Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

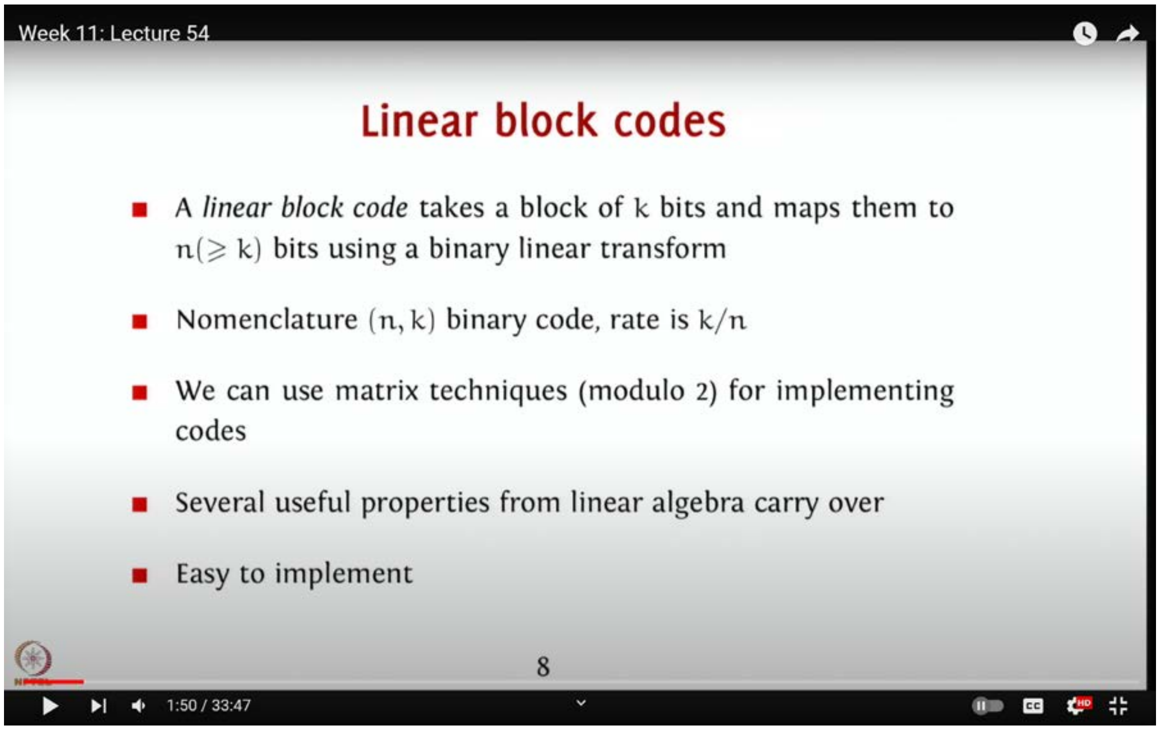
Week-11

Lecture-54

Error Control Coding: Linear Block Codes

Welcome to this lecture on Digital Communication Using GNU Radio. I am Kumar Appiah from the Department of Electrical Engineering at IIT Bombay. In our previous set of lectures, we explored error control codes, focusing particularly on parity check codes and repetition codes. We examined how parity check codes enforce even parity, meaning the code word contains an even number of 1s, and how repetition codes work by repeating a bit 3 or 5 times.

(Refer Slide Time: 01:50)



Week 11: Lecture 54

Linear block codes

- A *linear block code* takes a block of k bits and maps them to $n(\geqslant k)$ bits using a binary linear transform
- Nomenclature (n, k) binary code, rate is k/n
- We can use matrix techniques (modulo 2) for implementing codes
- Several useful properties from linear algebra carry over
- Easy to implement

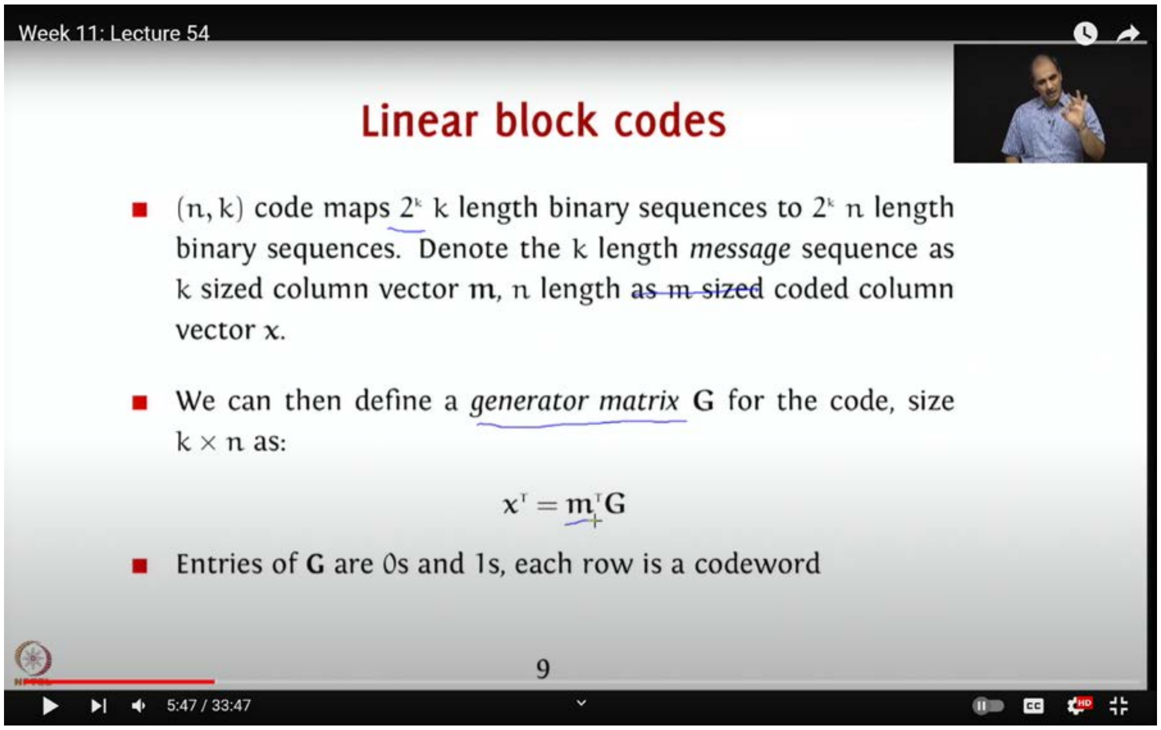
8

We found that repetition codes significantly reduce the probability of error on the binary symmetric channel, but they come with trade-offs. Specifically, while the parity check code can only detect a single error without correcting it, the repetition code can correct errors, albeit with very low efficiency due to its low data rate.

To broaden our understanding of coding strategies, in this lecture, we will delve into linear block codes, which are widely used error control mechanisms. Linear block codes are fundamental to many practical error control schemes, and we will explore the Hamming code more closely in subsequent lectures.

So, what exactly is a linear block code? A linear block code is defined by two parameters: n and k. Essentially, you take a block of k bits from your bit sequence and map it to n bits, where n is typically greater than k, using a binary linear transformation. This linear transformation is highly effective because it can be efficiently implemented in both software and hardware. The terminology we use, which you might already be familiar with, refers to this as an n,k binary code, where the rate of the code is $\frac{k}{n}$.

(Refer Slide Time: 05:47)



Linear block codes

- (n, k) code maps 2^k k length binary sequences to 2^k n length binary sequences. Denote the k length *message* sequence as k sized column vector m, n length ~~as m sized~~ coded column vector x.
- We can then define a *generator matrix* G for the code, size $k \times n$ as:

$$x^T = m^T G$$

- Entries of G are 0s and 1s, each row is a codeword

9

To implement these codes, matrix techniques are often employed. Since linear block codes are linear by nature, they can be implemented using matrix multiplication, with Modulo 2 addition to perform the necessary operations. Linear algebra provides several useful results, particularly those related to vector spaces, which are instrumental in understanding and implementing these codes. While we won't dive deeply into the intricacies of vector spaces in this lecture, I'll highlight the relevant properties as we go along to help clarify the concepts.

