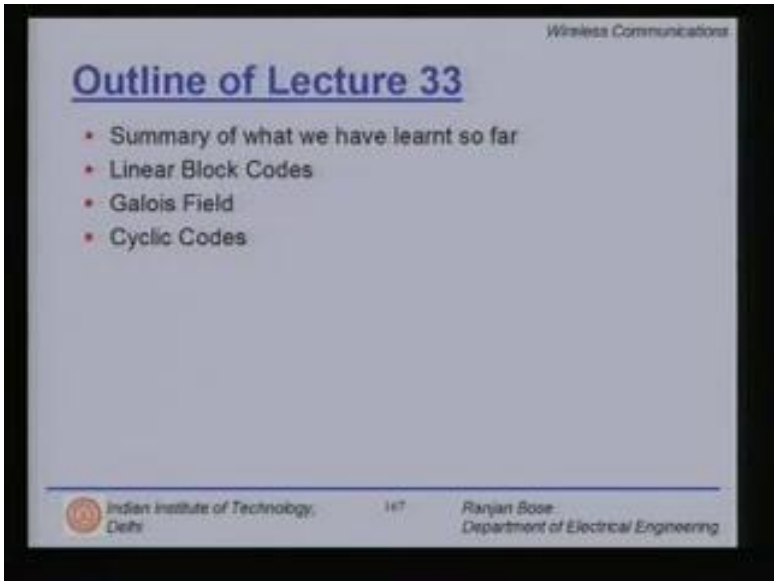
Wireless Communications
Dr. Ranjan Bose
Department of Electrical Engineering
Indian Institute of Technology, Delhi
Lecture No. # 33
Coding Techniques for Mobile Communications

Welcome to the next lecture on wireless communications. Today we will talk about coding techniques for mobile communications, let us have the brief outline for today's talk.

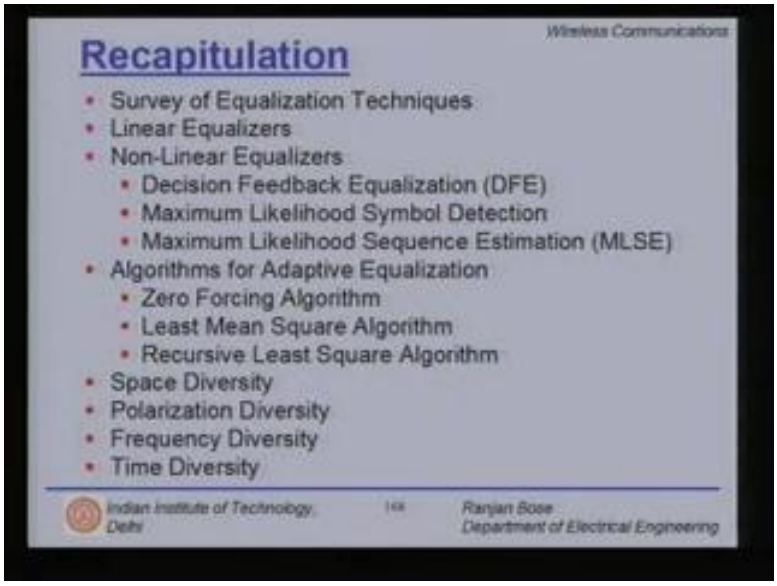
(Refer Slide Time: 00:01:29 min)



We will briefly summarize what we have learnt so far followed by brief description of linear block codes. We will then look at Galois field theory and then have a small introduction to cyclic codes so this is the outline for today's talk. First a brief recap, we have previously looked at various kinds of equalization techniques for mitigating the effects of multipath fading. We looked at linear equalizers followed by set of nonlinear equalizers specifically we talked about decision feedback equalizer, maximum likelihood symbol detection, maximum likelihood sequence estimation. Then we moved over to the various algorithms for adaptive equalization namely the zero forcing, the least mean square and the recursive least square algorithms.

