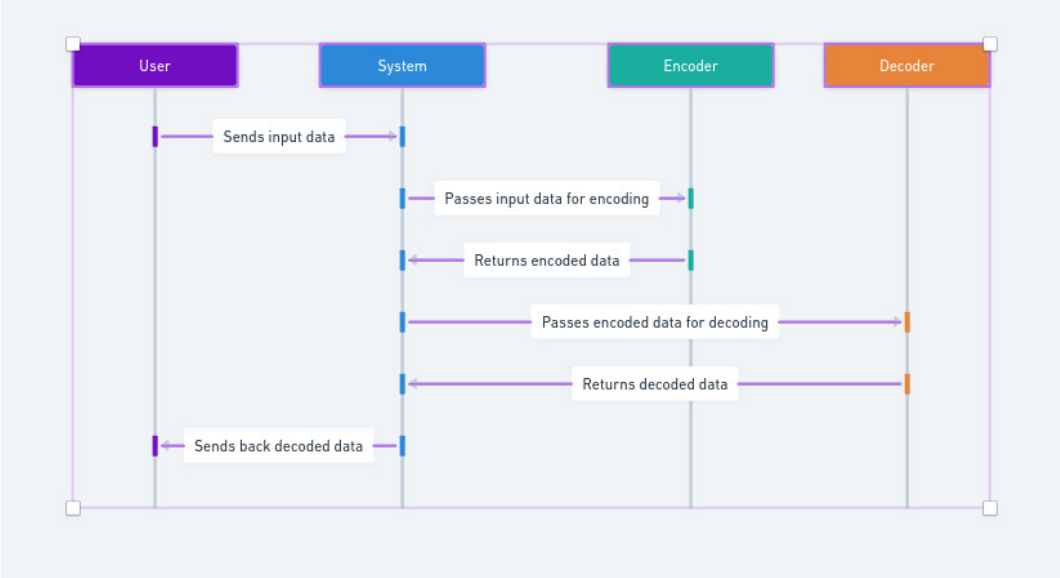


Figure 3: Algorithim Sequence flow

7 Conclusions and Scope for Future Works

Key findings of Hamming code implementation include:

- **Error Detection and Correction Capability:** Hamming codes detect and correct single-bit errors, which enhances communication system reliability.
- **Low Complexity:** The encoding and decoding processes are computationally lightweight, suitable for real-time applications.
- **Hardware Efficiency:** Hamming codes can be effectively implemented in both software and hardware, including FPGA and ASIC designs.

Scope for Future Work

While Hamming codes provide a strong foundation for error detection and correction, further enhancements can extend their capabilities for evolving applications. Future work can focus on the following aspects:

- Hardware Optimization

  Optimizing the hardware implementation of Hamming code, including faster encoding and decoding circuits, can benefit real-time communication and memory systems. This involves reducing power consumption, increasing speed, and enhancing scalability.

- Integration with Modern Protocols

  As communication protocols continue to evolve, integrating Hamming codes into modern network and communication standards, including wireless networks, 5G systems, and IoT applications.

- Quantum Error Correction

  Exploring the application of Hamming code principles in quantum computing environments, where quantum error correction is essential, can open new research directions and lead to novel hybrid quantum-classical solutions.

In conclusion, Hamming code remains a fundamental approach for error detection and correction. Continued research and development can expand its applications, making it more relevant for emerging technologies and critical communication systems.

References

[1] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[2] I. N. John, P. W. Kamaku, D. K. Macharia, and N. M. Mutua, "Error detection and correction using hamming and cyclic codes in a communication channel," *Pure and Applied Mathematics Journal*, vol. 5, no. 6, pp. 220–231, 2017. [Online]. Available: https://doi.org/10.11648/j.pamj.20160506.17

[3] O. Khadir, "A simple coding-decoding algorithm for the hamming code," 01 2022.

[4] Beneater, "Error detection video series," https://github.com/beneater/error-detection-videos, 2018, accessed: 2024-11-07.