Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

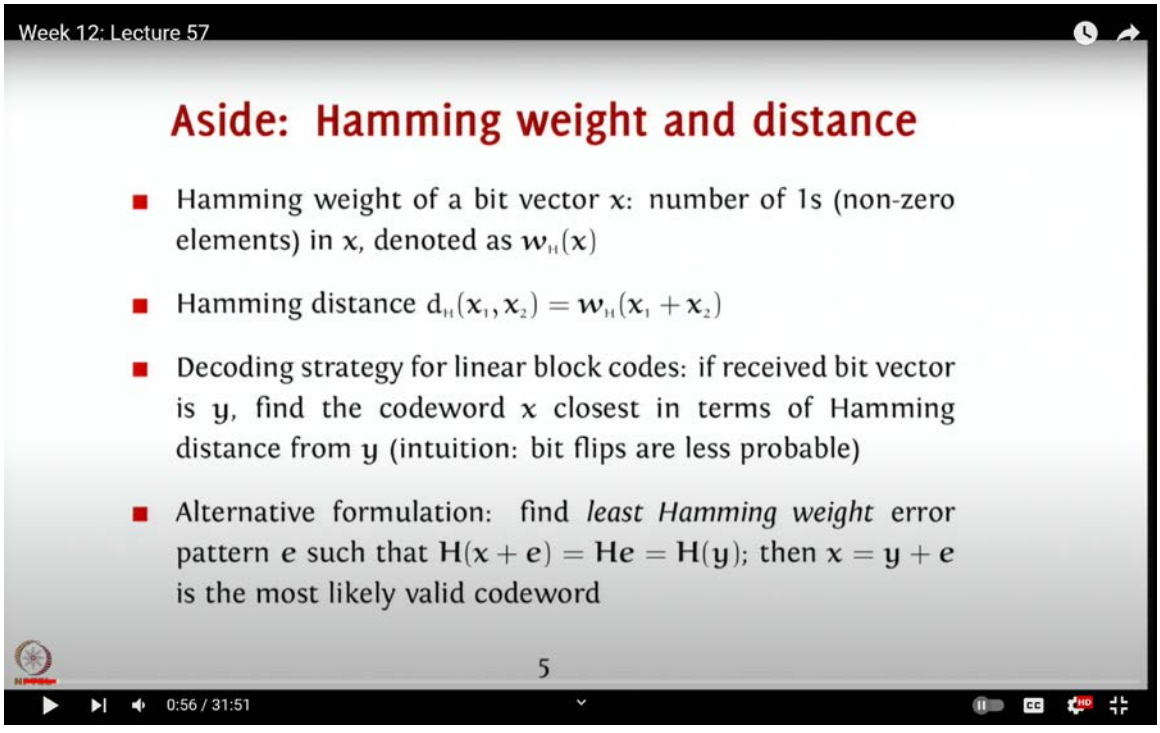
Week-12

Lecture-57

Error Control Coding: Hamming Codes

Welcome to this lecture on Digital Communication using GNU Radio. I'm Kumar Appiah from the Department of Electrical Engineering at IIT Bombay. Today, we will continue our discussion on perfect codes and Hamming codes. To recap, a perfect code ensures that for any received n-length bit sequence, a minimum number of bit flips will yield a unique code word, leaving no ambiguity.

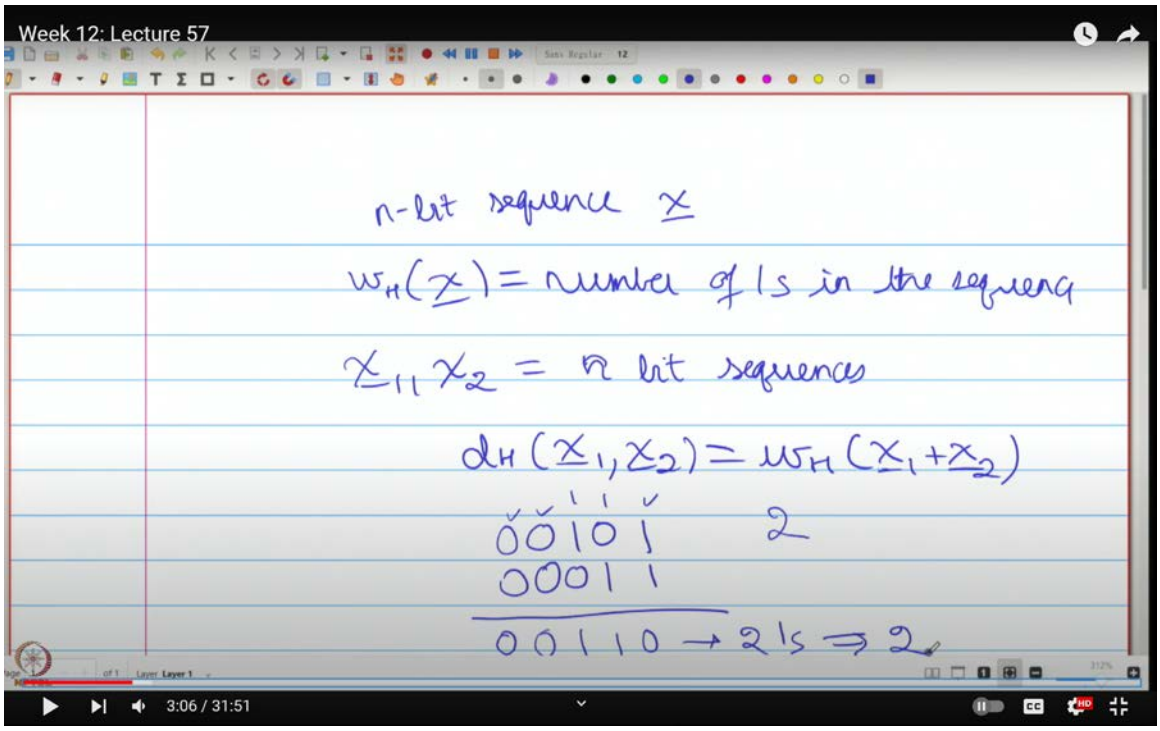
(Refer Slide Time: 00:56)



For instance, the 3-bit repetition code is a perfect code because each 3-bit sequence maps uniquely to a single code word with the least number of bit flips. In contrast, the 4-bit

repetition code suffers from ambiguity and thus is not a perfect code. We also covered the concepts of Hamming weight and distance in our previous discussion.

(Refer Slide Time: 03:06)



Let's briefly revisit these concepts. For binary vectors, Hamming weight and distance are closely related. Specifically, the Hamming weight of an n-bit sequence is the count of 1s in that sequence. For two n-bit sequences, x_1 and x_2, the Hamming distance, denoted as d_H, is defined as the weight of the XOR of x_1 and x_2.