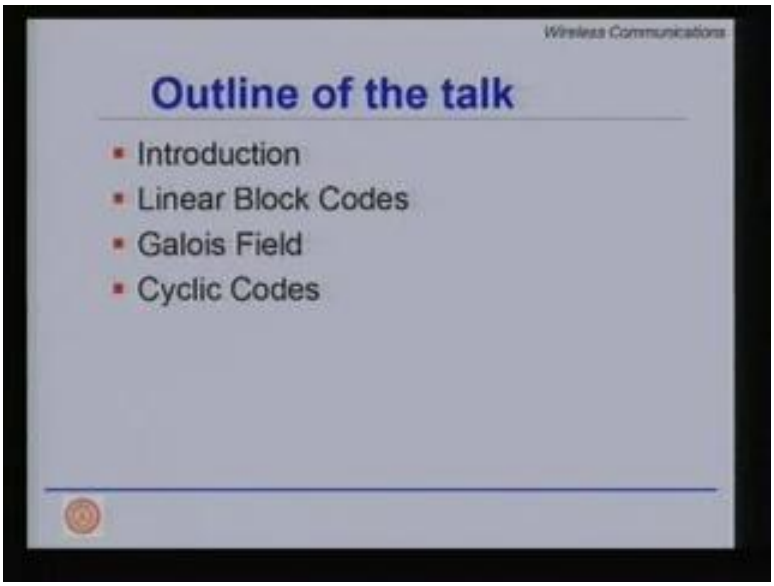
We then looked at various kinds of diversity techniques space diversity, polarization diversity, frequency diversity and finally time diversity. So we realized that out of the three important techniques to overcome the effects of bad fading channels the two are equalization and diversity. The third one is coding specifically channel coding. Today we would like to have a brief outlay of the channel coding techniques for mobile communications.

(Refer Slide Time: 00:01:49 min)



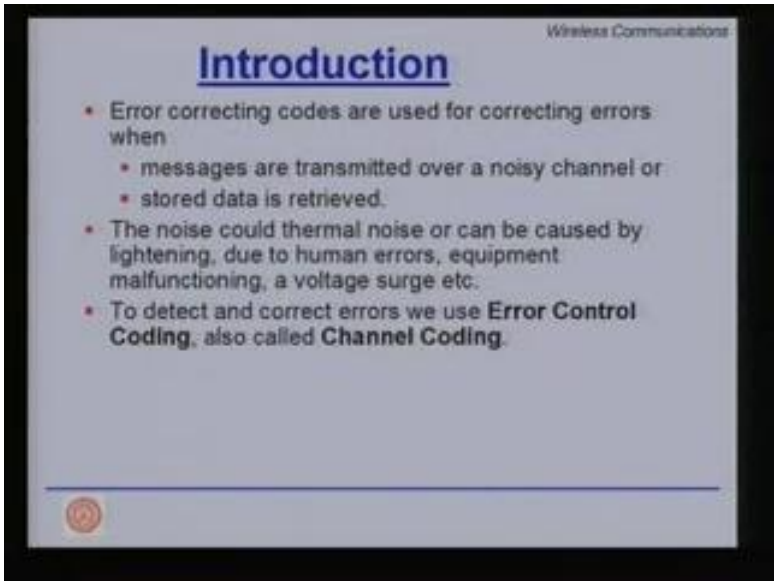
So within this description we will talk about liner block codes, Galois field and cyclic codes.

(Refer Slide Time: 00:03:16 min)



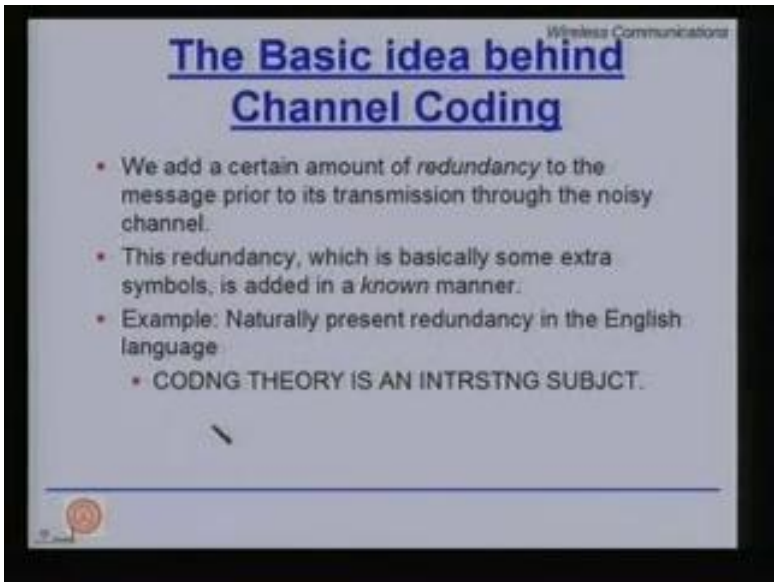
Now what are error control codes or error correcting codes? Error correcting codes are used for correcting errors when the messages are transmitted over a noisy channel or stored data is retrieved. We know that error control coding is used for mobile communications but if you store your data on CD there also we use error control coding. The noise that we are talking out is could be thermal noise or it can be caused by lightening human errors equipment malfunction, voltage surge or any other thing. So any of these things can cause noise, on top of that if you are working in a fading channel you will get burst errors when you pass through long durations of fade.

(Refer Slide Time: 00:03:20 min)



To detect and correct errors we use error control coding which is also known as channel coding.

