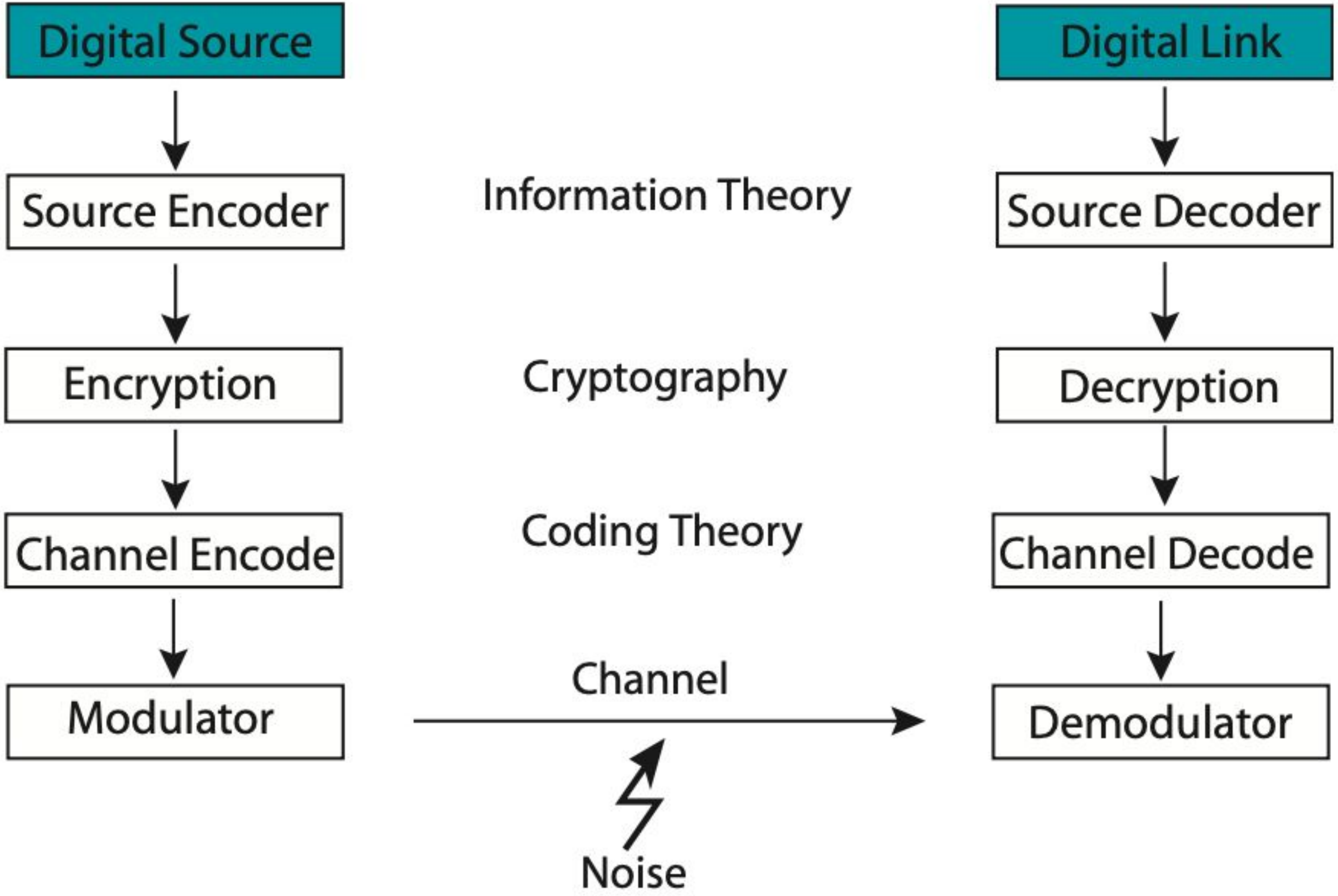
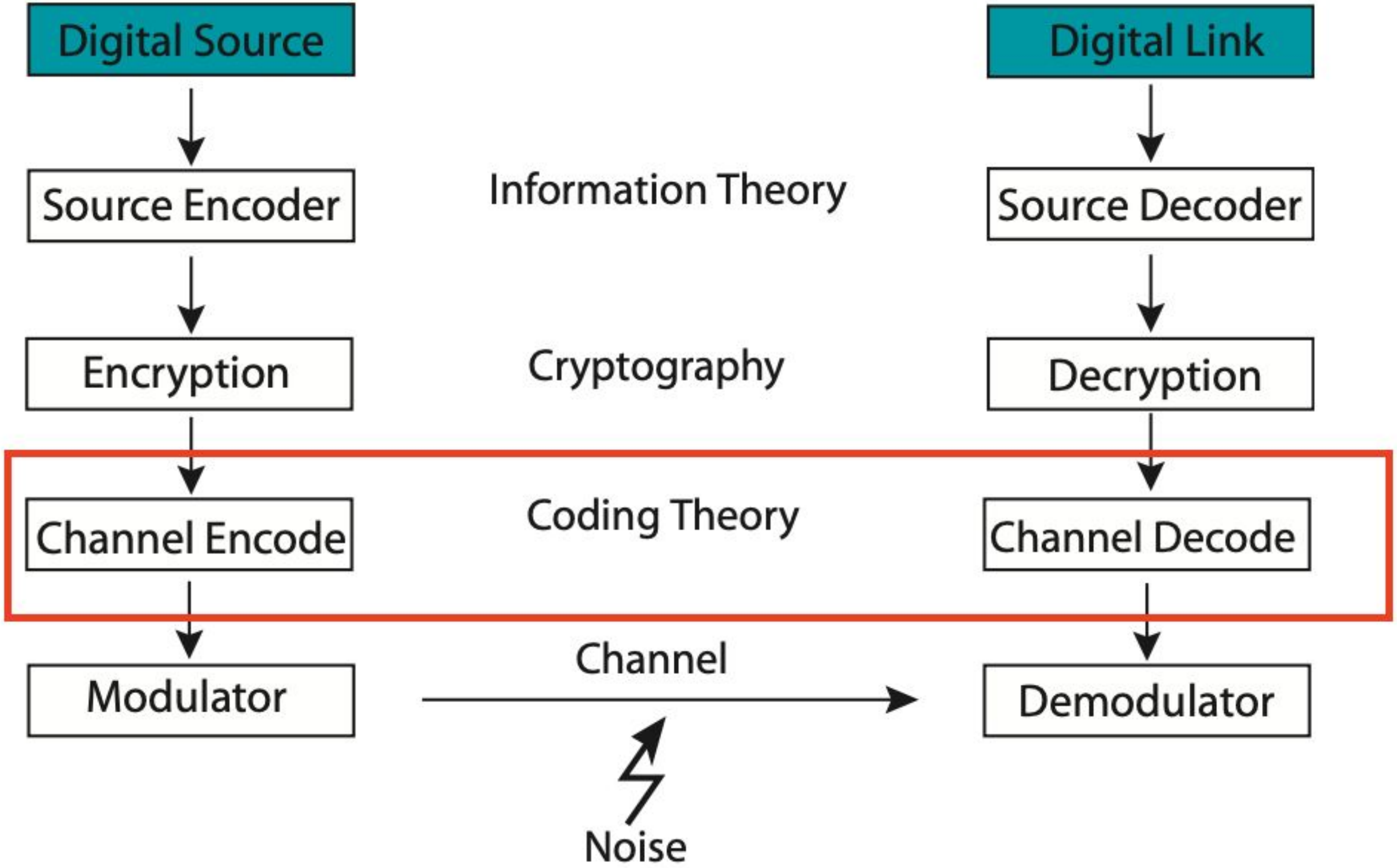
Hamming Codes: A Hardware Implementation

