Digital Communication using GNU Radio

Prof. Kumar Appaiah

Department of Electrical Engineering

Indian Institute of Technology Bombay

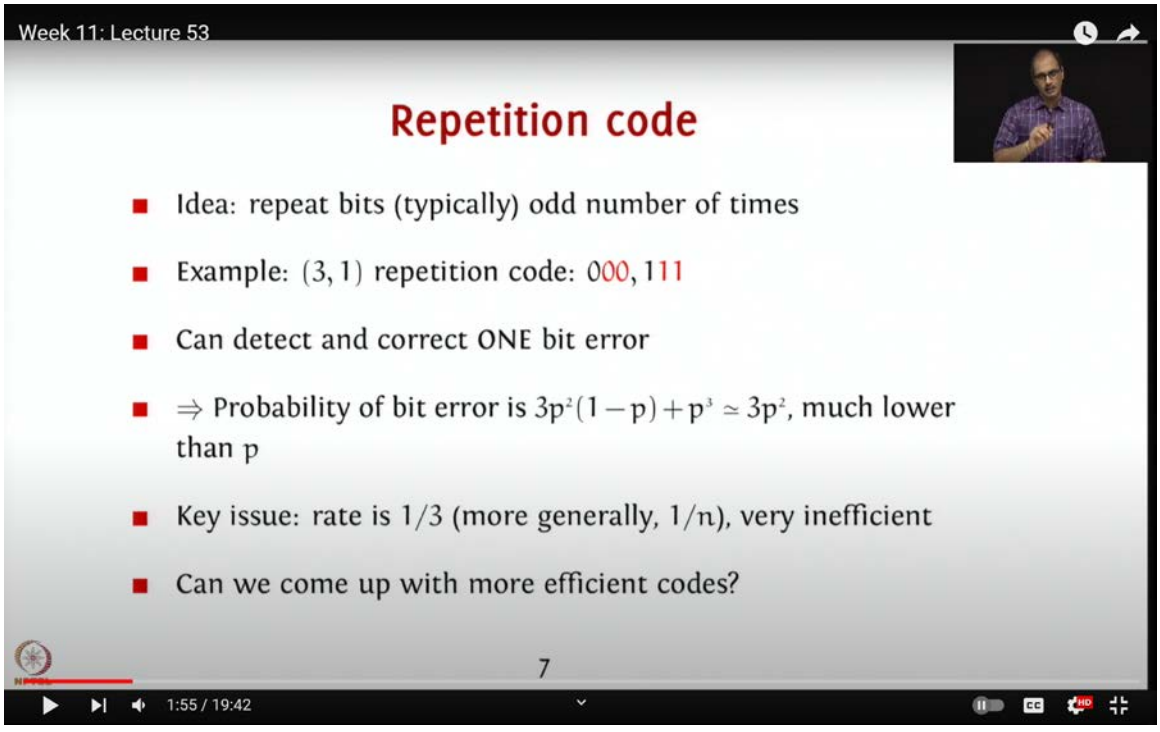
Week-11

Lecture-53

Error Control Coding: Repetition Codes

Welcome to this lecture on Digital Communication Using GNU Radio. My name is Kumar Appiah, and I am part of the Department of Electrical Engineering at IIT Bombay. In today's session, we will continue our exploration of error control codes, specifically focusing on block codes. In our previous discussion, we covered parity check codes. To recap, in a parity check code, we add a single bit of overhead to create an even parity code. This means that after appending the redundant bit, the total number of 1s in the codeword is even. Alternatively, from an XOR perspective, if you XOR all the bits in the codeword, the result will be 0.