Lastly, it's important to note that since we're working with bits, and operating within a binary field (0s and 1s), the addition and multiplication operations are equivalent to XOR and AND operations, respectively. These operations can be implemented very efficiently, whether using logical gates in hardware or through software.

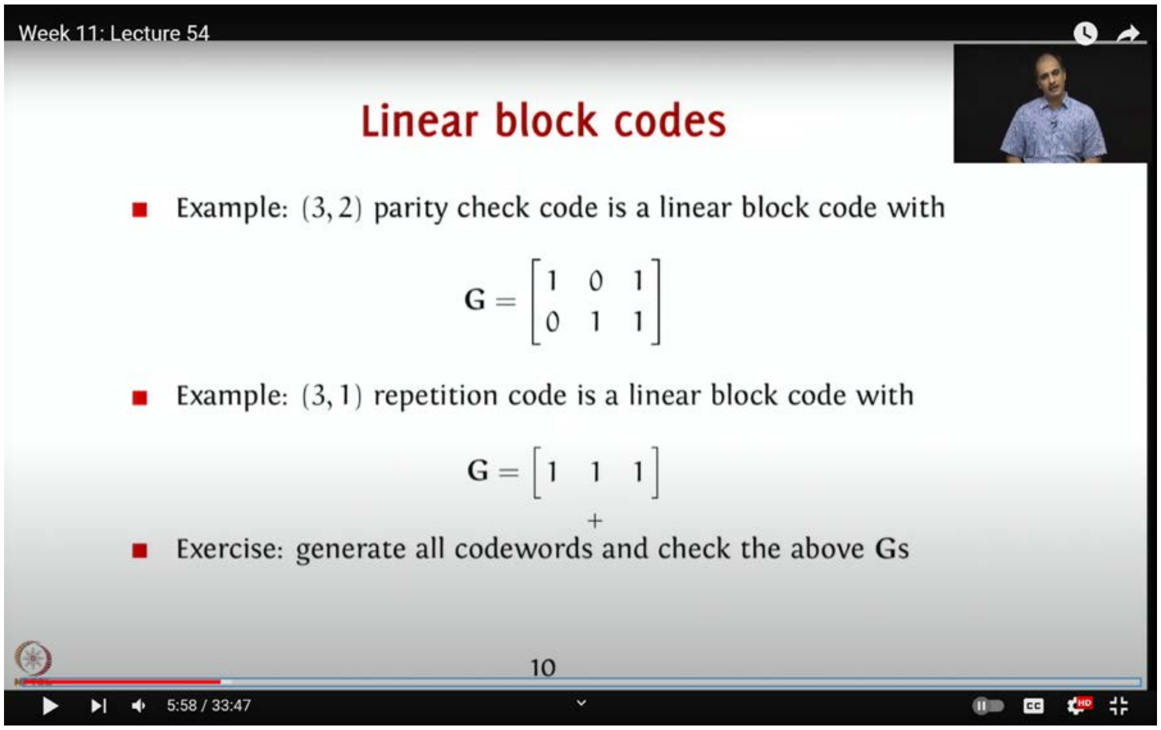
Thus, a linear block code essentially maps 2^k k-length binary sequences to 2^k n-length binary sequences, making it a powerful tool in error control coding.

If you take a block of k information bits, the total number of possible sequences you can generate is 2^k. These sequences are then mapped to 2^k sequences of length n, where n is greater than or equal to k. This implies that redundancy is introduced during the mapping process. For clarity, we represent the k-length message sequence as a column vector m, and the n-length coded sequence as a column vector X. Thus, the k-length column vector $\boldsymbol{m}$ is transformed into an n-length column vector $\boldsymbol{X}$ after applying the coding operation.

Given that this is a linear code, this transformation can be expressed as a linear transformation within the binary field (i.e., 0s and 1s). To do this, we introduce a generator matrix, denoted by $\boldsymbol{G}$. By convention, the generator matrix $\boldsymbol{G}$ is defined in the transpose sense, meaning it has k rows and n columns. When you pre-multiply the vector $\boldsymbol{m}$ by G, which is why we use $\boldsymbol{m}^{\mathrm{T}}$ (the transpose of $\boldsymbol{m}$), you obtain $\boldsymbol{X}^{\mathrm{T}}$ (the transpose of $\boldsymbol{X}$). The elements of $\boldsymbol{G}$ consist of 0s and 1s, with each row representing a code word.

Let’s break down what this transformation accomplishes. The generator matrix $\boldsymbol{G}$ is generally expressed as a "fat" matrix, which means it has more columns than rows, specifically, k rows and n columns. Each row corresponds to a code word because if you select $\boldsymbol{m}$ as $[1\ 0\ 0\ 0]^T\ or\ [0\ 1\ 0\ 0]^T$, you will retrieve the corresponding row of $\boldsymbol{G}$. Each of these rows is indeed a code word. Since $\boldsymbol{m}$ can take 2^k possible values, $m^T G$ can also yield up to 2^k distinct values. In some cases, if the rows are linearly dependent, you might get fewer unique values, but we will set that aside for now.

(Refer Slide Time: 05:58)



Linear block codes

- Example: $(3,2)$ parity check code is a linear block code with

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

- Example: $(3,1)$ repetition code is a linear block code with

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

- Exercise: generate all codewords and check the above Gs

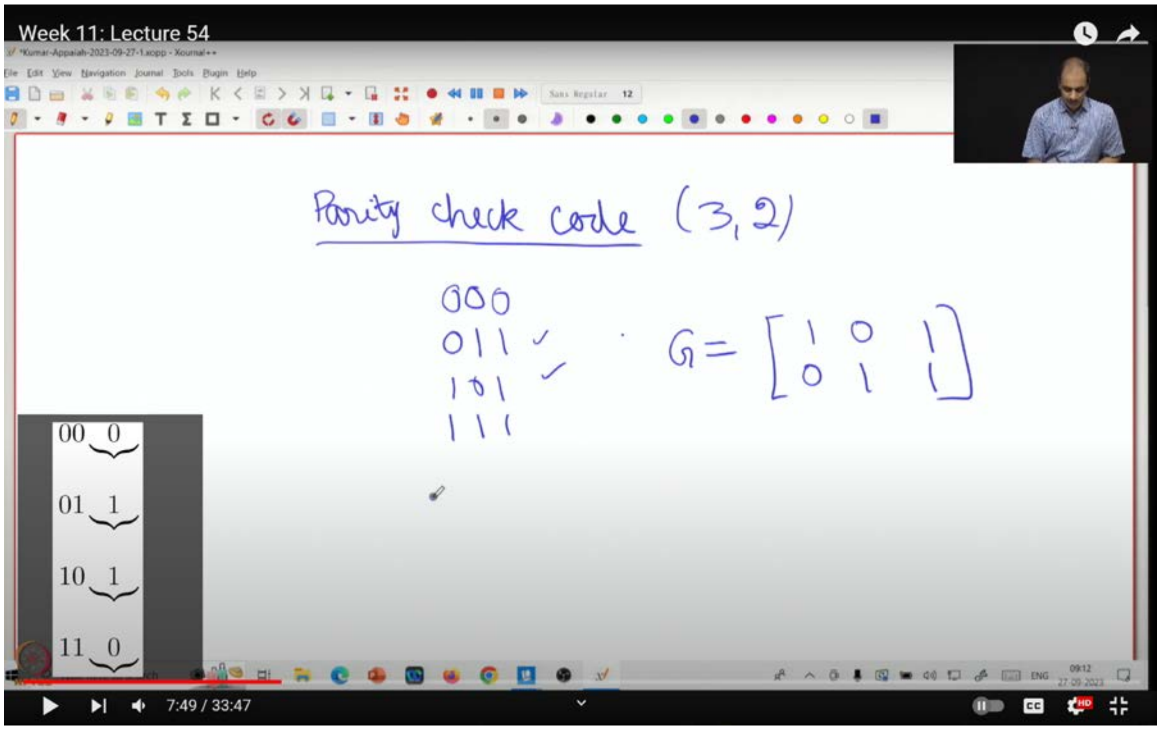
10

If you carefully construct $\boldsymbol{G}$, you will obtain $\boldsymbol{X}$, the coded vector. For every $\boldsymbol{m}$, there is a corresponding $\boldsymbol{X}$ that includes enough redundancy so that even if errors occur, it may still be possible to reconstruct $\boldsymbol{m}$ or at least detect the errors. This redundancy is the key to the error correction capability of the code.