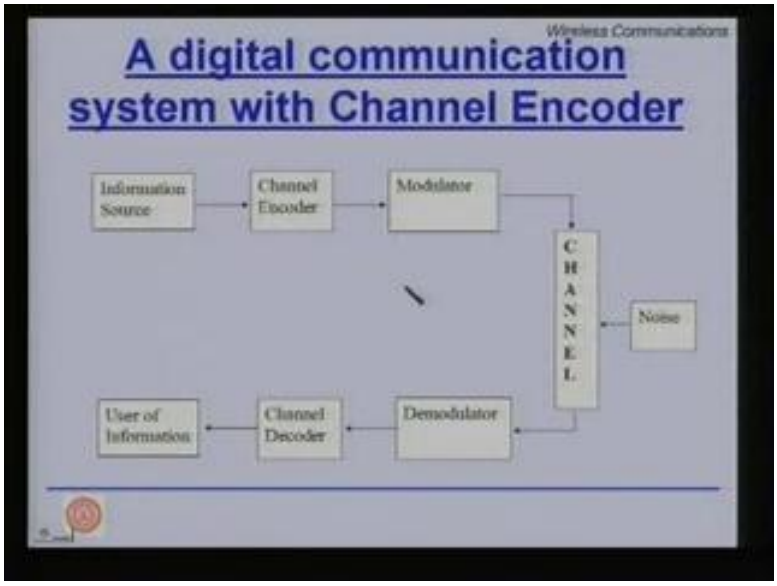
(Refer Slide Time: 00:04:23 min)



Now what is the basic idea behind channel coding? The basic idea is to add a certain amount of redundancy to the message prior to its transmission through the noisy channel. However this redundancy is added in a known fashion. This redundancy which is basically some extra symbols either said is always added in a known fashion which is known to us the transmitter and the receiver and is unknown to the noise clearly. Let us look at an example, a lot of redundancy is present naturally in the English language for example if you read the following sentence.

It has a lot of errors and missing letters in it. However it is not very difficult to understand and read the sentence coding theory is an interesting subject. So we have automatically corrected for errors by virtue of knowing that the English language has lot of redundancy built into it. Your coding theory which deals with not letters but bits works along similar lines, we have the redundancy so we can reconstruct that is detect and then probably correct the errors once we received an erroneous message signal.

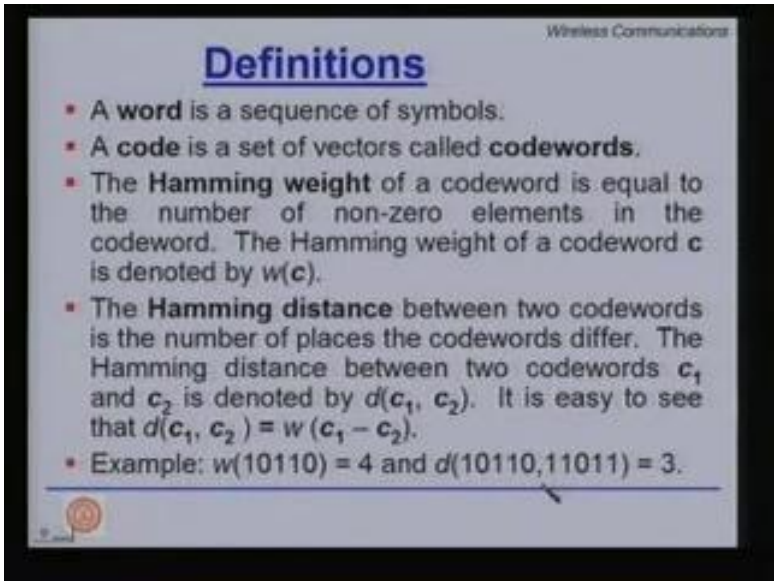
(Refer Slide Time: 00:05:51 min)



Now let us look at the block diagram of a digital communication system with a channel encoder and decoder. So you have an information source normally it is followed by source coder we have taken it out from the diagram because we want to focus on the channel encoder followed by the modulator. The signal is then pass through the channel where noise is added and in our case this channel would most likely be a fading channel. We have the demodulator followed by the channel decoder and then the user of the information.

So this is the model that we will work with. In most cases the channel coder and the modulator are two separate blocks. In subsequent lectures we will see that in some cases it is possible to combine the channel coding block and the modulator together to talk about coded modulation, example Trellis coded modulation.