Tyler Ewald, Zayn Patel, Manu de Tezanos Pinto, Tane Koh


Digital Source
Source Encoder
Encryption
Channel Encode
Modulator
Information Theory
Cryptography
Coding Theory
Channel
Noise
Digital Link
Source Decoder
Decryption
Channel Decode
Demodulator


Digital Source
Digital Link
Source Encoder
Information Theory
Source Decoder
Encryption
Cryptography
Decryption
Channel Encode
Coding Theory
Channel Decode
Modulator
Channel
Demodulator
Noise

What is a Hamming Code?

- Length 4 messages to length 7 codewords
  - Codewords are n and messages are k

What is a Hamming Code?

- Length 4 messages to length 7 codewords
- Single error correcting

Hamming Codes from Multiple Perspectives

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 \end{pmatrix}$$

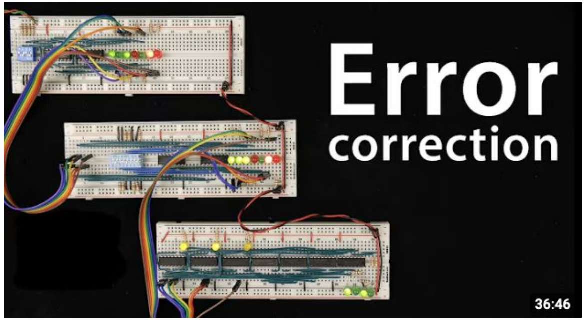
$$\downarrow$$

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 & p_1 & p_2 & p_3 \end{pmatrix}$$

Message to Codeword

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Generator Matrix



Hardware

Encoding 1100 with mod 2

1100 $\longrightarrow$ 1100 _ _ _

Encoding 1100 with mod 2

$$1100 \longrightarrow 1100\ \_\ \_\ \_$$

p_1

Encoding 1100 with mod 2

$$\underset{m_1}{1}\,\underset{m_2}{1}\,\underset{m_3}{0}\,\underset{m_4}{0} \longrightarrow 1100\ \underset{p_1}{\_\_}\ \_\_\ \_\_$$

Encoding 1100 with mod 2

1100 $\longrightarrow$ 1100 _ _ _

m_1 m_2 m_3 m_4

p_1

To get p_1 we will use: $m_1 + m_2 + m_4 \bmod 2$

Encoding 1100 with mod 2

$$\underset{m_1\,m_2\,m_3\,m_4}{1100} \longrightarrow 1100\ \underset{p_1}{\_}\ \_\ \_$$