Before delving into the mechanics of how to effectively use this generator matrix, let's ground our understanding with some practical examples. We'll revisit familiar examples, such as the 3-2 parity check code and the 3-1 repetition code, to illustrate these concepts in action.

Let's delve into the parity check code and explore its elements. For even parity, the code includes sequences like 000, 111, and 011. Interestingly, this code qualifies as a linear code. Naturally, it's also a block code, as it involves taking blocks of 2 bits and mapping them to 3 bits, which we discussed in the previous lecture. But more importantly, it's a linear block code. Why is that the case? To illustrate, we'll demonstrate that the code is indeed linear.

(Refer Slide Time: 07:49)



To construct the generator matrix $\boldsymbol{G}$ for this particular code, we'll take a systematic approach. One straightforward method is to select two linearly independent code words. For instance, the sequence 000 is trivial and doesn't offer much utility because multiplying anything by 000 will always yield 000. Instead, let's define our generator matrix $\boldsymbol{G}$ using different rows. For example, we could use the second and third rows, like 101 and 011.

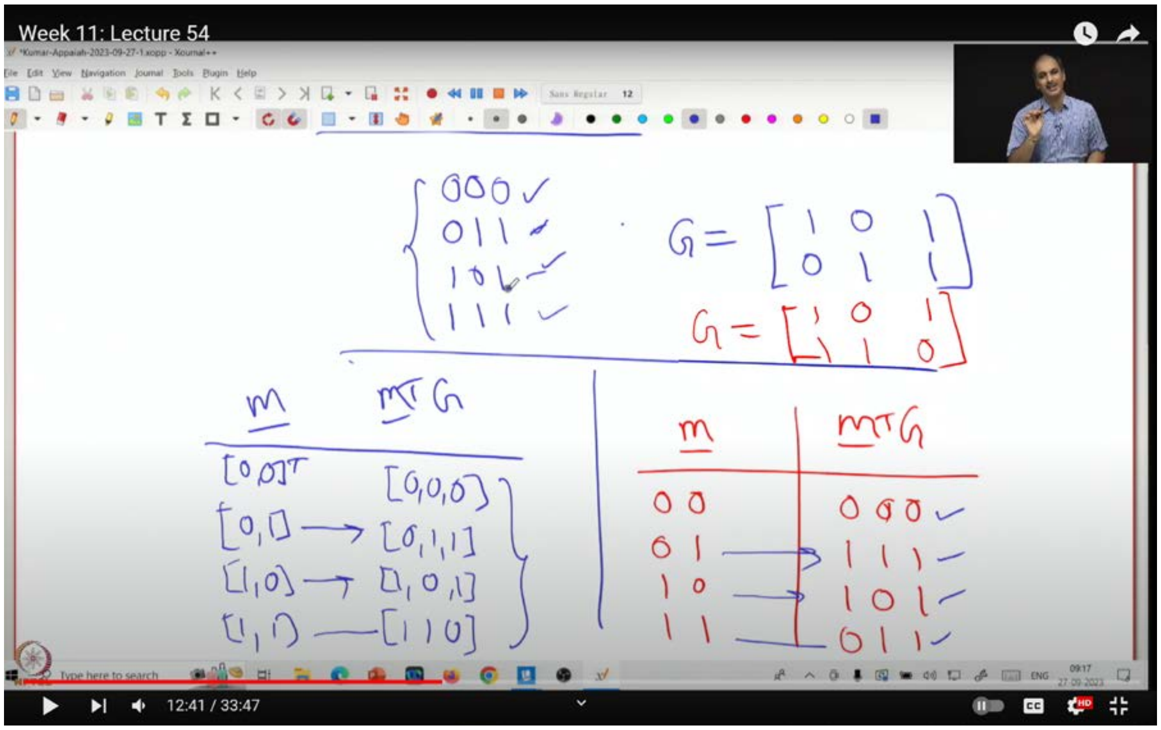
Now, let's formalize this. Here, k = 2, meaning $\boldsymbol{m}$ is a 2-bit column vector. To evaluate $m^T G$ for each $\boldsymbol{m}$, we can represent $\boldsymbol{m}$ as rows. Starting with $m = [0\ 0], m^T G$ yields:

$$m^T G = 0 \times [1\ 0\ 1] + 0 \times [0\ 1\ 1] = [0\ 0\ 0]$$

Next, for $m = [0\ 1]$:

$$m^T G = 0 \times [1\ 0\ 1] + 1 \times [0\ 1\ 1] = [0\ 1\ 1]$$

(Refer Slide Time: 12:41)



$$\left\{\begin{array}{l} 000 \checkmark \\ 011 \checkmark \\ 101 \checkmark \\ 111 \checkmark \end{array}\right.$$

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

m	$m^T G$
$[0\ 0]^T$	$[0,0,0]$
$[0,1]$	$[0,1,1]$
$[1,0]$	$[1,0,1]$
$[1,1]$	$[1\ 1\ 0]$

m	$m^T G$
0 0	0 0 0 ✓
0 1	1 1 1 ✓
1 0	1 0 1 ✓
1 1	0 1 1 ✓

For $m = [1\ 0]$:

$$m^T G = 1 \times [1\ 0\ 1] + 0 \times [0\ 1\ 1] = [1\ 0\ 1]$$

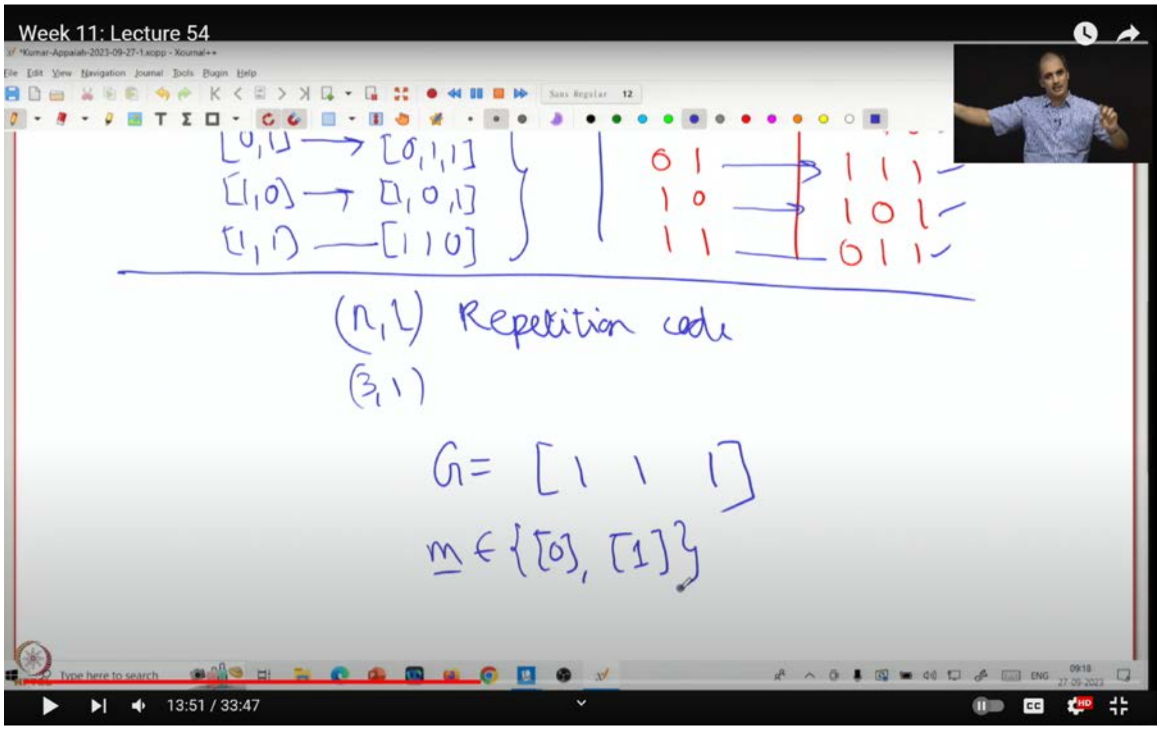
Finally, for $m = [1\ 1]$:

$$m^T G = 1 \times [1\ 0\ 1] + 1 \times [0\ 1\ 1] = [1\ 1\ 0]$$

What can we infer from this? There are several key points to highlight. First, the n-length zero vector is always a valid code word in any linear block code. This is because when $\boldsymbol{m}$ is chosen as the zero vector, the result is always the n-length zero vector, which is a fundamental property of linear transformations. Hence, the zero code word is inherently part of the linear block code.