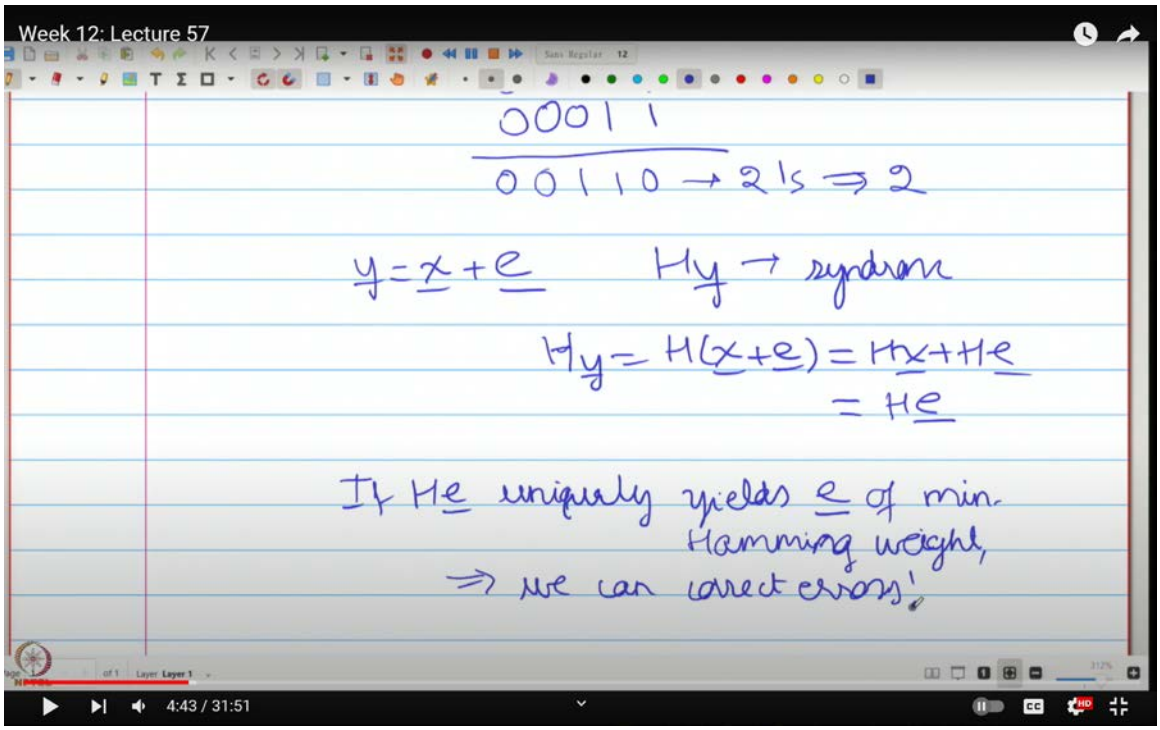
$$0 \text{ XOR } 0 = 0$$
$$0 \text{ XOR } 0 = 0$$
$$1 \text{ XOR } 1 = 0$$
$$0 \text{ XOR } 1 = 1$$
$$1 \text{ XOR } 1 = 0$$

To illustrate, consider two 5-bit sequences: 00101 and 00111. To find where the bits differ, we compare them bit by bit: same, same, different, different, same. Thus, the Hamming distance should be 2. Alternatively, we can compute this by XOR-ing the sequences

bitwise.

The result is 00101 XOR 00111 = 00010, which has a weight of 2, indicating a Hamming distance of 2. The Hamming distance is a useful metric because it represents the number of bit flips required to transform one n-bit sequence into another, which is crucial for determining the minimum number of bit flips needed to correct errors and obtain a valid code word.

(Refer Slide Time: 04:43)



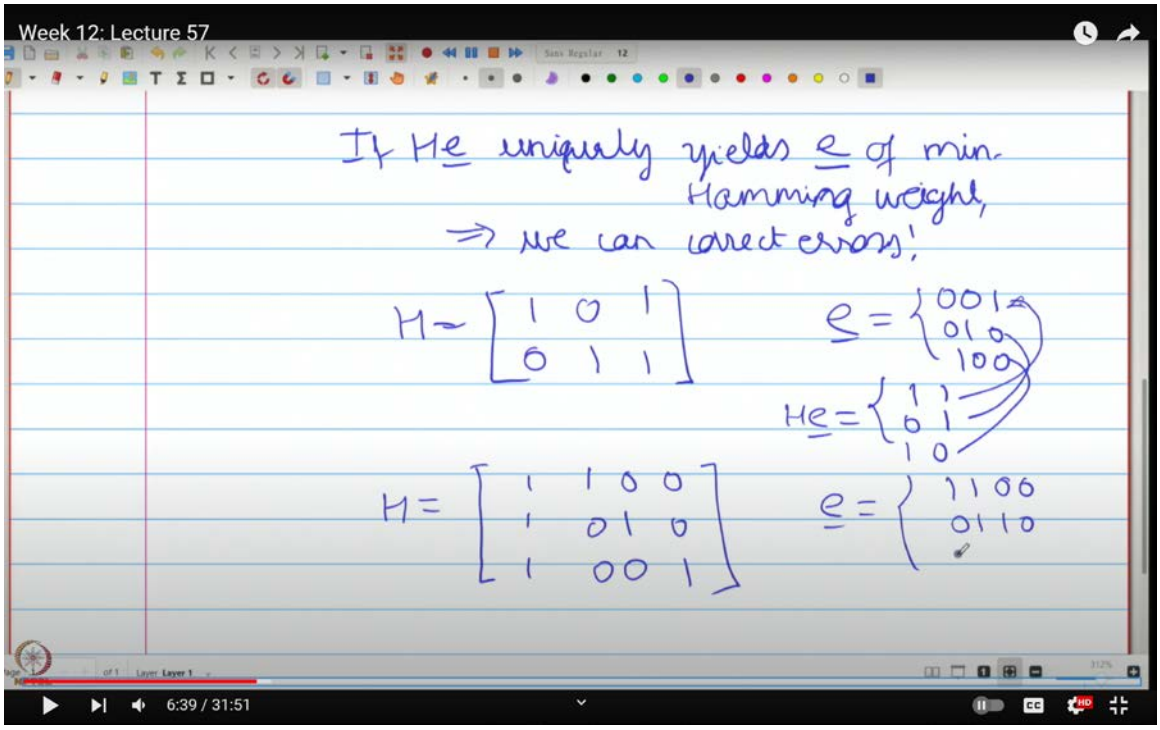
Lastly, remember that our goal is to find the least Hamming weight error pattern e such that H(X + e) = H(Y), where Y = X + e represents the received sequence with an error pattern e.