To get p_1 we will use: $m_1 + m_2 + m_4 \bmod 2$

$1 + 1 + 0 \bmod 2 = 0$

Encoding 1100 with mod 2

$$1100 \longrightarrow 1100\ 0\ \_\ \_$$

m_1 m_2 m_3 m_4

p_1 p_2

To get p_2 we will use: $m_1 + m_3 + m_4 \bmod 2$

Encoding 1100 with mod 2

1100 → 1100 0 _ _

m_1 m_2 m_3 m_4

p_1 p_2

To get p_2 we will use: $m_1 + m_3 + m_4 \bmod 2$

$1 + 0 + 0 \bmod 2 = 1$

Encoding 1100 with mod 2

$$\underset{m_1\,m_2\,m_3\,m_4}{1100} \longrightarrow 1100\ \underset{p_1}{0}\ \underset{p_2}{1}\ \underset{p_3}{\_}$$

To get p_3 we will use: $m_2 + m_3 + m_4 \bmod 2$

Encoding 1100 with mod 2

1100 $\longrightarrow$ 1100 0 1 _

m_1 m_2 m_3 m_4

p_1 p_2 p_3

To get p_3 we will use: $m_2 + m_3 + m_4 \bmod 2$

$1 + 0 + 0 \bmod 2 = 1$

Final Encoding of 1100 with mod 2

$$1100 \longrightarrow 1100\ 0\ 1\ 1$$

m_1 m_2 m_3 m_4 p_1 p_2 p_3

We can use the general (n, k) equations to verify what we learned earlier:

- $k \leq 2^r - 1 - r$ with r parity bits
- $n \geq 2^r - 1$ with r parity bits

XOR operator is the same as mod 2

Parity bit equations with mod 2

$p_1 = m_1 + m_2 + m_4 \mod 2$

$p_2 = m_1 + m_3 + m_4 \mod 2$

$p_3 = m_2 + m_3 + m_4 \mod 2$

XOR operator is the same as mod 2

Parity bit equations with mod 2

$p_1 = m_1 + m_2 + m_4 \bmod 2$

$p_2 = m_1 + m_3 + m_4 \bmod 2$

$p_3 = m_2 + m_3 + m_4 \bmod 2$

Parity bit equations with XOR

$p_1 = m_1 \oplus m_2 \oplus m_4$

$p_2 = m_1 \oplus m_3 \oplus m_4$

$p_3 = m_2 \oplus m_3 \oplus m_4$

Hamming Code Efficiency

In general we say the efficiency of a code is:

$$\frac{k \text{ (message length)}}{n \text{ (codeword)}}$$

Hamming Code Efficiency

In general we say the efficiency of a code is: $\dfrac{\text{k (message length)}}{\text{n (codeword)}}$

Hamming (7,4)	$Rate = \frac{4}{7} = 57.1\%$
Hamming (15,11)	$Rate = \frac{11}{15} = 73.3\%$
Hamming (31,26)	$Rate = \frac{26}{31} = 83.9\%$

High rate means more efficiency of redundancy bits.
But, only one error can be corrected.

Hamming Codes from Multiple Perspectives

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 \end{pmatrix}$$

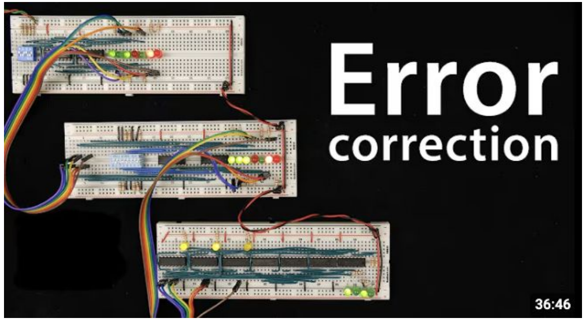
$$\downarrow$$

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 & p_1 & p_2 & p_3 \end{pmatrix}$$

Message to Codeword

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Generator Matrix



Hardware

Facts about Generator Matrices

- 1x4 matrix message
- 4x7 generator matrix
  - 1x7 codeword
- Matrix multiplication is faster and more efficient than manual adding of parity bits
- 1-1 and onto mapping from message to codeword
- Easy to store information

Encoding 1100 using Generator Matrix

These are the parity bits. They follow the same equations from earlier.