Secondly, all the code words we've generated here match the original code words, indicating that the generator matrix $\boldsymbol{G}$ we've constructed is indeed valid for this code. The mapping of $\boldsymbol{m}$ to $m^T G$ is evident and consistent.

(Refer Slide Time: 13:51)



$[0,1] \rightarrow [0,1,1]$
$[1,0] \rightarrow [1,0,1]$
$[1,1] \rightarrow [1\ 1\ 0]$

$0\ 1 \rightarrow 1\ 1\ 1$
$1\ 0 \rightarrow 1\ 0\ 1$
$1\ 1 \rightarrow 0\ 1\ 1$

$(n,1)$ Repetition code
$(3,1)$

$G = [1\ \ 1\ \ 1]$

$\underline{m} \in \{[0], [1]\}$

One additional point to consider is whether this generator matrix $\boldsymbol{G}$ is unique. It turns out that $\boldsymbol{G}$ is not necessarily unique. For instance, we could choose a different $\boldsymbol{G}$ matrix, such as:

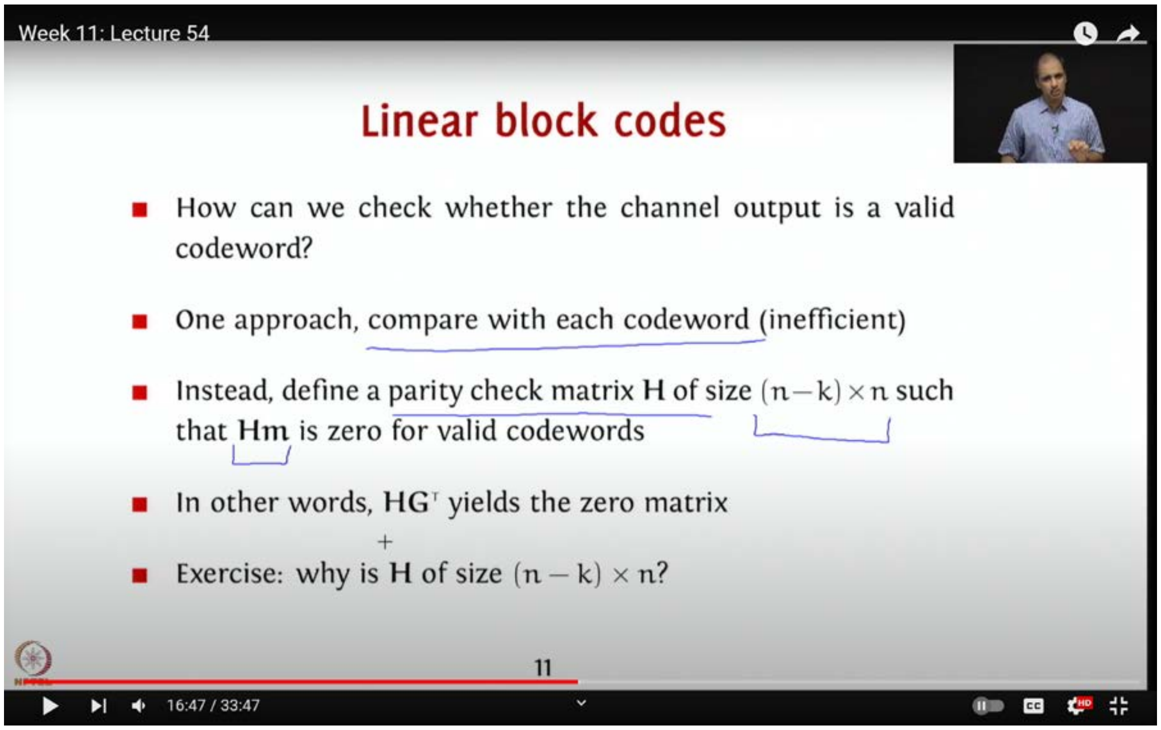
$$G = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

This new matrix is also valid since the rows are linearly independent, meaning you cannot simply multiply one row by a scalar to get the other. Thus, while $\boldsymbol{G}$ can vary, the fundamental properties and results of the code remain consistent.

Let's evaluate the vector $\boldsymbol{m}$ and the result of $m^T G$ for the given scenarios.

First, we start with $m = [0\ 0]$, and for simplicity, I'll only write the key outcomes. When $m = [0\ 0]$, the result is $[0\ 0\ 0]$. For $m = [0\ 1]$, we obtain $[1\ 1\ 0]$. When $m = [1\ 0]$, the outcome is $[1\ 0\ 1]$, and finally, for $m = [1\ 1]$, the result is $[1\ 1\ 1]$.

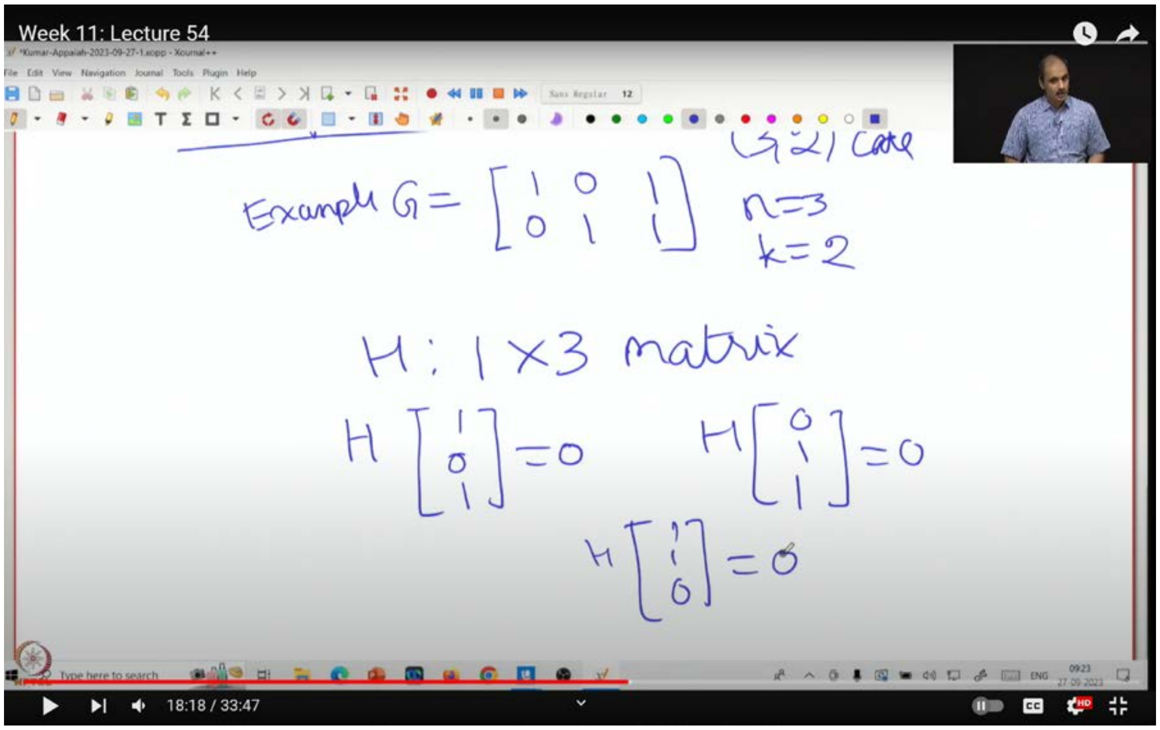
(Refer Slide Time: 16:47)



Linear block codes

- How can we check whether the channel output is a valid codeword?
- One approach, compare with each codeword (inefficient)
- Instead, define a parity check matrix H of size $(n-k) \times n$ such that Hm is zero for valid codewords
- In other words, HG^T yields the zero matrix
- Exercise: why is H of size $(n-k) \times n$?