When you compute H_Y, known as the syndrome, it is given by $H_Y = H \times (X + E)$, which simplifies to $H_Y = HX + HE$. Since H X equals zero, this reduces to $H_Y = HE$. The advantage of using a linear block code is that if H E can be uniquely mapped to an error pattern E, specifically, if H E uniquely identifies E and E has the minimum Hamming weight, then we can correct errors effectively.

To illustrate, recall the 4-bit parity code. We were unable to uniquely map H E to E in this case. For example, consider the repetition code with a parity-check matrix. If the generator matrix G is [1; 1; 1], there are only two 1-bit error patterns to consider. The mapping of H E back to E is feasible because:

$$\begin{aligned}&\text{For } 001, \text{ the syndrome is } 11 \text{ (error pattern)}\\&\text{For } 100, \text{ the syndrome is } 10 \text{ (error pattern)}\\&\text{For } 110, \text{ the syndrome is } 11 \text{ (error pattern)}\end{aligned}$$

(Refer Slide Time: 06:39)



Thus, there is a clear mapping between these patterns and their syndromes. However, if we consider a 4-bit repetition code where G is [1; 1; 1; 1], correcting a single error is straightforward because the syndrome will always uniquely identify the error, whether it's 1 1 1 1 or any other single-bit error pattern.

Let's check the syndromes for some error patterns. For example, if we have 1 1 0 0:

1. Multiply and sum the first two columns to get 0; 1; 1; 0; 1; 1; 0.

2. For 1 0 1 0, the result is 0; 1; 0.

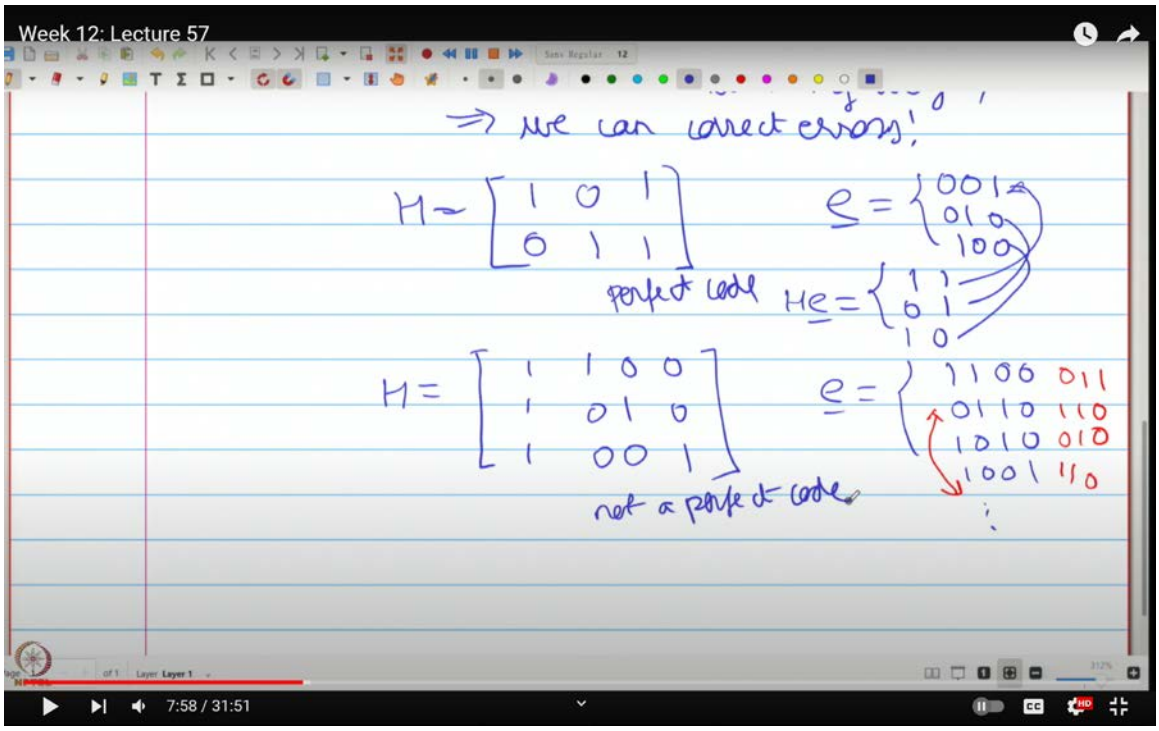
3. For 1 0 0 1, the result is 1; 0; 0; 1; 1; 1; 0.

Notice that patterns yielding the same syndrome indicate that not all 2-bit error sequences can be uniquely corrected. Therefore, this code is not perfect.

To clarify, a perfect code ensures that every possible sequence can be uniquely mapped to a single valid code word with a minimum number of bit flips. For example, the 3-bit repetition code is perfect because:

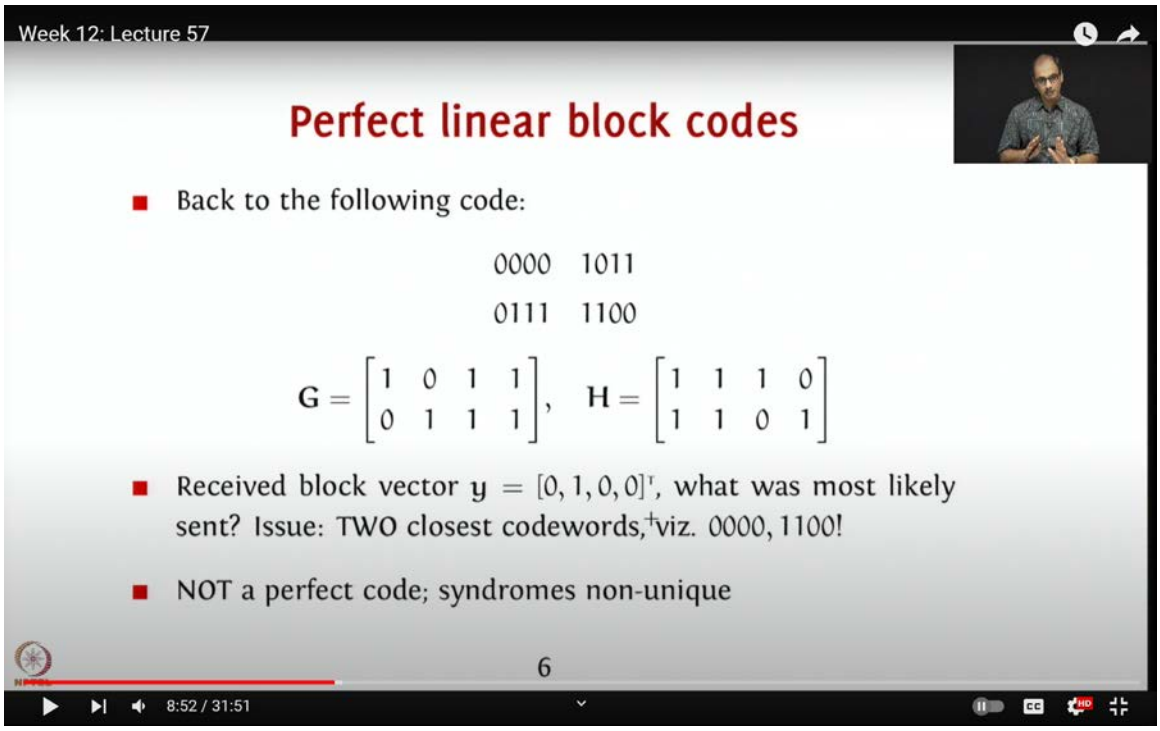
- The sequence 0 0 0 maps to itself.
- The sequence 1 1 1 maps to itself.