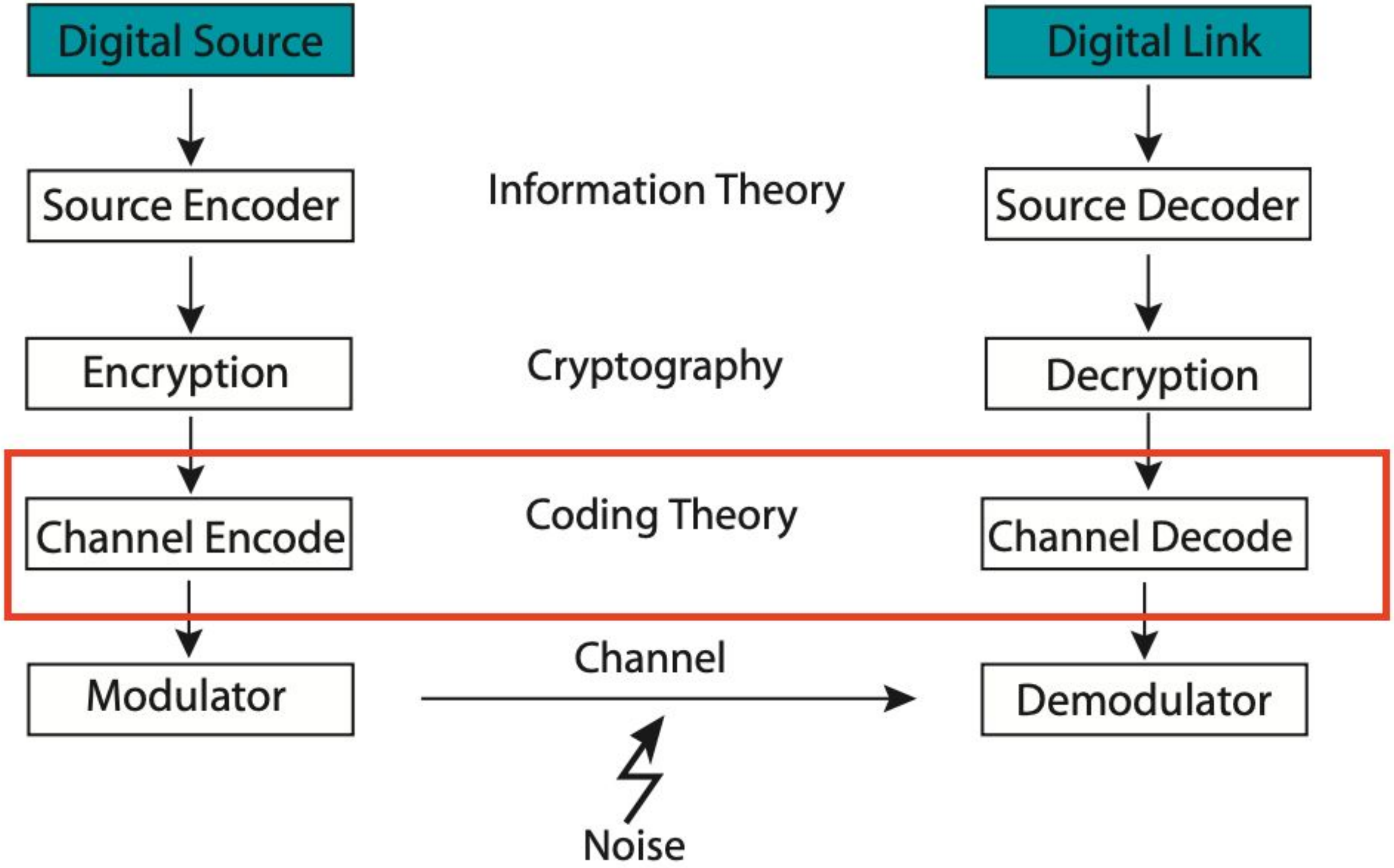
$$(1100)\begin{pmatrix} \boxed{\begin{matrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{matrix}} & \boxed{\begin{matrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 1\end{matrix}} \end{pmatrix} = (\boxed{1100}\boxed{\_\ \_\ \_})$$

$$
\left(
\begin{array}{ccccccc}
1 \times 1 & 0 \times 1 & 0 \times 1 & 0 \times 1 & 1 \times 1 & 1 \times 1 & 0 \times 1 \\
0 \times 1 & 1 \times 1 & 0 \times 1 & 0 \times 1 & 1 \times 1 & 0 \times 1 & 1 \times 1 \\
0 \times 0 & 0 \times 0 & 1 \times 0 & 0 \times 0 & 0 \times 0 & 1 \times 0 & 1 \times 0 \\
+\,0 \times 0 & +\,0 \times 0 & +\,0 \times 0 & +\,1 \times 0 & +\,1 \times 0 & +\,1 \times 0 & +\,1 \times 0 \\
\hline
\end{array}
\right)
$$

$$
(\;1 \quad 1 \quad 0 \quad 0 \quad 2 \quad 1 \quad 1\;)
$$

$$
\mathrm{mod}2
$$

$$
(1100011)
$$


Digital Source
Digital Link
Source Encoder
Information Theory
Source Decoder
Encryption
Cryptography
Decryption
Channel Encode
Coding Theory
Channel Decode
Channel
Modulator
Demodulator
Noise

Facts about Parity Check Matrices

- 3x7 parity check matrix
- 7x1 codeword
  - 3x1 error (“syndrome”)
- Identifies the error bit *very* easily
- Easy to store information

Facts about the Syndrome Vector

- Output of parity check matrix multiplied by codeword, transposed
- If there is no error the syndrome is a 3x1 zero vector
- If there is an error the syndrome is a 3x1 vector identical to a column in parity check matrix

Decoding with the Parity Check Matrix (General Form)

$$\mathbf{H} \qquad \mathbf{c}^{\mathsf{T}}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

Decoding with the Parity Check Matrix (Zero Errors)

$$\underset{\mathbf{H}}{\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}} \quad \underset{}{\overset{\mathbf{c^T}}{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

The syndrome identifies no error column so we received the correct codeword.

