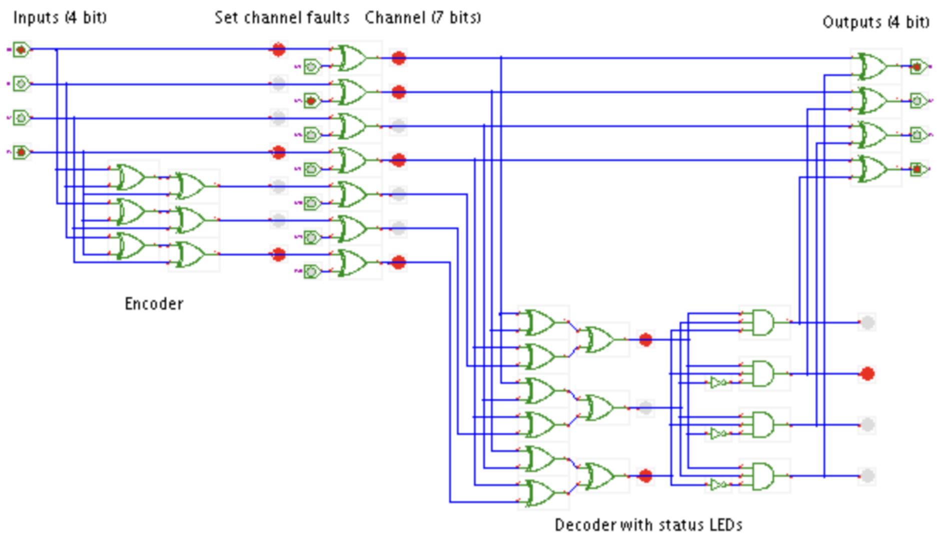
TEXT CORRECTION WITH HAMMING CODES

Authors: Neha Pedgaonkar, Yashowardhan Rai, Disha Zaveri.

Abstract: In this project, we explore the practical applications of error detection and correction codes, specifically focusing on the (7,4) Hamming code. Initially, we construct a hardware circuit to implement the (7,4) Hamming code, demonstrating via LEDs its capability to detect and correct errors in transmitted data. Subsequently, we extend our exploration by integrating the Hamming code within an Arduino-based system for text correction. By sending text data to the Arduino and then the digital circuit in groups of 4 bits, we introduce errors in transmission via bit flips, corrupting the text. Leveraging the Hamming code, the digital encoder corrects the corrupted text data, subsequently sending it to be displayed on an LCD screen. By combining hardware implementation with practical demonstrations, this project provides a comprehensive understanding of error detection and correction codes, showcasing their relevance in real-world applications.

Introduction: In today's interconnected world, where digital data flows incessantly across various communication channels, the integrity of transmitted information stands as a cornerstone of reliability. Motivated by the ever-present need for accurate data transmission, this project embarks on an exploration of error detection and correction codes. Rooted in a passion for electronics and a desire to understand the inner workings of these codes, the project sets out to construct a (7,4) Hamming code circuit. This circuit serves as a tangible representation of theoretical concepts, demonstrating the efficacy of error correction in detecting and rectifying transmission errors. Furthermore, by extending this exploration to practical applications, such as integrating error correction mechanisms into an Arduino-based system for text correction, the project aims to underscore the vital role of error correction codes in ensuring data integrity in real-world scenarios. Through this endeavour, the project seeks to not only deepen understanding but also highlight the transformative potential of error correction technologies in modern communication systems.

Design concept and implementation:

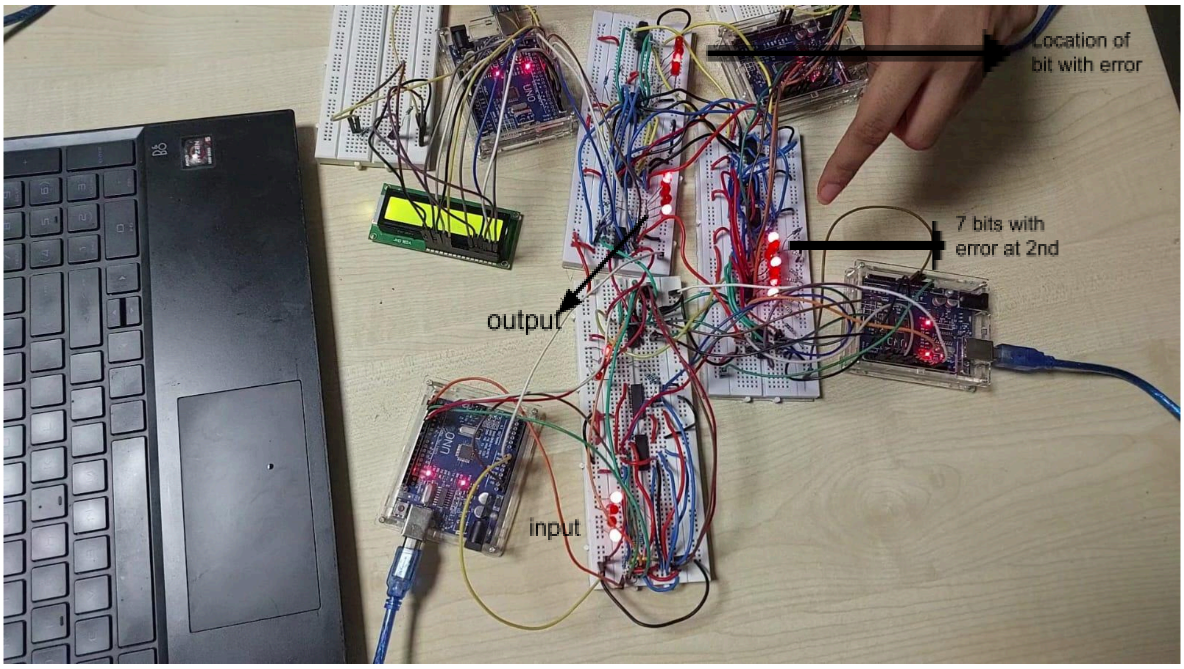


Input will be sent from the arduino that will send the image information as groups of 4 bits. Error is introduced by an arduino, both constant and random. The first 4 bits which potentially have an error as well as the final output is displayed on an LCD screen via an Arduino.

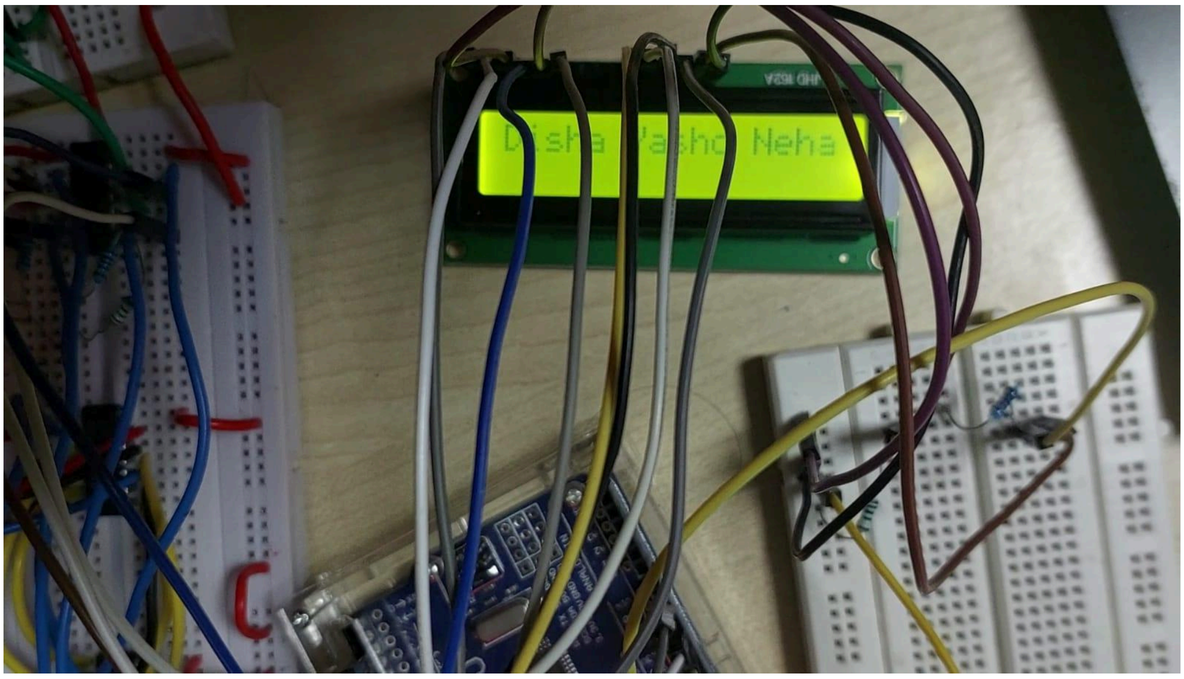
List of components used: XOR, NOT, AND gates, LEDs, Arduino, JHC 162A LCD Display, Computer.

Results:

The successful operation of the digital implementation of the (7, 4) Hamming code is evidenced by the simultaneous illumination of input and output LEDs, providing a visual confirmation of error detection and correction. Moreover, the correlation between error-indicating LEDs and the output displayed on the serial monitor reaffirms the code's accuracy in identifying and rectifying errors introduced during data transmission.



In addition to the visual cues, the accuracy of the error correction process is further validated by the correct display of the input text on the final output screen. This not only underscores the efficacy of the (7, 4) Hamming code in ensuring data integrity but also highlights its practical relevance in real-world applications.



However, amidst these achievements, a notable challenge arises when the Arduino is not powered on simultaneously. In such instances, discrepancies in the output text become apparent, reflecting delays in the processing of input data. This is seen for both constant and random errors.