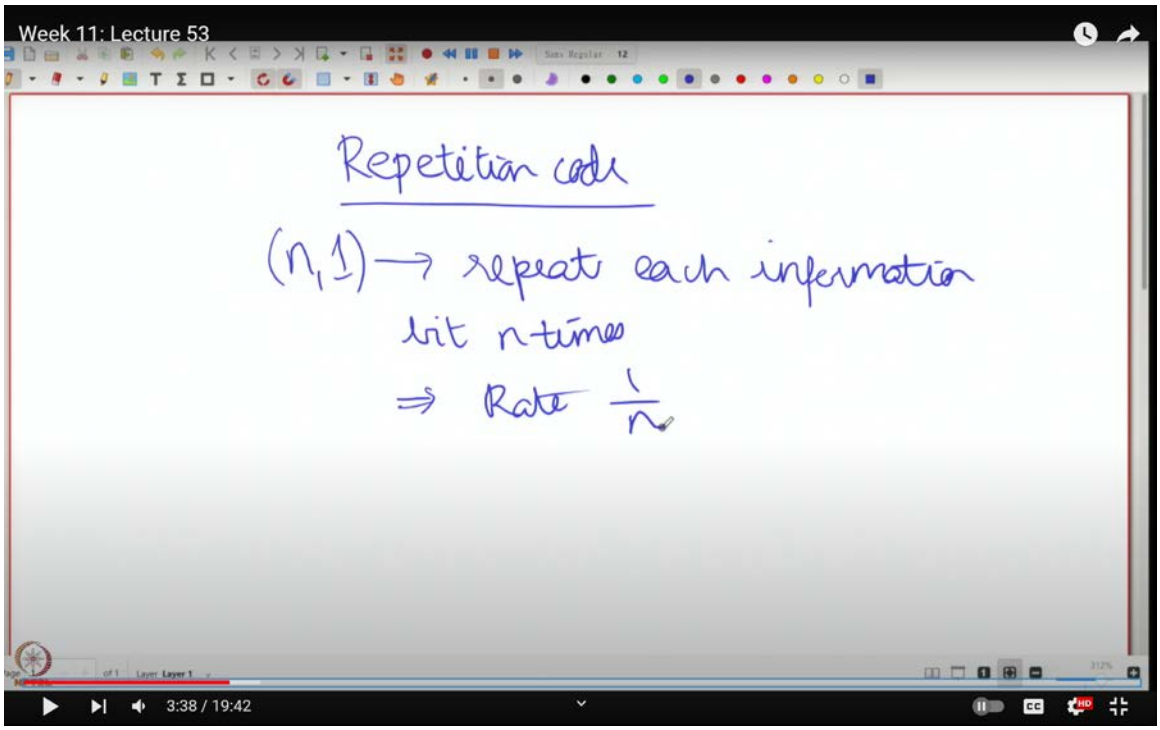
(Refer Slide Time: 01:55)



We learned that with parity check codes, we can detect the presence of no errors or a single error, and in the case of a single error, we can discard that erroneous code block. However, when two or more errors occur, the code fails, leading to potential mistakes. Today, we will shift our focus to a different class of codes known as repetition codes.

We will specifically consider odd repetition codes, where each bit is repeated an odd number of times. Although it is possible to repeat bits an even number of times, we will focus on odd repetitions because they allow for simple majority logic decoding at the receiver, as we will demonstrate shortly.

(Refer Slide Time: 03:38)