Decoding with the Parity Check Matrix (One Error)

$$\mathbf{H} \qquad \mathbf{c}^{\mathbf{T}}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Decoding with the Parity Check Matrix (One Error)

$$\mathbf{H}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{c}^{\mathsf{T}}$$

$$\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Channel Noise

Decoding with the Parity Check Matrix (One Error)

$$\mathbf{H} \qquad \mathbf{c}^{\mathbf{T}}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Bit **flipped**.

Decoding with the Parity Check Matrix (One Error)

$$\underbrace{\begin{pmatrix} 1 & \boxed{1} & 0 & 1 & 1 & 0 & 0 \\ 1 & \boxed{0} & 1 & 1 & 0 & 1 & 0 \\ 0 & \boxed{1} & 1 & 1 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{H}} \overbrace{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}}^{\mathbf{c}^T} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

The syndrome identifies the error column (and index of bit string) and we flip this bit.

Hamming Codes from Multiple Perspectives

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 \end{pmatrix}$$

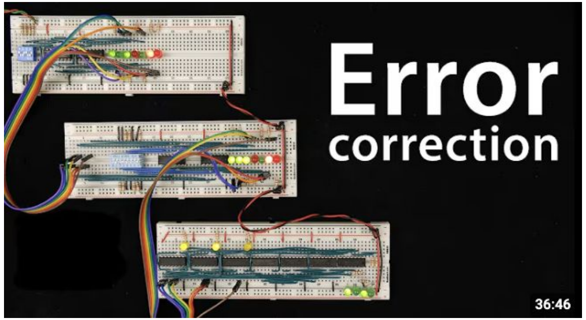
$$\downarrow$$

$$\begin{pmatrix} m_1 & m_2 & m_3 & m_4 & p_1 & p_2 & p_3 \end{pmatrix}$$

Message to Codeword

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Generator Matrix



Hardware

Arduino Encoding

```
void ProcessMessage(String message)
{
  // Process command only if it is 4 bits long
  if (message.length() == 4)
  {
    // Show user what message they are sending
    Serial.print("Sending message:  ");
    Serial.println(message);

    // Encode the message to local variables
    String bit_1 = message.substring(0, 1);
    String bit_2 = message.substring(1, 2);
    String bit_3 = message.substring(2, 3);
    String bit_4 = message.substring(3, 4);

    //
    if (bit_1 == "1")
    {
      digitalWrite(og_bit_1, HIGH);
    }
    else if (bit_1 == "0")
    {
      digitalWrite(og_bit_1, LOW);
    }
```

```
if (bit_2 == "1")
{
  digitalWrite(og_bit_2, HIGH);
}
else
{
  digitalWrite(og_bit_2, LOW);
}

if (bit_3 == "1")
{
  digitalWrite(og_bit_3, HIGH);
}
else
{
  digitalWrite(og_bit_3, LOW);
}

if (bit_4 == "1")
{
  digitalWrite(og_bit_4, HIGH);
}
else
{
  digitalWrite(og_bit_4, LOW);
}
```

Output   Serial Monitor ×

Message (Enter to send message to 'Arduino UNO R4 Minima' on 'COM8')

```
Received: 11111111111111
Invalid code.
Received: 1101
Sending message:  1101
Message Sent
Received: 1111
Sending message:  1111
Message Sent
Received: 0000
Sending message:  0000
Message Sent
```