(Refer Slide Time: 07:58)



Here, a single bit flip will map to 1 1 1, and flipping more bits will eventually map back to 0 0 0. Thus, all sequences are uniquely mapped, demonstrating the concept of a perfect code.

(Refer Slide Time: 08:52)



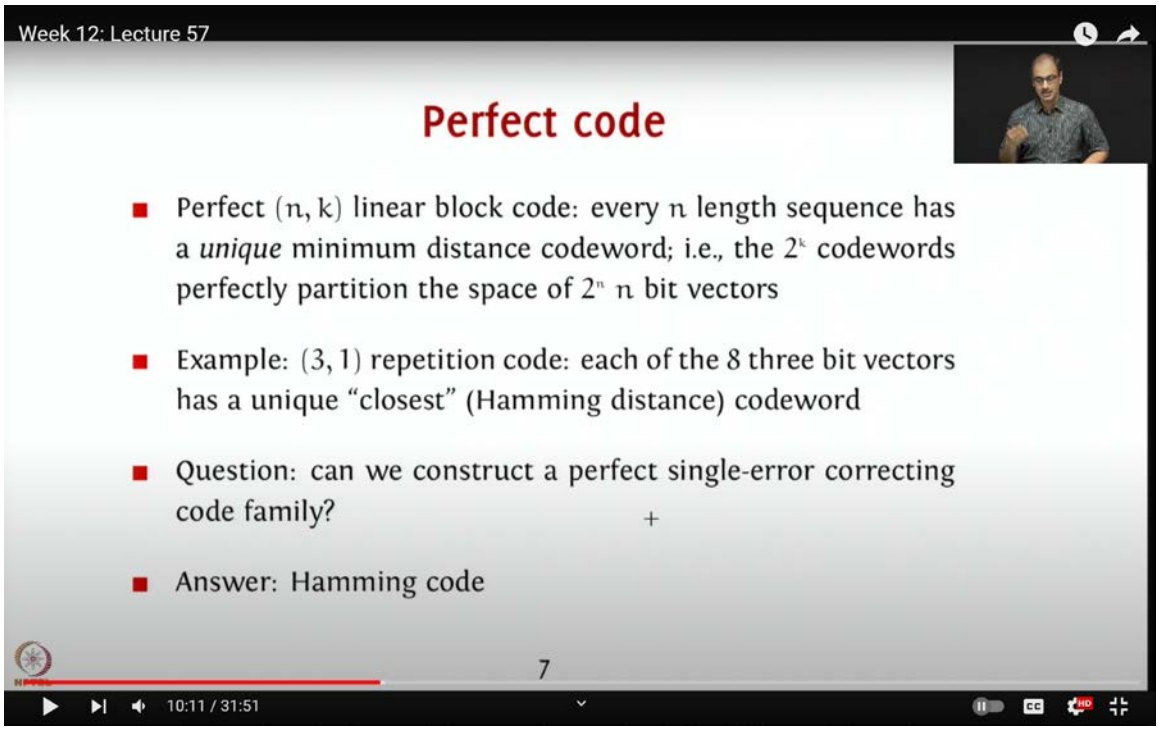
To ensure that the minimum distance code words are unique, we need to construct codes where, given an n-length bit vector, performing the minimum number of bit flips will lead us to a distinct code word. The question then arises: can we construct a family of perfect single error-correcting codes? The answer is yes, and we are going to focus on a specific family known as Hamming codes.

Hamming codes are among the earliest and most well-known error control codes. They were pivotal in the development of subsequent modern coding schemes. Their uniqueness and ease of understanding make them a great subject for detailed discussion. We will delve into constructing Hamming codes using their parity-check matrix, rather than focusing on the intricate details of their formulation.

The Hamming code belongs to the class of perfect (n, k) linear block codes. A perfect (n, k) linear block code ensures that every n-bit sequence has a unique minimum distance code word. In other words, no other code word will have the same minimum distance as this particular code word. For Hamming codes, n is chosen to be 2^m - 1, and k is 2^m - 1 - m,

where m is an integer that starts from 1 and increases.

(Refer Slide Time: 10:11)



Let's analyze this choice. For binary Hamming codes:

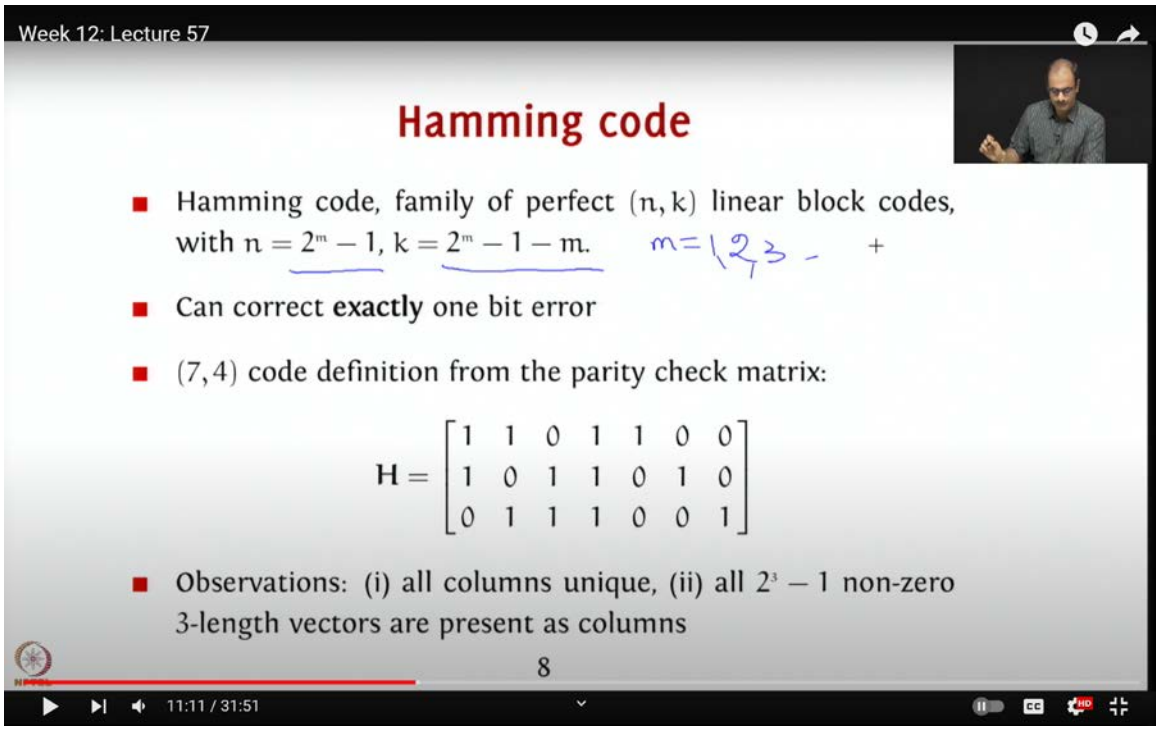
- If m = 1, we get n = 2^1 - 1 = 1 and k = 2^1 - 1 - 1 = 0. This is not a valid code.
- For m = 2, we get n = 2^2 - 1 = 3 and k = 2^2 - 1 - 2 = 1. This corresponds to the 3-bit repetition code, which qualifies as a Hamming code according to this definition.
- For m = 3, we have n = 2^3 - 1 = 7 and k = 2^3 - 1 - 3 = 4. This code is referred to as the [7, 4] Hamming code.