(Refer Slide Time: 18:18)



(3,2) Case

$$\text{Example } G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad n=3, \ k=2$$

H : 1×3 matrix

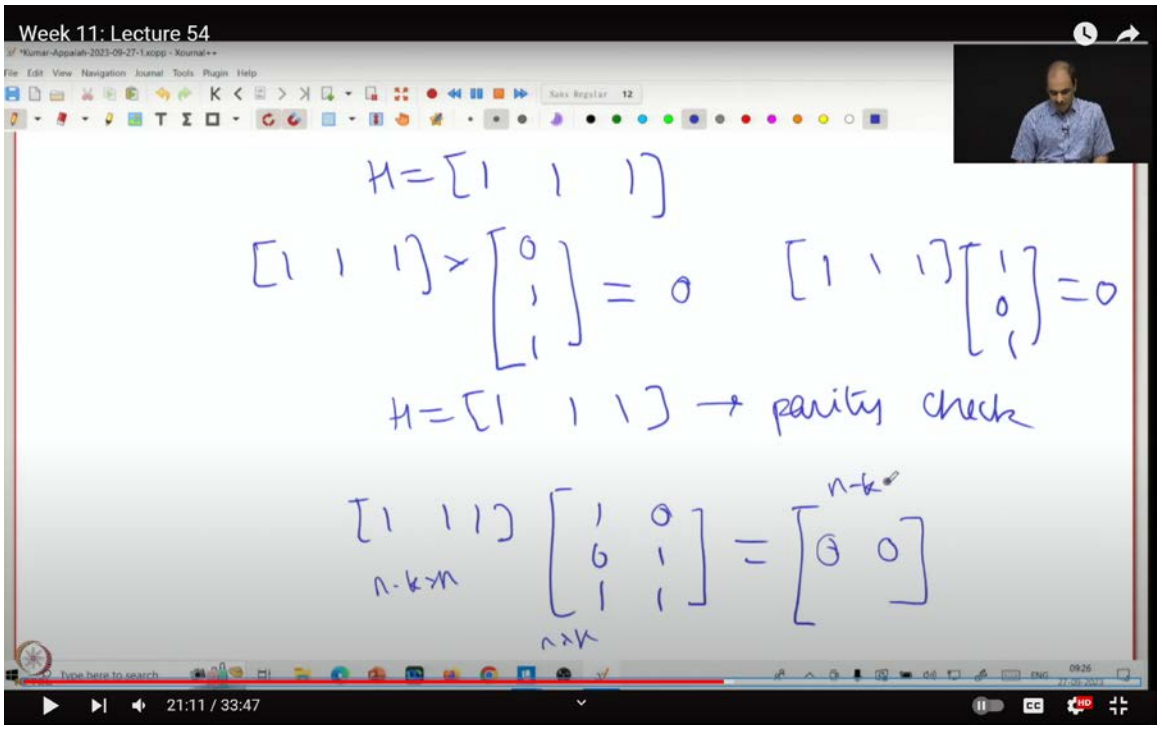
$$H\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0 \qquad H\begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 0$$

$$H\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0$$

If you examine these sequences, you see they correspond to the following mappings:

- $\boldsymbol{m}$ = [0 0] maps to [0 0 0]
- $\boldsymbol{m}$ = [0 1] maps to [1 1 0]
- $\boldsymbol{m}$ = [1 0] maps to [1 0 1]
- $\boldsymbol{m}$ = [1 1] maps to [1 1 1]

(Refer Slide Time: 21:11)



$$H=[1\ \ 1\ \ 1]$$

$$[1\ \ 1\ \ 1]\times\begin{bmatrix}0\\1\\1\end{bmatrix}=0 \qquad [1\ \ 1\ \ 1]\begin{bmatrix}1\\0\\1\end{bmatrix}=0$$

$$H=[1\ \ 1\ \ 1]\rightarrow \text{parity check}$$

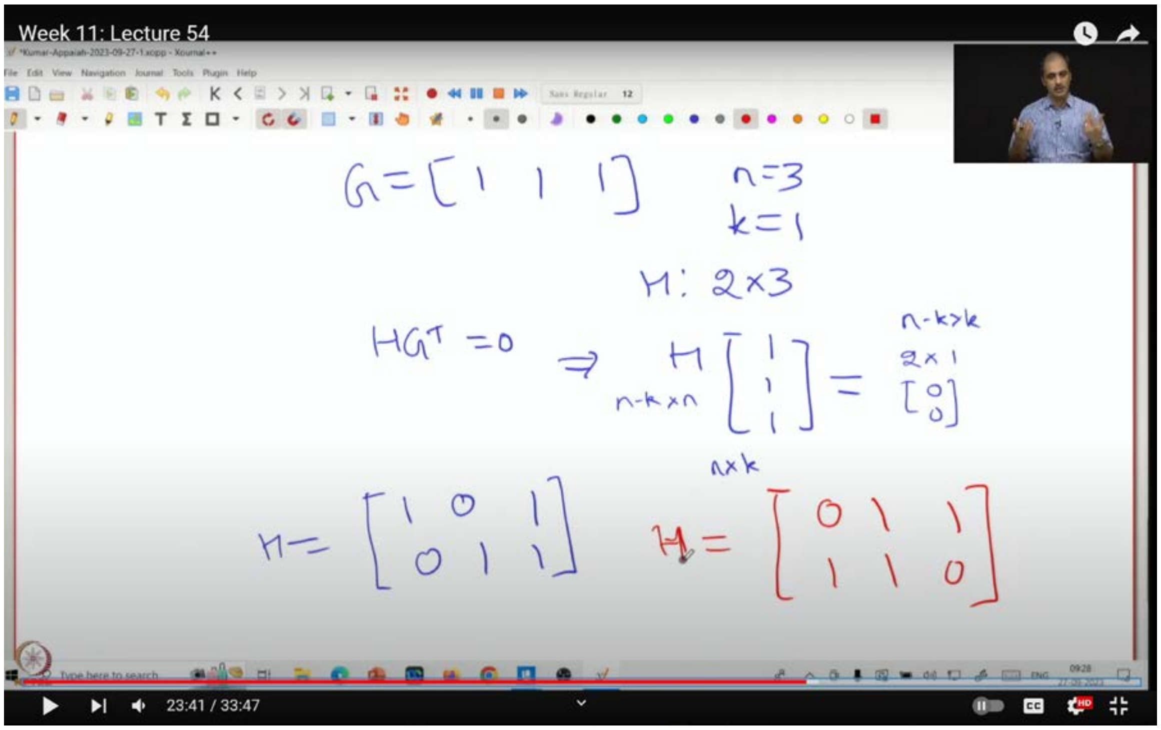
$$\underbrace{[1\ \ 1\ \ 1]}_{n-k\times n}\underbrace{\begin{bmatrix}1&0\\0&1\\1&1\end{bmatrix}}_{n\times k}=\overset{n-k}{\begin{bmatrix}0&0\end{bmatrix}}$$

Now, let's compare these results. We see that [0 0 0] consistently maps to [0 0 0], [1 1 1] maps to [1 1 1], [1 0 1] remains [1 0 1], and [0 1 1] maps to [0 1 1]. This illustrates that we have another equivalent generator matrix $\boldsymbol{G}$ for the code, albeit with a slight difference. The key difference lies in the mapping from the messages to the coded vectors. For instance, in one case, [0 1] maps to [0 1 1], while in another case, it maps to [1 1 1]. Similarly, [1 1] may map to [1 1 0] in one scenario and [0 1 1] in another. Despite these differences, in terms of error performance and other metrics, these two codes are equivalent. They represent the same code, just realized differently.