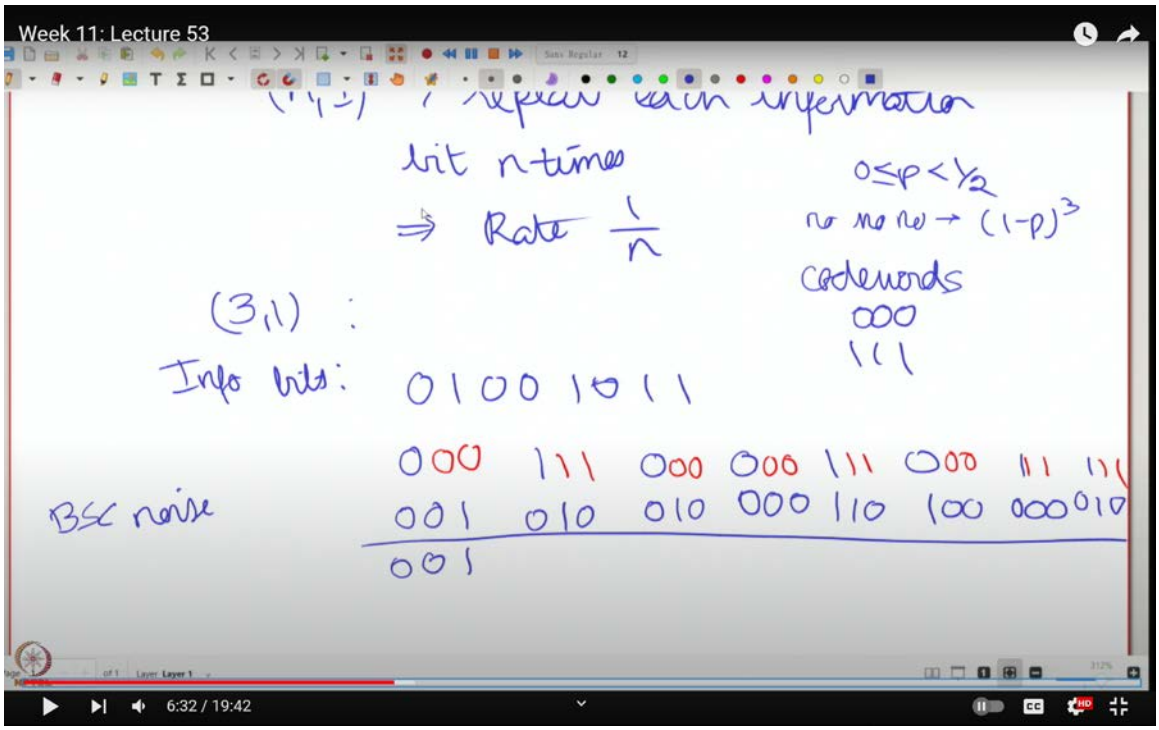


Let's start with the simplest example, a 3-1 repetition code. In this code, the number of coded bits is 3 for every single information bit. This means that each information bit is repeated three times, resulting in a significant reduction in the bit rate—essentially down to one-third of the original rate. Despite being quite inefficient in terms of bit rate, this code allows for the detection and correction of single-bit errors.

Unlike the parity check code, repetition codes allow you not only to detect a single-bit error

but also to correct it. How does this work? Let’s delve into that with a simple discussion. Typically, a repetition code is denoted as an n-1 code, where each information bit is repeated n times. The rate of the code is 1/n, meaning that for every n bits transmitted, only 1 bit carries the original information, while the rest is redundancy or overhead, calculated as (n-1)/n.

(Refer Slide Time: 06:32)