Now, why do we use n = 2^m - 1 and k = 2^m - 1 - m? To understand this, let's look at the parity-check matrix. Our goal is to correct exactly one error. In terms of perfect codes, this means every syndrome (which is the result of the parity-check matrix) should be unique. For a perfect code, the minimum distance between code words is such that if you flip one bit, the resulting vector should still be closest to the original code word.

Therefore, the minimum distance between code words must be such that any single bit flip

will still identify the closest code word correctly, ensuring reliable error correction.

(Refer Slide Time: 11:11)



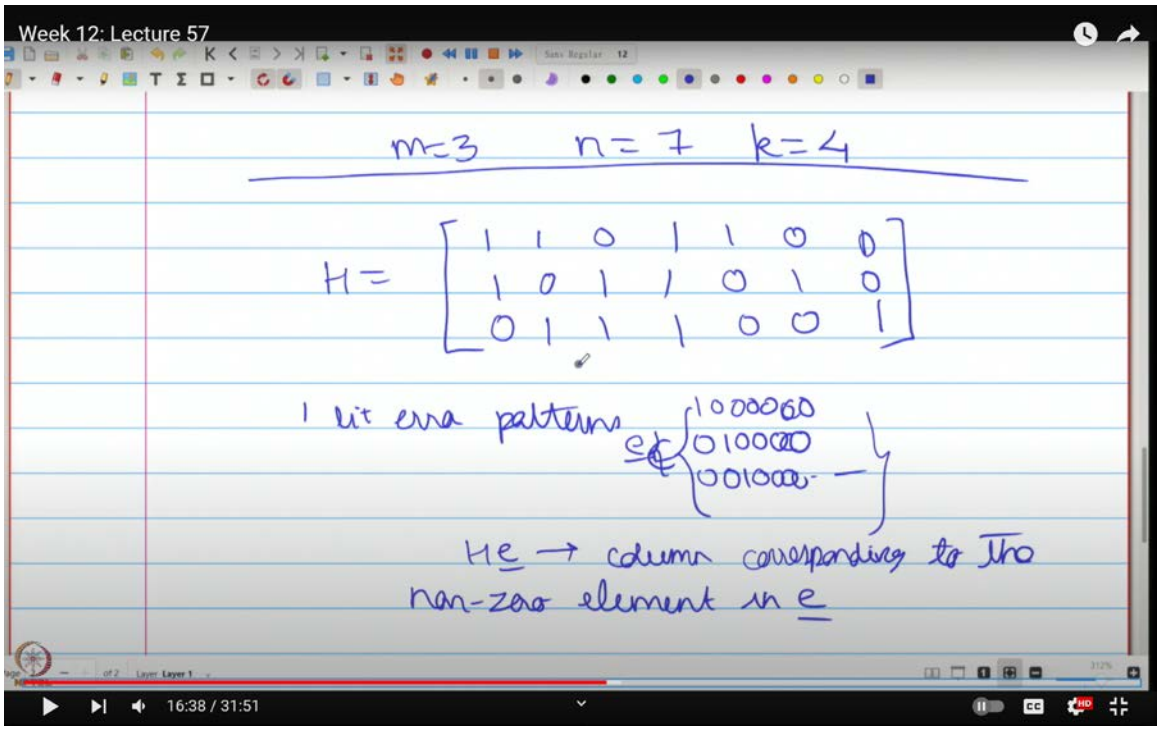
Let's consider the scenario where you flip any one bit. The key characteristic of the Hamming code is that the closest code word, in terms of Hamming distance, will always be the original code word from which you started. To define this precisely, we use a specific form of the parity-check matrix for the [7, 4] Hamming code.

Here, we have a parity-check matrix which we'll denote in a particular format. There are indeed multiple possible parity-check matrices for the [7, 4] Hamming code, but we'll use one particular matrix for illustrative purposes. This matrix includes the pattern [1, 1, 0, 1] and an identity matrix. Placing the identity matrix in this position simplifies the process of writing the generator matrix using an efficient method known as the $A\ I$ trick.

With this specific parity-check matrix for the [7, 4] Hamming code, our claim is that each one-bit error pattern results in a unique syndrome. Let's break this down: if an error occurs in a particular bit position, the syndrome generated is unique to that position. For instance, if an error occurs in the first bit, the resulting syndrome is [1, 1, 0]; if the error is in the

second bit, the syndrome is [1, 0, 1]; and if the error is in the third bit, the syndrome becomes [0, 1, 1]. Thus, each one-bit error corresponds to a distinct syndrome.