This is an important concept to understand. For example, someone might present a generator matrix for a 7, 4 Hamming code, and someone else might offer a different matrix for the same 7, 4 Hamming code. The mappings may differ, but the codes are fundamentally equivalent. The mapping between the messages and the coded vectors can be considered as a lookup table, which can always be adjusted as needed.

There are a couple of additional points to note. First, you can always swap the rows of the generator matrix $\boldsymbol{G}$ without affecting performance, though I won't prove that here. You can also swap the columns of $\boldsymbol{G}$ without losing any performance. Additionally, the zero vector is always a code word. When constructing a generator matrix, ensure you select linearly independent code words. Avoid including [0 0 0] as a row in $\boldsymbol{G}$, as it leads to trivial results and won't generate all the necessary code words.

(Refer Slide Time: 23:41)



$$G = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \qquad n=3, \quad k=1$$

$$H: 2\times 3$$

$$HG^T = 0 \Rightarrow \underset{n-k\times n}{H} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad n-k>k \quad 2\times 1 \quad n\times k$$

$$H = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

Next, let’s consider the 3, 1 repetition code, which is quite straightforward. Here, the generator matrix $\boldsymbol{G}$ is very simple. With a 1-bit repetition code, $\boldsymbol{m}$ is a 1-element vector. Consequently, $\boldsymbol{G}$ has only one row. This is the only $\boldsymbol{G}$ you can construct in this case because when you multiply $\boldsymbol{m}$ by $\boldsymbol{G}$, the result is either [0 0 0] or [1 1 1], depending on whether $\boldsymbol{m}$

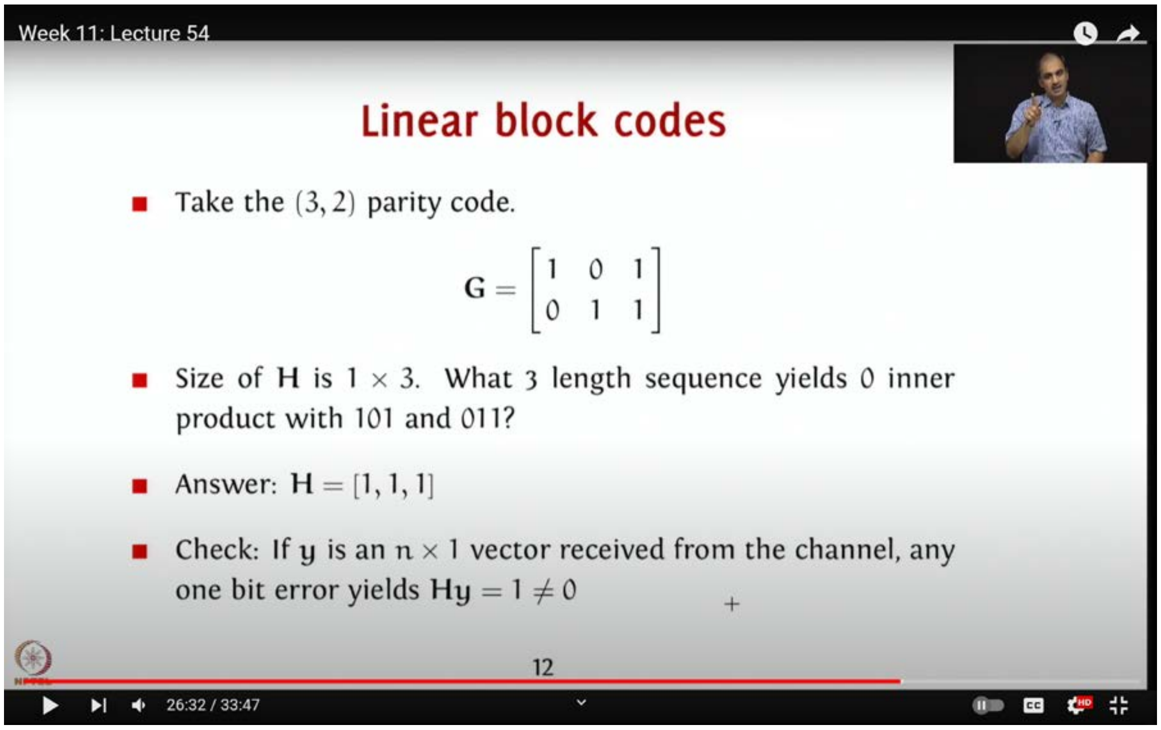
is 0 or 1.

So, for the 3, 1 repetition code, $\boldsymbol{G}$ is a row of three 1s. The code has $2^1 = 2$ code words, making this conceptually very simple. The exercise here is to generate all code words using this $\boldsymbol{G}$ and verify them, as we just did. Another interesting exercise is to enumerate all possible equivalent generator matrices for this linear block code.

Finally, how do we check if the channel output is a valid code word? One method is to evaluate all possible $m^T G$ combinations, and when the receiver gets an n-bit sequence, you can check whether it matches one of the code words. However, this approach is not efficient, especially for large k. Instead, due to the linear nature of the code, you can determine whether a sequence is a code word by performing a linear operation.

The key hint here is that since the code words were generated by taking linear combinations of the rows of $\boldsymbol{G}$, all you need to do is check if the received sequence is indeed a linear combination of those rows. This verification is done using a parity-check matrix, which serves as a pair to the generator matrix $\boldsymbol{G}$.

(Refer Slide Time: 26:32)



Linear block codes

- Take the $(3, 2)$ parity code.

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

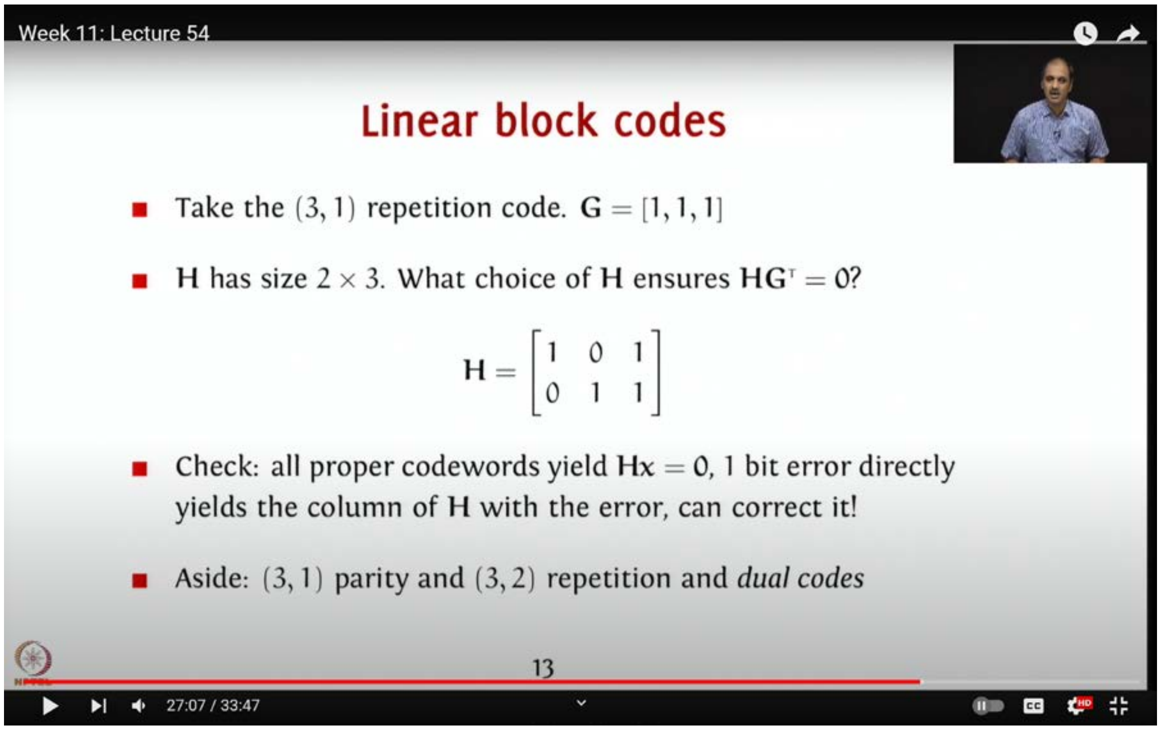
- Size of $\mathbf{H}$ is 1×3. What 3 length sequence yields 0 inner product with 101 and 011?
- Answer: $\mathbf{H} = [1, 1, 1]$
- Check: If y is an $n \times 1$ vector received from the channel, any one bit error yields $\mathbf{H}y = 1 \neq 0$

12

Whenever you generate code words using a generator matrix G, there is a corresponding matrix H that serves as a check to verify whether a given n-bit vector is indeed a valid code word for the code in question. Essentially, you can think of this as having a total of 2^n possible n-length sequences, out of which 2^k are valid code words. These 2^k valid code words form a vector space.