Parity Bit Generation

Mathematical Approach with mod 2

$$p_1 = m_1 + m_2 + m_4 \bmod 2$$

$$p_2 = m_1 + m_3 + m_4 \bmod 2$$

$$p_3 = m_2 + m_3 + m_4 \bmod 2$$

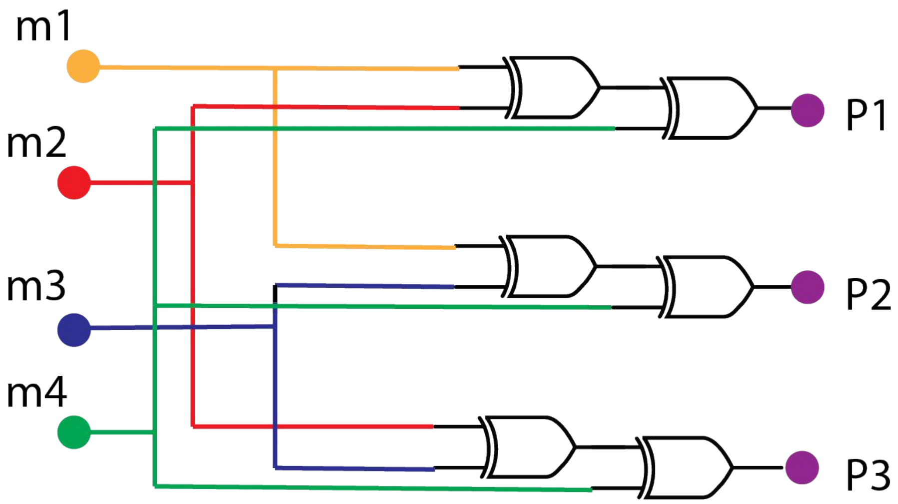
Parity bit equations with XOR

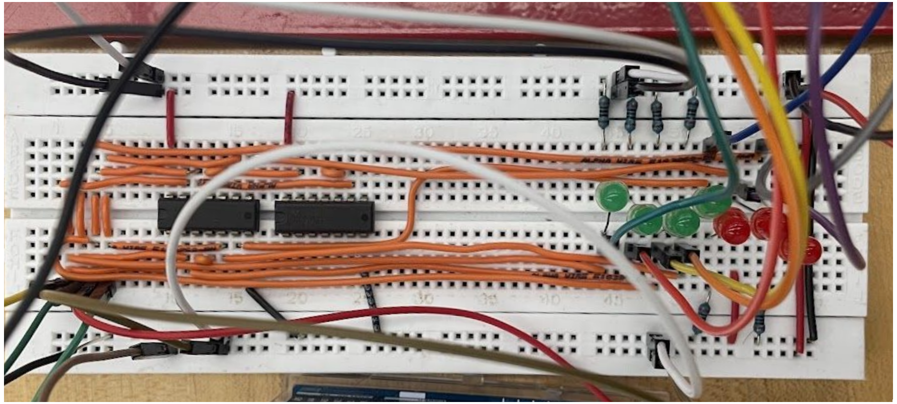
$$p_1 = m_1 \oplus m_2 \oplus m_4$$

$$p_2 = m_1 \oplus m_3 \oplus m_4$$

$$p_3 = m_2 \oplus m_3 \oplus m_4$$

Formation using gates





Error Detection

Linear Algebra Approach

$$\mathbf{H} \quad \color{red}{\mathbf{c}^T}$$

$$\begin{pmatrix} 1 & \boxed{1} & 0 & 1 & 1 & 0 & 0 \\ 1 & \boxed{0} & 1 & 1 & 0 & 1 & 0 \\ 0 & \boxed{1} & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Hardware Approach

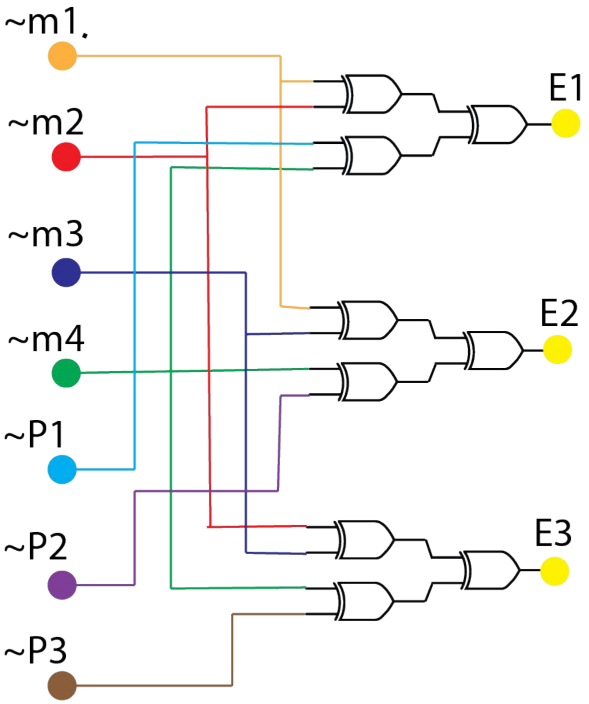
Parity bit equations with XOR

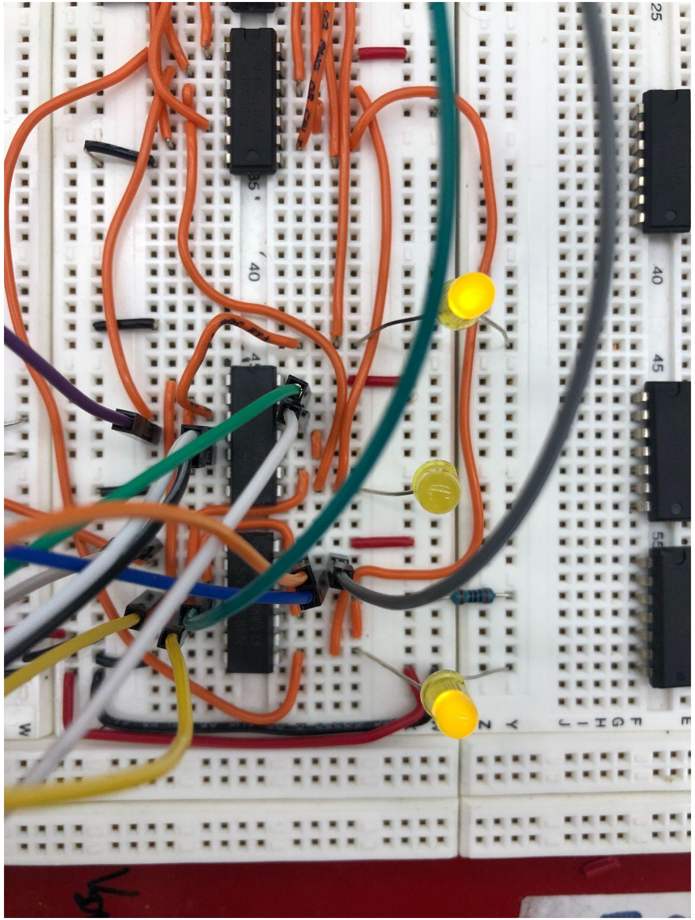
$$E_1 = (\sim m_1 \oplus \sim m_2) \oplus (\sim m_4 \oplus \sim p_1)$$