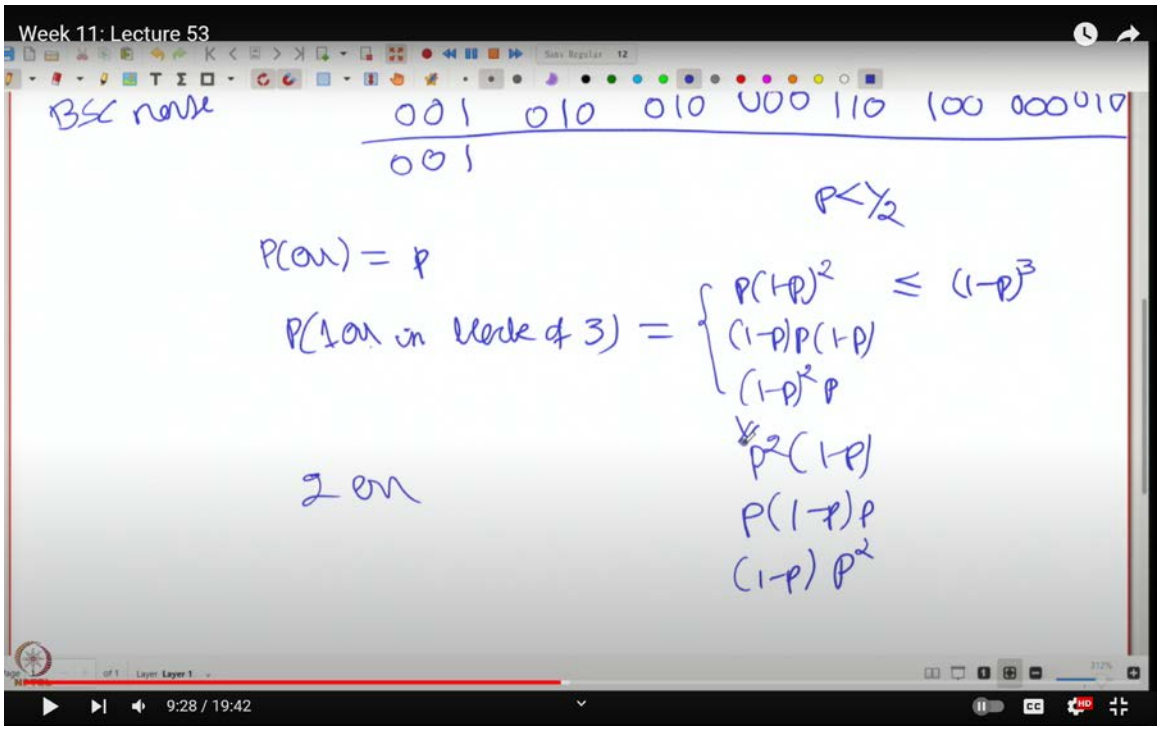
Let's take the example of a 3-1 repetition code. To illustrate this, consider the following sequence of information bits: 0, 1, 0, 0, 1, 0, 1, 1. Since repetition codes extend the bit length, the sequence after applying the 3-1 repetition code would be: 000, 111, 000, 000, 111, 000, 111, 111.

Now, let’s introduce noise using a binary symmetric channel (BSC), where we flip each bit with a probability p. For example, consider the noise sequence: 001, 010, 000, 000, 010, 000, 110, 100.

Given that this is a block code, we analyze each block of 3 bits to make a decision. The method involves comparing the received sequence against the possible codewords, which

in this case are 000 and 111. If we receive a block like 001, it's clear that something went wrong because it's neither 000 nor 111.

(Refer Slide Time: 09:28)