(Refer Slide Time: 16:38)



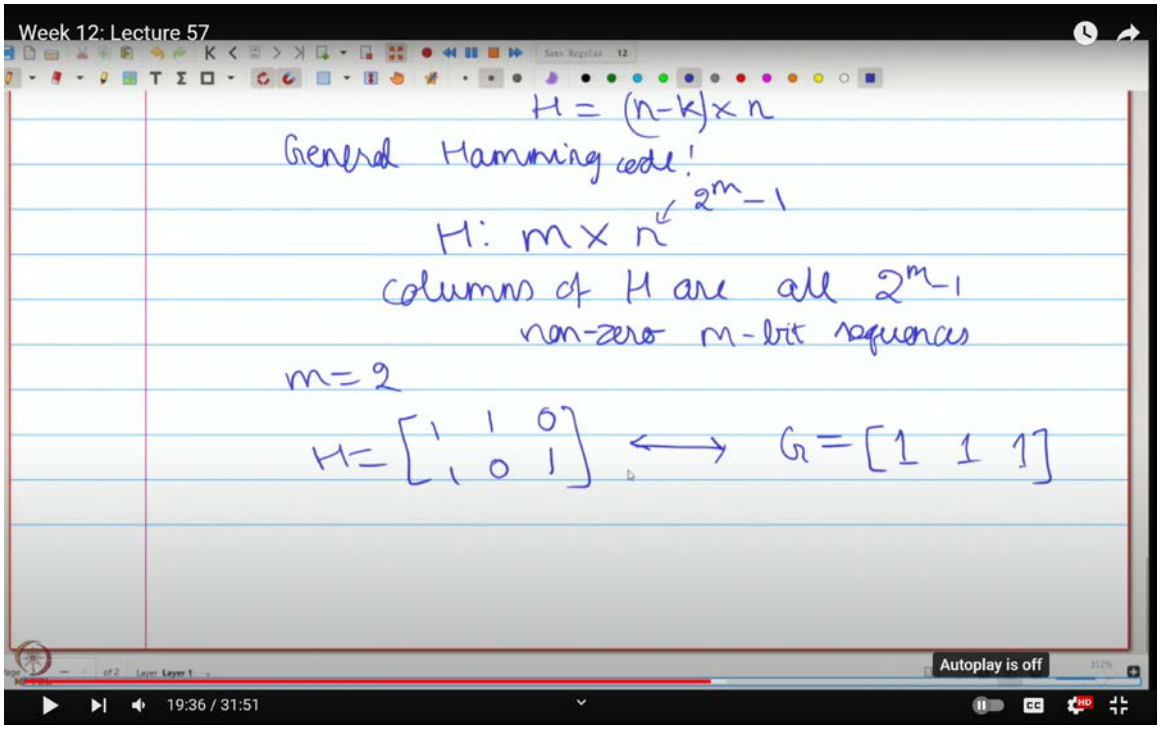
To correct errors, you compute $H \times Y$, where Y is the received n-bit sequence. This calculation yields $H \times X + E$, which simplifies to $H \times E$, where E is the error vector. The syndrome $H \times E$ will correspond to one of the columns in the parity-check matrix. By identifying which column matches the syndrome, you can determine the error pattern and correct it.

In essence, the design of the matrix H ensures that each single-bit error results in a unique syndrome, allowing straightforward error detection and correction. The number of single-bit error patterns is equal to n, and therefore, H must have n unique columns. Consequently, the size of H is $(n - k) \times n$, ensuring that it can accommodate all possible single-bit error patterns uniquely.

Let's break this down. First, note that for a Hamming code, you have 2^m - 1 single-bit error patterns. Consequently, the parity-check matrix H must have columns representing all

possible non-zero m-bit sequences. For example, if m = 3, then k = 4, and H will be an $m \times n$ matrix where n = 2^m - 1. In this case, the matrix H will have columns consisting of all possible non-zero m-bit sequences.

(Refer Slide Time: 19:36)



Let's construct a Hamming code for m = 2. We will compute the parity-check matrix H. Here, we have randomly placed the columns but ensured the matrix conforms to the required structure. For this matrix, using the identity trick, we can find the generator matrix G. This example corresponds to the [3, 1] repetition code.

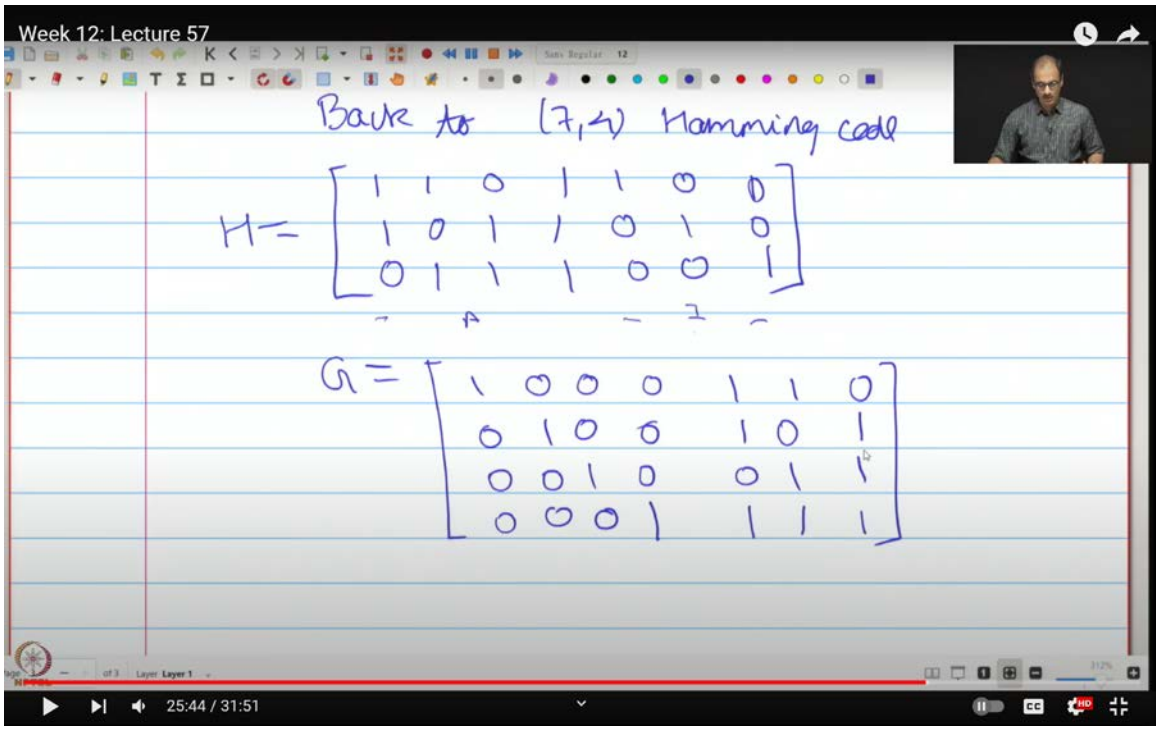
Now, for m = 3, let's consider a more detailed example. We'll construct the parity-check matrix H for this case.

In this matrix, I've arranged the columns manually. For simplicity, I've used some tricks to avoid writing too much, including copying and adjusting columns as needed.

With n = 15 and k = 11, this is a [15, 11] Hamming code. Constructing the generator matrix G for this code can be an exercise for you. You can swap columns to align the identity

matrix on the right, which will give you the required structure.