Discussion: The difference in output text when the arduino is not simultaneously switched on is because the arduino with the code for displaying text is delayed in taking in the bits, and hence takes in the wrong string of bits. We tackled this by resetting the arduino after connecting them to our laptops. Implementing a solution to address the issue of Arduino synchronisation, such as utilising an Arduino Mega with sufficient input pins and developing concurrent-running code, would enhance the reliability of the system. This particular code could also be expanded to image correction by converting pixels to bits and using a JH4 20x4 LCD display.

Summary: In summary, the research project successfully explores the practical applications of error detection and correction codes, with a focus on the (7,4) Hamming code. Through the construction of a hardware circuit and its integration into an Arduino-based system, the project demonstrates the effectiveness of Hamming codes in detecting and correcting errors in transmitted data. Despite encountering challenges related to Arduino synchronisation, the project provides valuable insights into the implementation and optimization of error correction mechanisms in real-world scenarios.

Resources:

https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/50-hamming/hamming.html#:~:text=Hamming%20code%20demonstration&text=This%20circuit%20demonstrates%20the%20use,with%20the%20four%20data%20bits ,
3 Blue 1 Brown videos on Hamming Error Correction
Image Encoding with Error Correction Ability Using Hamming Code, Ng Kyle - 135200401
Interfacing LCD to Arduino-Tutorial to Display on LCD Screen (circuitstoday.com)
https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders#:~:text=Like%20BCH%20codes%2C%20Reed%E2%80%93Solomon,append%20to%20the%20original%20message

Code:

Text Input Code

```
void setup() {
  // Set digital pins 2, 3, 4, 5 as outputs
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
}
void loop() {
    // Convert text to ASCII
  char text[] = "Disha Yasho Neha";
  int ascii_values[strlen(text)];
  for (int i = 0; i < strlen(text); i++) {
    ascii_values[i] = int(text[i]);
  }

  // Convert ASCII to binary and send in 4-bit pieces
  for (int i = 0; i < sizeof(ascii_values) / sizeof(ascii_values[0]);
i++) {
    char binary_str[9]; // 8 bits + '\0'
```

```
      itoa(ascii_values[i], binary_str, 2);
      // Pad binary string with zeros to ensure it's 8 bits
      while (strlen(binary_str) < 8) {
        memmove(binary_str + 1, binary_str, strlen(binary_str) + 1);
        binary_str[0] = '0';
      }
      // Send in 4-bit pieces
      for (int j = 0; j < 8; j += 4) {
        send_binary(binary_str + j);
      }
    }

}
// Function to send 4-bit binary data to digital pins
void send_binary(char* data) {
  digitalWrite(2, data[0] == '1' ? HIGH : LOW);
  digitalWrite(3, data[1] == '1' ? HIGH : LOW);
  digitalWrite(4, data[2] == '1' ? HIGH : LOW);
  digitalWrite(5, data[3] == '1' ? HIGH : LOW);
  delay(500); // Adjust delay as needed
}
```

Constant Error Code

```
void setup() {
  // put your setup code here, to run once:
  DDRD = B11111100;
  pinMode(8, OUTPUT);
  Serial.begin(9600); // Initialize serial communication
  PORTD=B00001000;
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Random Error Code

```
void setup() {
  // put your setup code here, to run once:
  DDRD = B11111100;
  pinMode(8, OUTPUT);
  Serial.begin(9600); // Initialize serial communication
}
```

```
void loop() {
  // put your main code here, to run repeatedly:
  PORTD = B00000000;
  digitalWrite(8, LOW);
  int input = random(2,6);
  Serial.println(input-1);
  digitalWrite(input, HIGH);
  delay(2000);
}
```
LCD Display Code
```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // sets the interfacing pins

void setup() {
  lcd.begin(16, 2);  // initializes the 16x2 LCD
  // Set digital pins 6-9 as inputs for receiving 4-bit data
  pinMode(6, INPUT);
  pinMode(7, INPUT);
  pinMode(8, INPUT);
  pinMode(9, INPUT);

  // Initialize serial communication
  Serial.begin(9600);
}

void loop() {
  lcd.clear();
  for (int j=0;j<16;j++)
  {
  // Read 4-bit inputs and convert them to a string
  String bits = "";
  for (int i = 6; i <= 9; i++) {
    bits += digitalRead(i);
    Serial.println(bits);
  }
  delay(500);
  for (int i = 6; i <= 9; i++) {
    bits += digitalRead(i);
    Serial.println(bits);
  }
  // Convert the binary string to ASCII characters (assuming 8-bit
ASCII)
```

```
  String asciiChars = "";
  for (int i = 0; i < 8; i++) {
    int startIndex = i * 8;
    String subStr = bits.substring(startIndex, startIndex + 8);
    char asciiChar = strtol(subStr.c_str(), NULL, 2);
    asciiChars += asciiChar;
  }

  // Print the ASCII characters to the serial output
  Serial.println(asciiChars);

  // Display the ASCII characters on the LCD screen
  lcd.setCursor(j,0);
  lcd.print(asciiChars);
  // Delay for a short interval
  delay(500);
  }
}
```