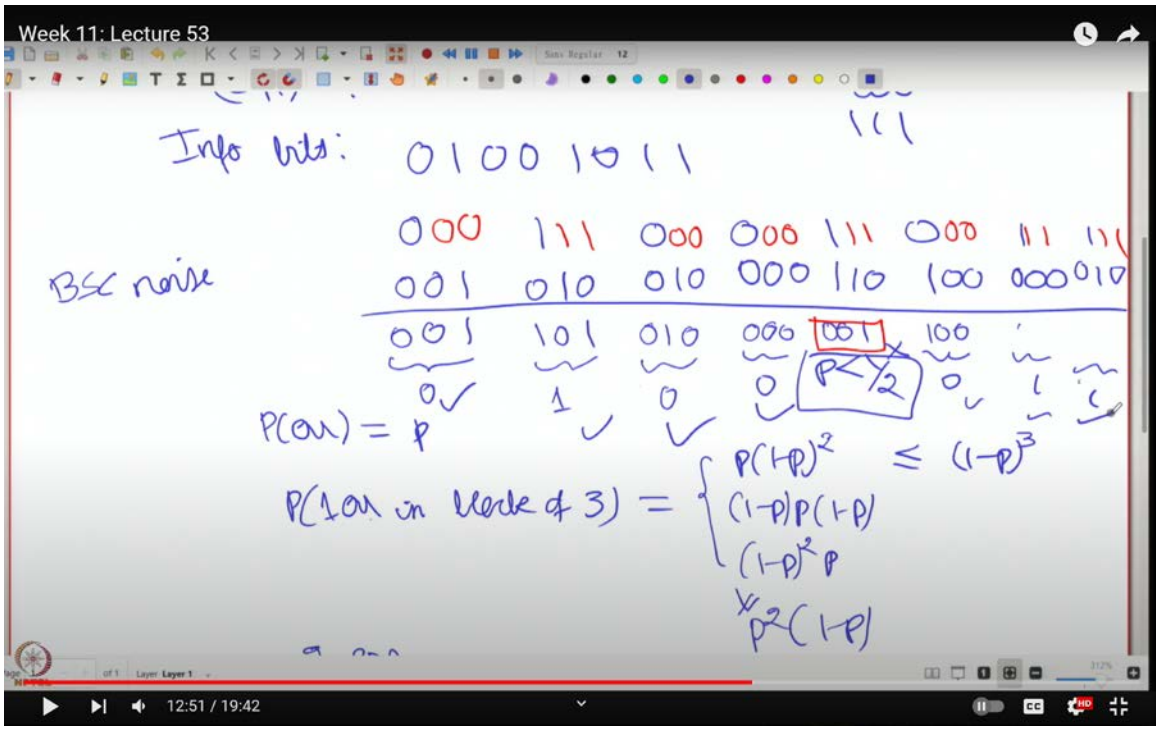
The most likely scenario is that no error has occurred, as the probability of no errors happening in all three bits is $(1-p)^3$, which is the highest when p is less than 0.5. If any error does occur, it's most likely to be a single-bit error, which has a probability of $p \times (1-p)^2$. This is clearly smaller than $(1-p)^3$, confirming that a single-bit error is less likely than no errors at all.

To formalize this, if we want to calculate the probability of exactly one error occurring in a block of 3 bits, we need to consider the three possible cases: an error in the first bit, the second bit, or the third bit. Each of these events has a probability of $p \times (1-p)^2$, and since there are three possible single-bit errors, the total probability for a single error is $3 \times p \times (1-p)^2$.

Now, compare this with the probability of no errors, $(1-p)^3$. Since p is less than 0.5, $(1-p)^2$ is greater than p, making $(1-p)^3$ larger than $p \times (1-p)^2$, hence confirming that no error is

the most likely scenario, followed by a single-bit error.

(Refer Slide Time: 12:51)