$$E_2 = (\sim m_1 \oplus \sim m_3) \oplus (\sim m_4 \oplus \sim p_2)$$

$$E_3 = (\sim m_2 \oplus \sim m_3) \oplus (\sim m_4 \oplus \sim p_3)$$

Detection using gates





Error Correction

Linear Algebra Approach

$$\mathbf{c}^T$$

$$\underset{\mathbf{H}}{\begin{pmatrix} 1 & \boxed{1} & 0 & 1 & 1 & 0 & 0 \\ 1 & \boxed{0} & 1 & 1 & 0 & 1 & 0 \\ 0 & \boxed{1} & 1 & 1 & 0 & 0 & 1 \end{pmatrix}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Hardware Approach

Logic Equations

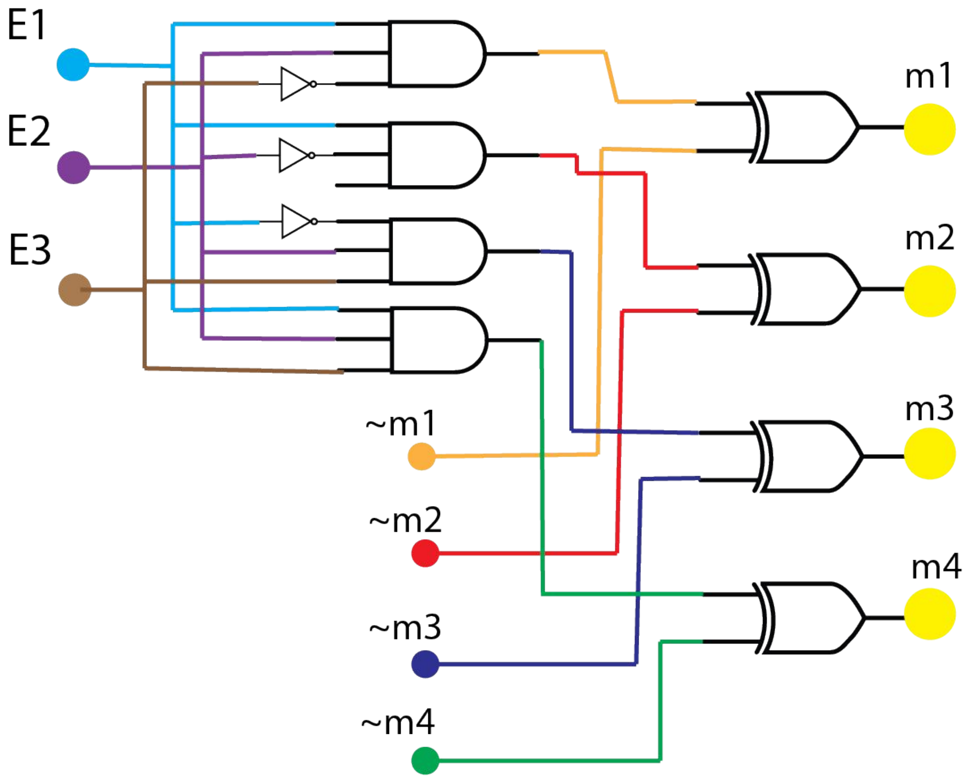
$$m_1 = (E_1 \text{ x } E_2 \text{ x } \bar{E}_3) \oplus \sim m_1$$

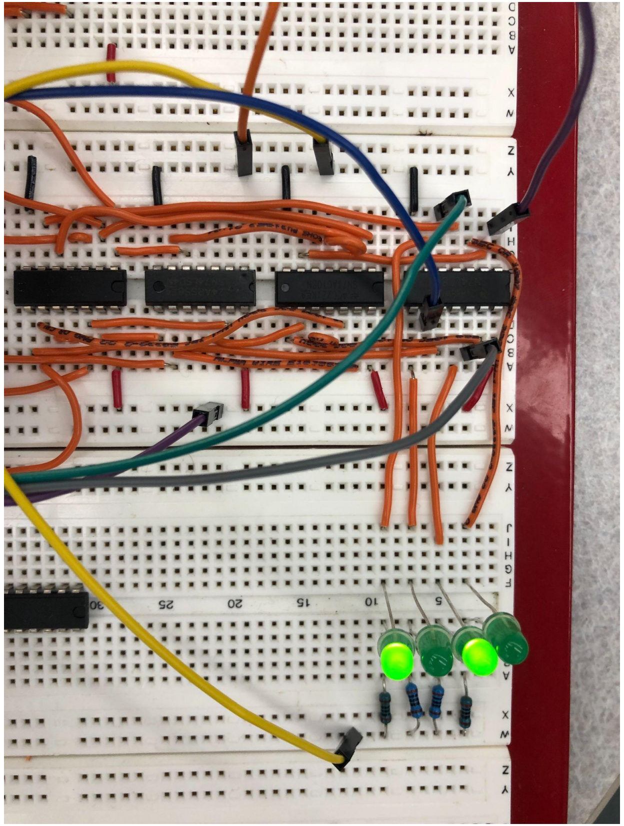
$$\boxed{m_2 = (E_1 \text{ x } \bar{E}_2 \text{ x } E_3) \oplus \sim m_2}$$