The remaining 2^{n-k} sequences are not valid code words. The matrix H can be used to determine if a sequence is a valid code word or not. Specifically, H is a matrix of size $(n-k) \times n$ such that Hm = 0 for all valid code words m. In other words, H is constructed so that it produces a zero inner product with all the valid code words.

(Refer Slide Time: 27:07)



Linear block codes

- Take the $(3, 1)$ repetition code. $\mathbf{G} = [1, 1, 1]$
- $\mathbf{H}$ has size 2×3. What choice of $\mathbf{H}$ ensures $\mathbf{HG}^{\mathsf{T}} = 0$?

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

- Check: all proper codewords yield $\mathbf{Hx} = 0$, 1 bit error directly yields the column of $\mathbf{H}$ with the error, can correct it!
- Aside: $(3, 1)$ parity and $(3, 2)$ repetition and *dual codes*

13

Another way to view this is that H multiplied by the transpose of G, $H \cdot G^T$, results in the zero matrix. But how does this work, and why is it effective? To gain an intuitive understanding, let's perform an exercise to guess the parity check matrices for both the repetition code and our parity check code.

Let's consider a specific case of a parity check code with parameters n = 3 and k = 2. The generator matrix G for this code could be something like this:

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Here, n = 3 and k = 2. The matrix H, which will have (n-k) rows and n columns, must satisfy the condition $H \cdot m = 0$ for all valid code words m. For example, H must satisfy:

$$H \cdot \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = 0, \quad H \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0, \quad H \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 0, \quad \text{and} \quad H \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = 0$$

The matrix H must be chosen to satisfy these equations. Let's assume H is a binary matrix with entries A, B, C. The first equation tells us A + C = 0 (in binary, this is equivalent to $A \oplus C = 0$). The second equation tells us B + C = 0, and the third tells us A + B = 0.

By guessing A, B, C, we can deduce the matrix H. If we guess $H = (1 \quad 1 \quad 1)$, this satisfies all the conditions:

$$1 + 1 = 0, \quad 1 + 1 = 0, \quad 1 + 1 = 0 \quad (in\ binary)$$

Thus, $H = (1 \quad 1 \quad 1)$ is the parity check matrix for the [3,2] code. You can verify that $H \cdot G^T = 0$, confirming that this indeed is the correct parity check matrix.

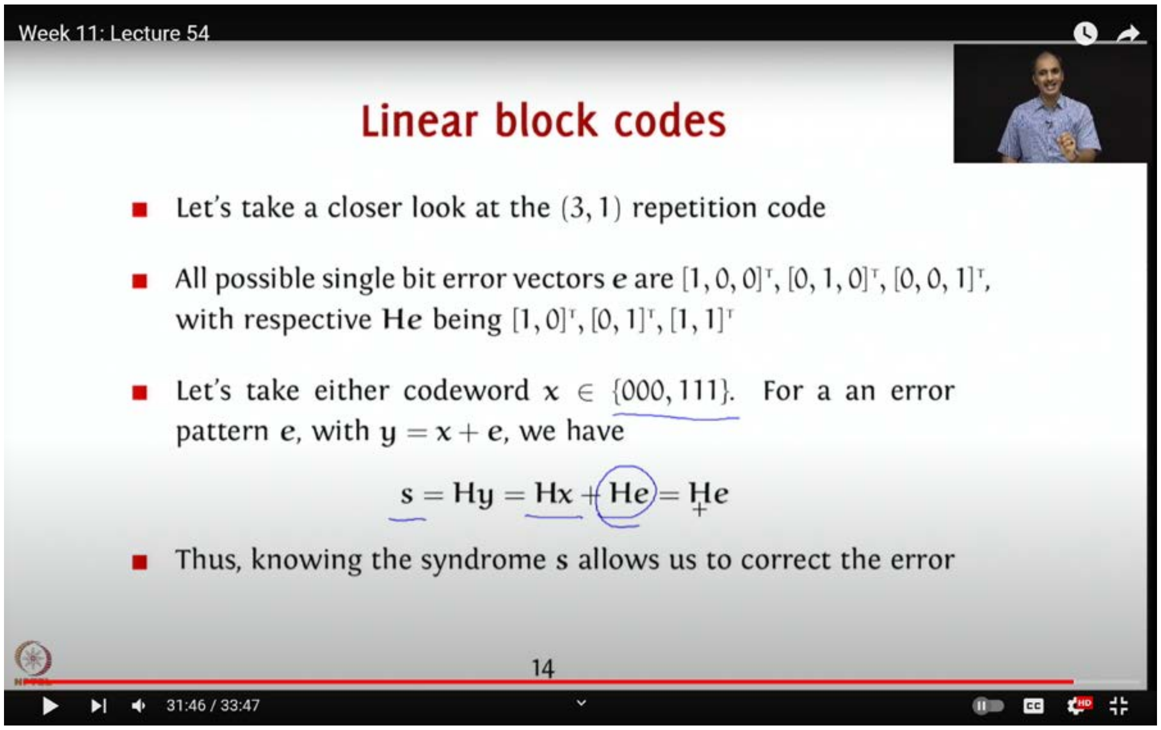
Now, let's extend this to the repetition code. Suppose we have n = 3 and k = 1. The corresponding matrix H must satisfy the condition $H \cdot G^T = 0$, where $G = (1 \quad 1 \quad 1)$. For this, H must be a $(n - 1) \times n$ binary matrix, i.e., 2×3.

You can either guess H or systematically solve for H. For example, choosing $H = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ satisfies $H \cdot G^T = 0$. This is not the only valid H; another valid choice is $H = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$. Both are equivalent and usable.

Ultimately, H is the parity check matrix for the code, and there can be multiple valid forms of H. The reason H is of size $(n - k) \times n$ is due to the vector space dimensions, where n-k represents the dimension of the space that does not contain valid code words. This concept is also tied to dual codes, a significant area of study in coding theory.

Therefore, constructing a parity check matrix H is a powerful tool that allows you to verify whether a received sequence is a valid code word without having to compare it against all possible code words $m \cdot G$. Instead, a linear operation using H suffices, making the verification process much more efficient.

(Refer Slide Time: 31:46)


Week 11: Lecture 54

Linear block codes

- Let's take a closer look at the (3, 1) repetition code
- All possible single bit error vectors $\mathbf{e}$ are $[1,0,0]^T, [0,1,0]^T, [0,0,1]^T$, with respective $\mathbf{He}$ being $[1,0]^T, [0,1]^T, [1,1]^T$
- Let's take either codeword $\mathbf{x} \in \underline{\{000, 111\}}$. For a an error pattern $\mathbf{e}$, with $\mathbf{y} = \mathbf{x} + \mathbf{e}$, we have

$$\underline{\mathbf{s}} = \mathbf{Hy} = \underline{\mathbf{Hx}} + \boxed{\mathbf{He}} = \mathbf{He}$$

- Thus, knowing the syndrome $\mathbf{s}$ allows us to correct the error

14

31:46 / 33:47

When dealing with a three-length sequence that results in a zero inner product, 101 and 011 both yield 111 as the solution. It's important to note that 000 is a trivial solution, and we should exclude it from consideration. Now, consider a scenario where y is an $n \times 1$ received vector from the channel. Here's where it gets interesting: any single bit error will lead to $H \times y$ equaling 1, which indicates an error since it's not zero.