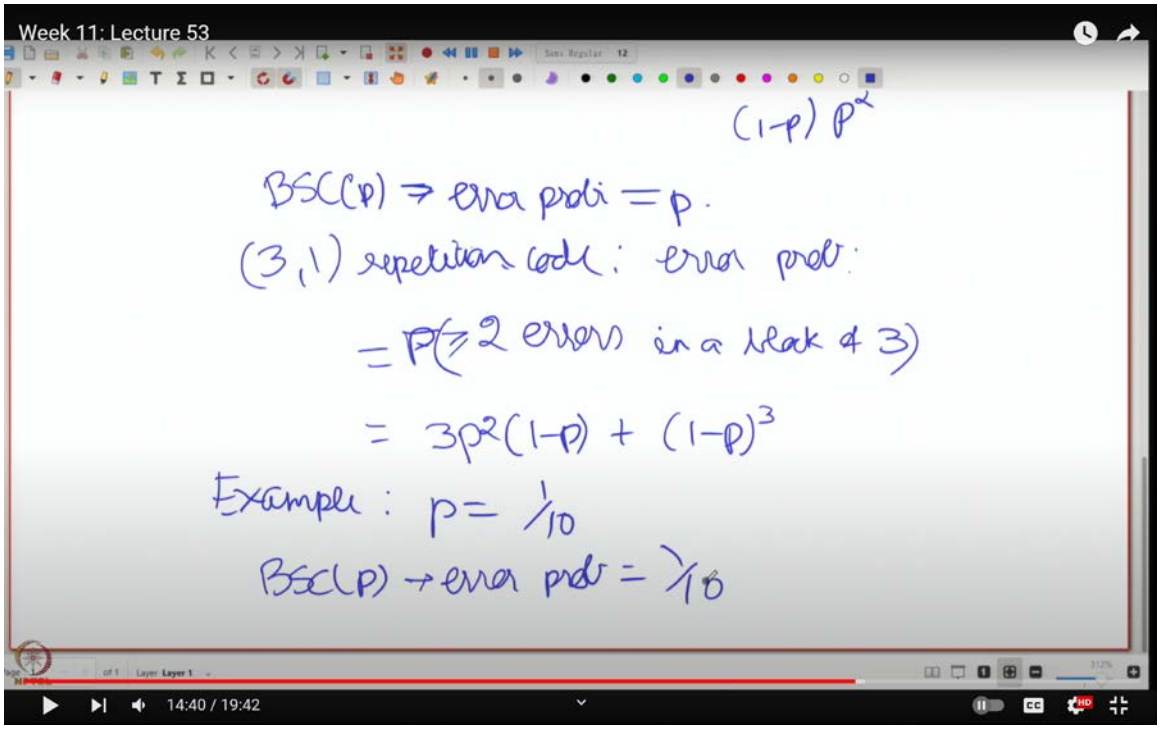
This discussion highlights the effectiveness of repetition codes in correcting single-bit errors by leveraging the probability characteristics of the BSC. While repetition codes reduce bit rates, they offer a straightforward method for error detection and correction, particularly in scenarios with low error probabilities.

This means that if you want to identify the maximum likelihood event, the most likely scenario is one where no error has occurred. Therefore, the optimal approach is to assume no errors initially. However, when you receive a sequence like 001, the most probable conclusion is that a single bit error has occurred. While it's possible to consider other events, the most likely scenario is indeed the occurrence of a single error.

Now, let's break this down further. Suppose you receive the sequence 001. There are multiple ways this could have happened. For instance, you might have originally sent 111, with the first and second bits flipping to 0. This situation corresponds to two errors. On the other hand, if you sent 000 and only the last bit flipped, that would indicate a single error.

When we compare these two possibilities, the scenario involving a single error is more probable. Mathematically, the probability of two errors occurring is $p^2 \times (1-p)$, and this is even smaller than the probability of one error occurring, which is $p \times (1-p)^2$. Since the probability p is less than 0.5, the likelihood of a single error is higher than that of two errors.

(Refer Slide Time: 14:40)