(Refer Slide Time: 25:44)



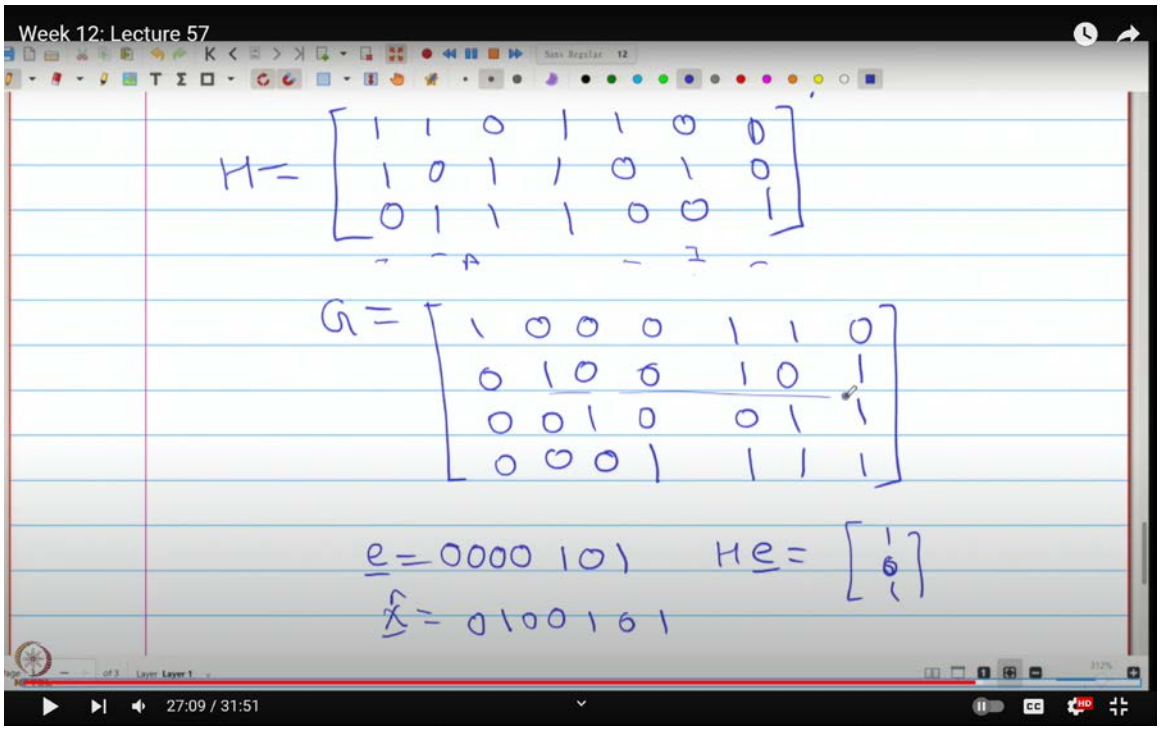
In the constructed parity-check matrix, all columns are unique, and they include all 2^3 - 1 non-zero 3-bit vectors. The key observation here is that multiplying H by a zero vector results in zero, which is true for any parity-check matrix. However, multiplying by any one-bit vector will always produce a unique result, and similarly, no two different 2-bit vectors will sum to zero. This implies that you need to flip at least three bits to correct an error, ensuring that the code words of the Hamming code have a minimum distance of at least three. Therefore, this guarantees that any single-bit error pattern can be uniquely decoded, making the Hamming code an efficient and effective error-correcting code.

Let's clarify the key concept here. Using the same methods we've discussed, we can construct the generator matrix for the Hamming code. For the [7,4] Hamming code, we'll employ the same approach used previously. We start with the identity matrix and an accompanying matrix A. For the generator matrix G, we arrange it also.

Let’s examine the properties of this matrix. Each code word in the Hamming code has a

minimum Hamming weight of 3. This means every non-zero code word contains at least three 1's. For instance, a code word like 1 0 0 0 0 0 0 or 1 1 0 0 0 1 1 has at least three 1's.

(Refer Slide Time: 27:09)



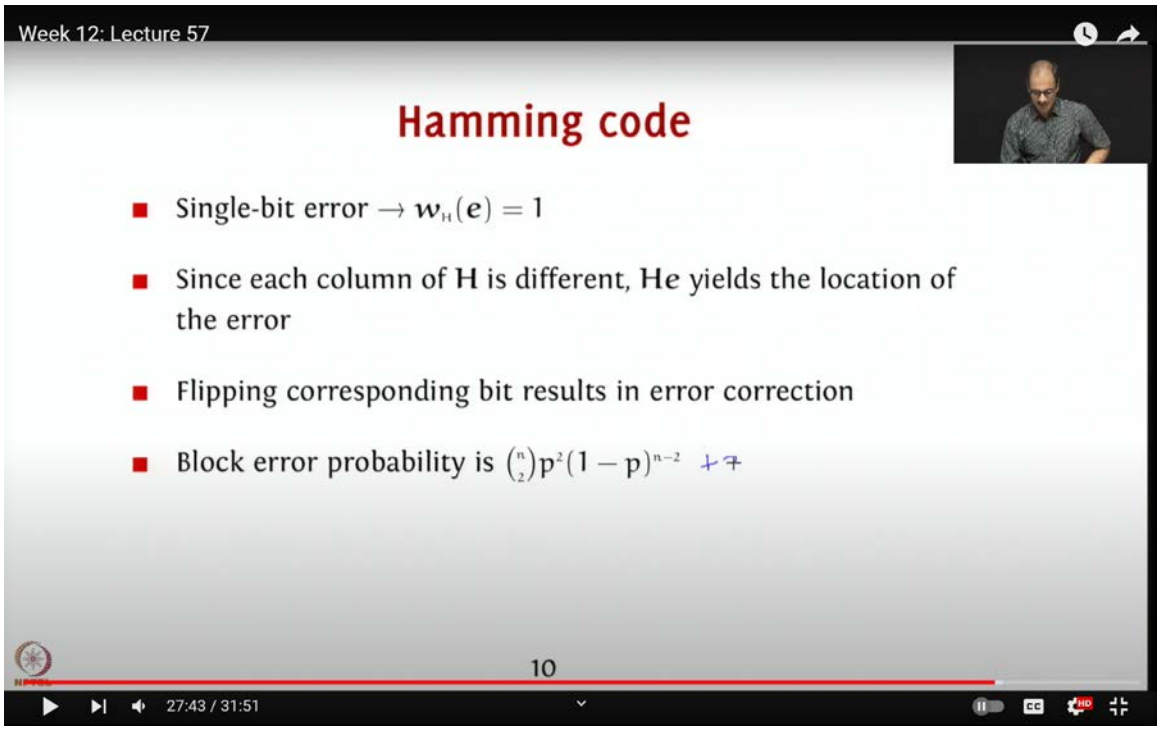
To verify this, let’s check how these code words interact. If you take any code word and add it to another, such as 1 1 0 plus 1 0 0 plus 0 1 0, the result is 0. This confirms that adding specific code words results in zero, consistent with the properties of linear codes. By performing similar calculations for all possible vectors, we can verify that there are 16 code words, each with a length of 7. If you flip one bit in any code word, you can detect an error. However, flipping two bits can lead to detection but not necessarily correction.