Why does this happen? Recall that we chose H = [111], which means that the code should have even parity. If only one bit is flipped, whether it's 101, 011, 110, or 001, multiplying the received code word by H (i.e., 111) will result in 1. Thus, the parity check matrix (or the parity code) has the ability to detect a bit error by signaling a non-zero value when multiplied by the received code word.

This approach effectively detects errors, and here's the underlying intuition: you're essentially XORing the bits of the code and checking if the parity is zero. If the result is 1, you can conclude that a single-bit error has occurred. As mentioned in previous lectures, if there are two-bit errors or if all three bits are flipped, this method will fail. However, if only one bit is in error, the method will accurately detect it.

Now, let's consider the 3-1 repetition code we discussed earlier. The generator matrix for this code is 111, and this is one of the equivalent parity check matrices. You can verify that all proper code words satisfy $H \times x = 0$. If a single-bit error occurs, the corresponding column where the error took place is directly identified.

How does this work? Consider H = [101, 011]. If you post-multiply by any code word (e.g., 000 or 111), the results are telling. For instance, suppose you sent 000, but a bit error occurred. You would get a result like 01, which picks out the middle column. Reading 01 in binary gives you 1, indicating that the error occurred in the second bit, which indeed aligns with the actual error.

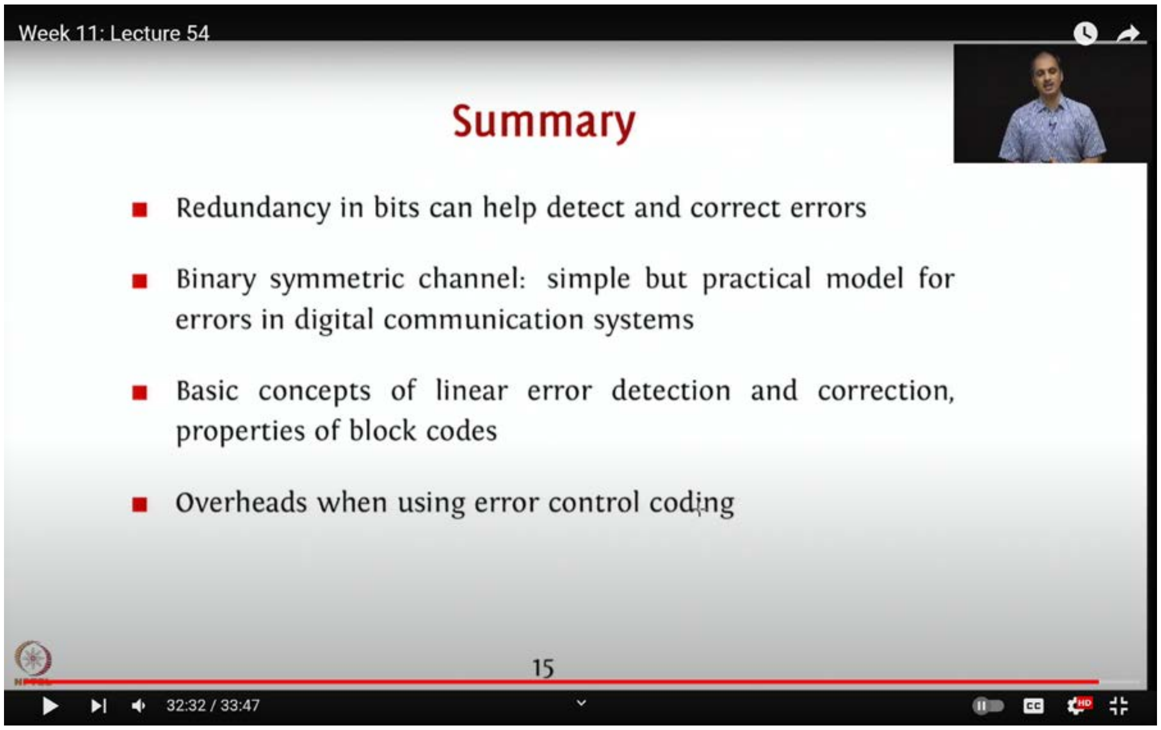
This isn't magic; it's a systematic process. If you sent 111 but the first bit was flipped, the result might initially confuse you. But once you correctly label the positions, say as 1, 2, and 3, the approach becomes clearer. For example, if you label the columns as 1, 2, and 3, and you receive 011, it picks out the sum of those two columns, resulting in 10. This indicates that the error occurred in the corresponding column, and flipping that bit will correct the code word.

Thus, using a repetition code's parity check matrix, you can identify the column where the error occurred simply by analyzing the result (e.g., 01) and then correcting the corresponding bit to obtain a valid code word. This method, though straightforward, is incredibly effective in detecting and correcting single-bit errors.

This is where a well-designed parity check code proves invaluable. When you design your parity check code with an effective parity check matrix, multiplying this matrix by the received vector will yield the parity. More specifically, it will identify the column or a combination of columns from the parity check matrix, pinpointing the exact location of the error. You can then flip the relevant bit to correct the code word, which helps you determine which k-bit message was originally sent. This method confirms that your repetition code can reliably correct a single-bit error.

As we have seen, with linear codes, by multiplying the received block of n bits by the parity check matrix H, you can easily verify whether you have a valid code word. If the result is not zero, it indicates an error. The non-zero result tells you the nature of the error, assuming that a detectable error has occurred. Let's delve into the 3-1 repetition code we discussed earlier. The single-bit error vectors are 100, 010, and 001, and the corresponding results of $H \times error$ are 1001 or 11, according to the parity check matrix we examined.

(Refer Slide Time: 32:32)



Summary

- Redundancy in bits can help detect and correct errors
- Binary symmetric channel: simple but practical model for errors in digital communication systems
- Basic concepts of linear error detection and correction, properties of block codes
- Overheads when using error control coding

15

Consider any code word X, which can be either 000 or 111, written as a column vector for convenience. For an error pattern E, if Y = X + E, then whether X is 000 or 111, $H \times Y$ always provides the correct location of the error. This is because S, defined as $H \times Y$, is also known as the syndrome. We can write $H \times Y$ as $H \times (X + E)$. Due to linearity, this simplifies to $H \times X + H \times E$. Since $H \times X$ is zero for a valid code word (i.e., $H \times 000$ or $H \times 111$ results in zero), we are left with $H \times E$.

Thus, for any linear block code, the multiplication of H by the received vector Y yields the same error pattern, regardless of the code word sent. In other words, whether you transmitted 000 or 111, if the error pattern is identical, $H \times Y$ will yield the same result. This means that the syndrome $H \times Y$ depends solely on the error pattern, not on the specific code word transmitted. This result will be crucial in our upcoming discussion on single-error-correcting codes, specifically Hamming codes, where we will use the syndrome to correct errors.

To summarize, in error-correcting codes, particularly block codes, adding redundancy in the bits helps in detecting and correcting errors. In this lecture, we focused on the binary symmetric channel, a simple yet practical model for managing errors in digital communication systems. We discussed binary phase shift keying (BPSK) over additive white Gaussian noise (AWGN) as a typical example of the binary symmetric channel.

We also covered the fundamental concepts of linear error detection and correction, providing basic examples of linear block codes used for error detection and correction. Remember, when expanding from k blocks to n blocks, you incur an overhead. Out of the n bits, only k are information bits, while n - k represent redundancy. The redundancy-to-total ratio $\frac{n-k}{n}$ represents the overhead, and the code rate is $\frac{k}{n}$. Ideally, you want this rate to be as high as possible, though redundancy comes with a cost. In the next lecture, we will continue our exploration of error control codes, with a detailed look at Hamming codes. Thank you.