$$m_3 = (\bar{E}_1 \text{ x } E_2 \text{ x } E_3) \oplus \sim m_3$$

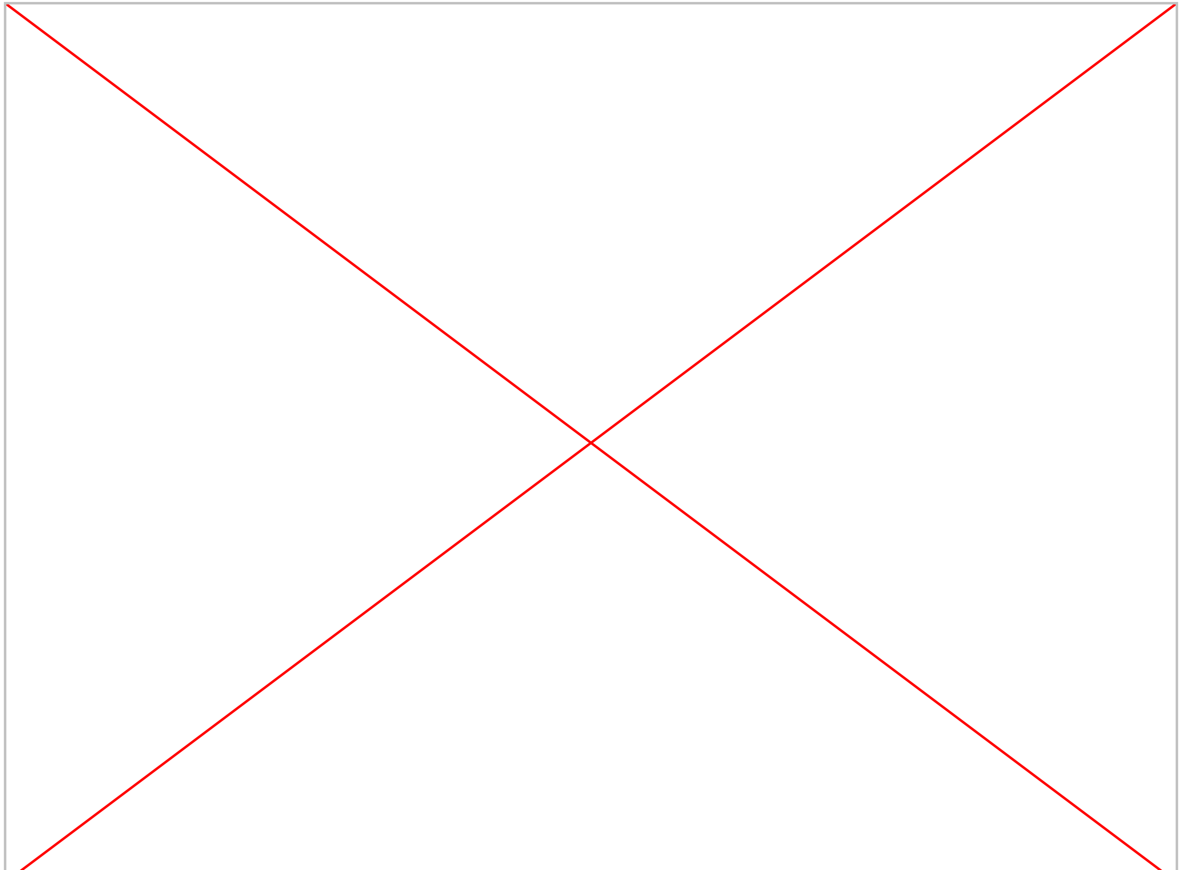
$$m_4 = (E_1 \text{ x } E_2 \text{ x } E_3) \oplus \sim m_4$$

Correction using gates





Hardware Demo



Sources we used

- https://docs.google.com/document/d/1dmfaHM1yQhStLPJ9fYwTUpYEeFIZNqfZtUVphUjjrqs/edit?usp=sharing

Extra slides

Decoding with the Parity Check Matrix (Two Errors)

$$
\underset{\mathbf{H}}{\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}}
\underset{\mathbf{c}^T}{\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}}
$$

Channel Noise

Decoding with the Parity Check Matrix (Two Errors)

$$\mathbf{H} \qquad \mathbf{c}^{\mathsf{T}}$$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Bits **flipped.**

Decoding with the Parity Check Matrix (Two Errors)

$$\overset{\mathbf{H}}{\begin{pmatrix} \boxed{\begin{matrix}1\\1\\0\end{matrix}} & \begin{matrix}1\\0\\1\end{matrix} & \begin{matrix}0\\1\\1\end{matrix} & \begin{matrix}1\\1\\1\end{matrix} & \begin{matrix}1\\0\\0\end{matrix} & \begin{matrix}0\\1\\0\end{matrix} & \begin{matrix}0\\0\\1\end{matrix} \end{pmatrix}} \overset{\mathbf{c}^T}{\begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

The syndrome identifies the wrong error column (column 1) when column 6 and 7 caused the error.