Let’s illustrate this with an example. Suppose we have the vector 0 0 0 0 1 0 1. If you apply this to the error vector, the syndrome might incorrectly indicate where the error is. For instance, the syndrome calculation might show 1 0 1, suggesting an error in a different location than where it actually occurred.

The correct decoded code word, in this case, would be 1 0 1 0 1 0 0 1 0 1, which is one of the valid code words. The Hamming code is perfect for single-bit errors because it

guarantees correction. However, when two-bit errors occur, while the code can detect the errors, it cannot always correct them.

(Refer Slide Time: 27:43)



Thus, with a single-bit error, the unique columns of H ensure that the error can be located and corrected. The probability of block errors can be analyzed using combinatorial terms, but typically, the p^2 term, which accounts for two-bit errors, will dominate.

The Hamming code is indeed an elegant solution for single-bit error correction, but it has its limitations when it comes to handling multiple errors. Specifically, while the Hamming code always maps to a unique element, if you encounter two-bit errors, it might incorrectly identify a valid code word, resulting in a block error. Determining the exact bit error probability for such cases is a bit more complex and is covered in specialized references, which you can consult for detailed calculations.

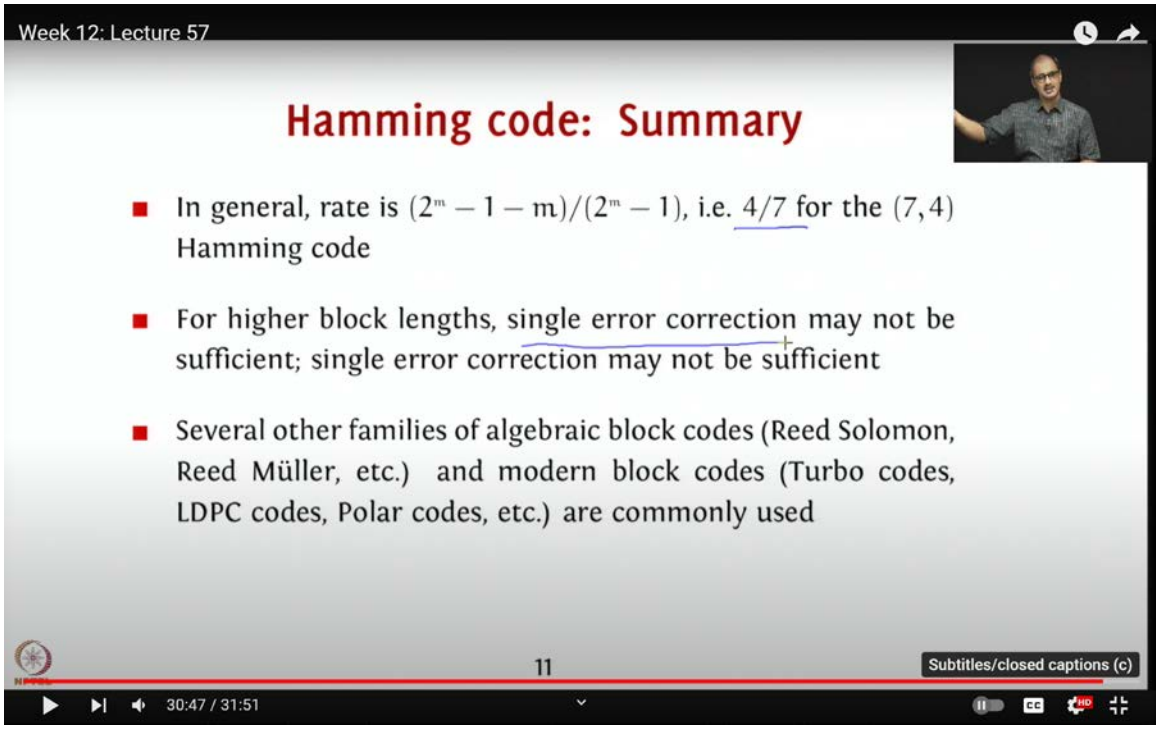
As for the rate of the Hamming code, it is given by the formula:

$$\text{Rate} = \frac{2^m - 1 - m}{2^m - 1}$$

For the [7,4] Hamming code, this results in a rate of $\frac{4}{7}$. If you extend this to larger values of m, for example, with m = 4, the rate becomes $\frac{11}{15}$, and so on. As m increases, the Hamming code can achieve better rates, but the challenge of single error correction persists.

In practice, for larger block lengths, single error correction becomes increasingly insufficient. As the block size grows, the likelihood of encountering more than one error rises significantly. This is analogous to repeatedly tossing a coin: although getting tails might be rare on a single toss, the probability of landing two tails increases with the number of tosses. Thus, as block lengths increase, the probability of encountering multiple errors becomes much higher, making single error correction less effective.

(Refer Slide Time: 30:47)



Using Hamming codes for very large block lengths can be efficient in terms of rate, but it is generally not ideal for practical applications. This is because the ability to correct only a single-bit error becomes inadequate when the probability of errors in a binary symmetric channel is relatively high. For error probabilities such as 10^{-3}, 10^{-4}, or 10^{-5}, Hamming codes can be quite beneficial, even for large values of m. However, when faced with a

substantial number of bit errors, the limitations become apparent.

Before we conclude our discussion on Hamming codes, it's important to recognize that they serve as a foundation for many modern coding techniques. For instance, Reed-Solomon and Reed-Muller codes are widely used in various applications. Reed-Solomon codes are particularly common in error correction for optical discs like compact discs and DVDs, while Reed-Muller codes are used in block error control.

Additionally, there are advanced families of block codes such as Turbo codes, LDPC codes, and Polar codes. These codes are designed with randomness-oriented strategies and perform exceptionally well with large block lengths, offering excellent error performance and high rates. They are known for their efficiency and optimality, particularly in terms of channel capacity, which we will explore further in upcoming lectures.

Moreover, convolutional codes are another category used for continuous error control. These codes form a trellis structure to handle errors progressively, similar to equalization techniques. For more detailed information on these and other coding techniques, I encourage you to consult comprehensive textbooks on error control coding.

In summary, while Hamming codes and linear block codes provide an effective and efficient starting point for error correction, especially in noisy channels, modern coding techniques offer advanced solutions for handling errors in more complex scenarios. In the next lecture, we will implement the Hamming code using GNU Radio and analyze its performance. Thank you.