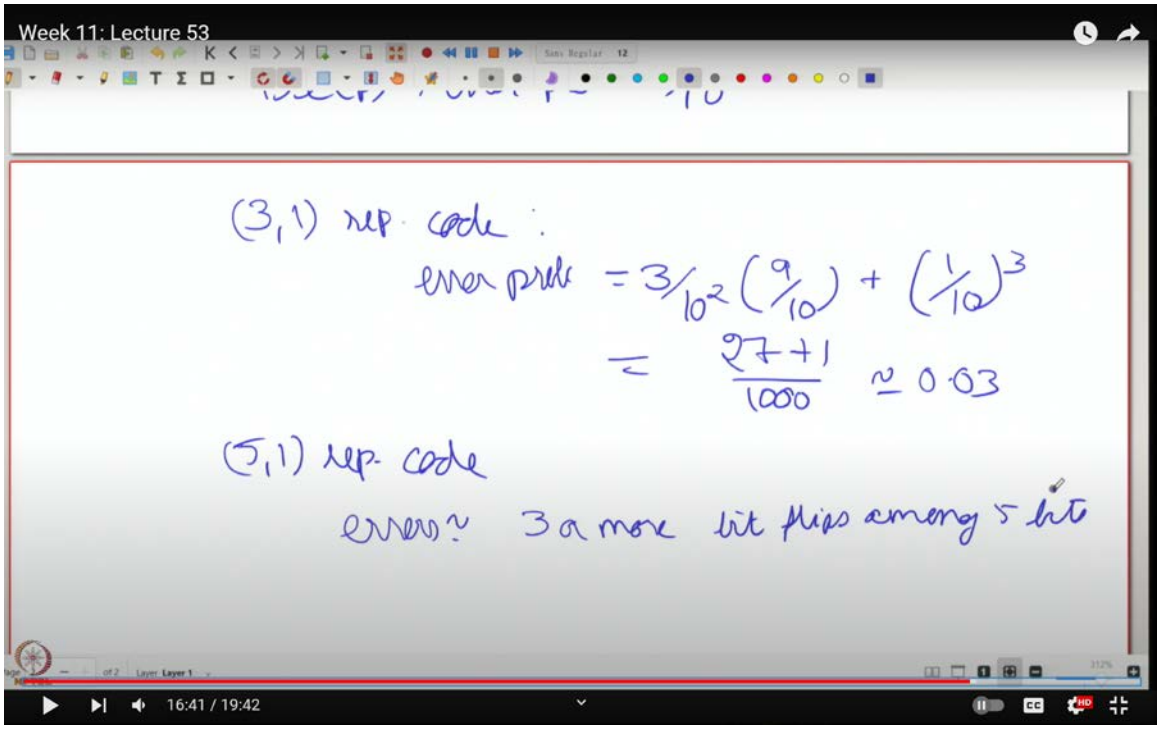
In other words, in the situation where you sent 000 but received 001, it's more likely that only the last bit flipped, resulting in a single error. Conversely, if you sent 111 and received 001, it would require two bits to flip, which is less probable. Therefore, by applying maximum likelihood detection, we would correctly deduce that 000 was sent, and thus the information bit is 0.

The conclusion isn't drawn just because the majority of the bits are 0, but because it's statistically more likely. Now, consider the situation where you receive 101. Again, this is not a valid codeword, as the only valid codewords are 000 and 111. There are two possible scenarios: either 000 was sent and both the first and last bits flipped (which has a

probability of $p^2 \times (1-p)$), or only the middle bit flipped (which has a probability of $p \times (1-p)^2$). Given that p is less than 0.5, the second scenario is more likely, meaning 111 was sent, and the majority logic would deduce that 1 was sent.

Now, let's consider another example where you receive 010. This situation is straightforward, with no errors, so you would directly conclude the information bit is 0. However, an interesting case arises when you send 111, and the noise pattern results in 110, where the first two bits flip, leading to 001. Here, maximum likelihood detection suggests assuming the minimum number of bit flips, which would mean concluding that 000 was sent. Unfortunately, this is a mistake because, in reality, 111 was sent, but two bits flipped.

(Refer Slide Time: 16:41)

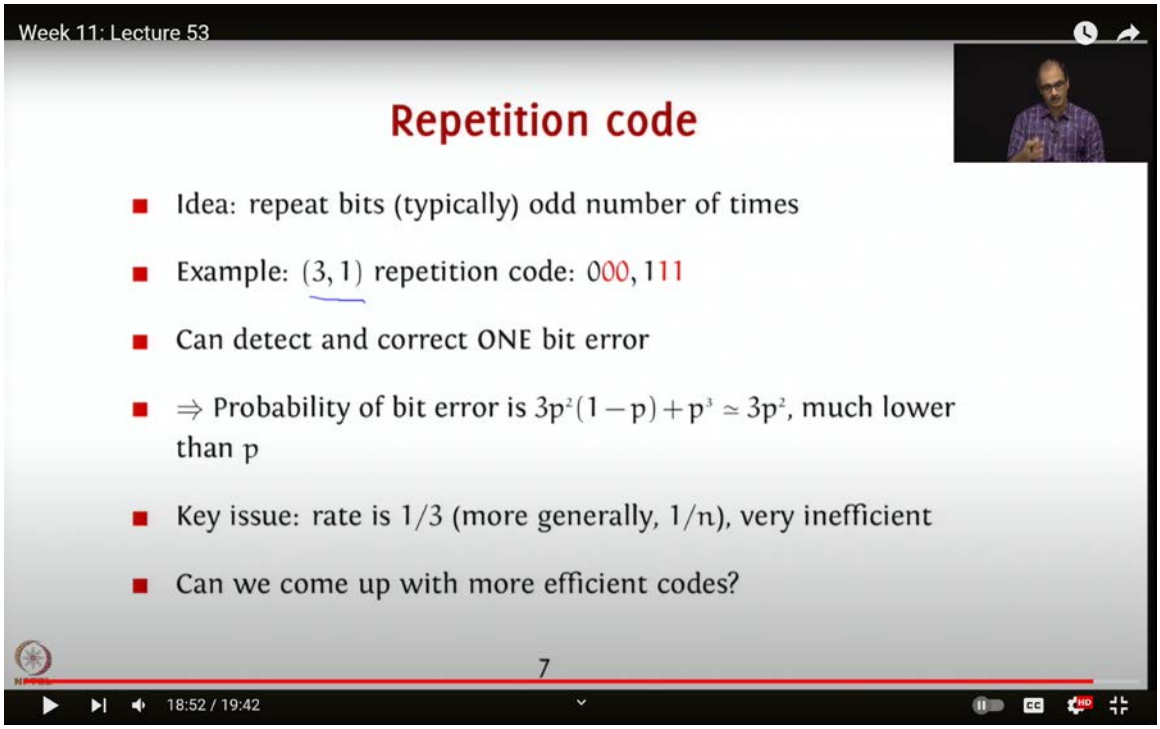


This demonstrates the limitation of repetition codes. When more than one bit flips in a block of three, errors occur. For example, if you compare the original bitstream with the received stream, you might find that one bit was flipped incorrectly while the rest were correct. Errors arise when two or more bits flip in a block of three bits.

Comparing this with an uncoded Binary Symmetric Channel (BSC), where the error probability per bit is p, using a 3-1 repetition code reduces the error probability. For the 3-1 code, an error occurs only if two or more bits flip in the same block. The probability of exactly two errors is $3p^2 \times (1-p)$, and the probability of three errors is $(1\text{-}p)^3$.

Let's calculate this with an example: suppose p = 0.1, then the error probability in a BSC is 0.1. For the 3-1 repetition code, the error probability becomes $3p^2 \times (1-p) + p^3$. Substituting p = 0.1, this gives approximately 0.03, which is significantly lower than the original 0.1 error probability.

(Refer Slide Time: 18:52)



If you extend this to a 5-1 repetition code, the scenario changes. Here, majority logic is applied over 5 bits, and errors occur with 3 or more bit flips among the 5 coded bits. The error probability is further reduced because the leading term involves p^3, which is even smaller. With repetition codes, as p decreases, using these codes significantly reduces the error probability.

However, there is a trade-off. The rate for an n-1 repetition code is 1/n, making it

inefficient. While you gain the ability to correct more errors (e.g., one error for 3-1, two errors for 5-1, three errors for 7-1), the rate decreases progressively. Despite its inefficiency, repetition codes are beneficial in scenarios where reliability is paramount and simple decoding logic is preferred. At the receiver, decoding is as simple as applying majority logic.

Imagine you have a block of n bits, where n is an odd number. To determine the original data, you simply count the number of zeros and ones in the block and conclude that the majority represents the correct bit. This method highlights the usefulness of repetition codes, which are straightforward to implement. However, the trade-off is a significant reduction in data rate. In extreme cases where the signal-to-noise ratio is poor and the error probability p is high (close to 0.5), repetition codes can be particularly valuable. But in more practical scenarios, there are more efficient coding techniques available.

In the upcoming lectures, we will explore these more efficient codes. Specifically, we will introduce the concept of linear block codes and then move on to discuss Hamming codes. Thank you.