(Refer Slide Time: 00:06:59 min)

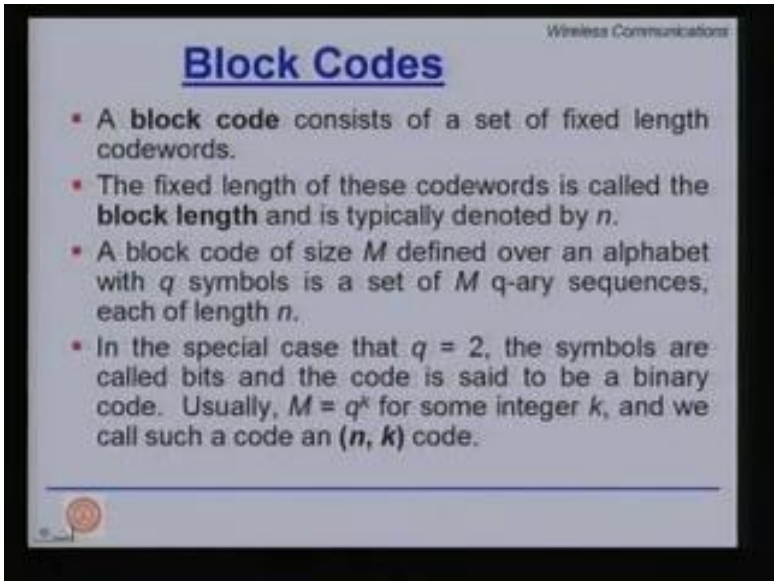


Now let us define certain terms that we will use throughout our lectures. Firstly a word is a sequence of symbols. A code is a set of vectors called codewords. The hamming weight of a codeword is equal to the number of non-zero elements in the codeword. So it's just a number, the hamming weight of a codeword c is denoted by w (c) is called the hamming weight or simply the weight of a codeword. The other term is hamming distance, the hamming distance between two codewords is the number of places the codewords differ. Again the hamming distance is again a number. The hamming distance between two codewords c_1 and c_2 is denoted by distance c_1, c_2.

It is easy to see the distance between c_1 and c_2 is equal to the weight of the codeword $c_1 - c_2$. Clearly hamming weight and hamming distance have something to do with finding out how similar or how different are two codewords. If the hamming distance is more they are different, if the hamming distance is less they are more similar to each other. Let us look at an example, let us took at the word 1 0 1 1 0, the weight of this is 1 2 3 because there are only 3 non-zero elements here the other two are zero. So it should be 3 here and the distance between these 2 is the number of places they are different it is different at number 2 and then it is different at number 3 and then it is different at number 5 so it is different at three places so it should be 3 which is correct, for the weight is 3 and the distance is also 3.

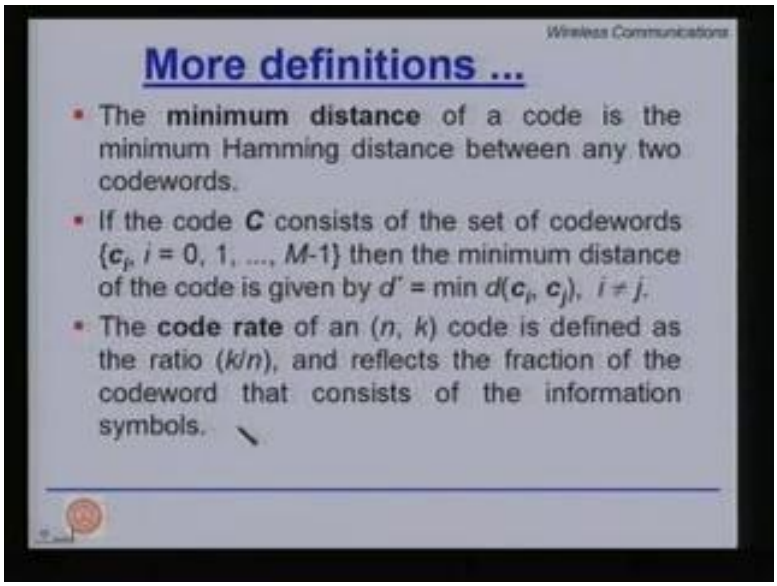
Now let us come to a special class of error correcting codes called block codes. A block code consists of a set of fixed length codewords. Hence the name block codes, the block length is fixed. The fixed length of these code words is called the block length and is typically denoted by small n, lower case n. A block code of size M defined over an alphabet with q symbols is a set of M q-ary sequences each of length n. Please note for binary case q will be 2 but that doesn't mean that block codes necessarily have to be binary, you can have non-binary block codes as well.

(Refer Slide Time: 00:09:14 min)



For a special case at q is equal 2 that is binary the symbols are called bits and the code is said to be a binary code. Usually M is equal to q^k for some integer k and we call such a code an n, k code. So basically what a block code does is takes k bits and transforms them into n bits where n is larger than k thereby adding n-k redundancy. How you add those redundancy makes the difference between a good code and a not so good code.

(Refer Slide Time: 00:10:54 min)

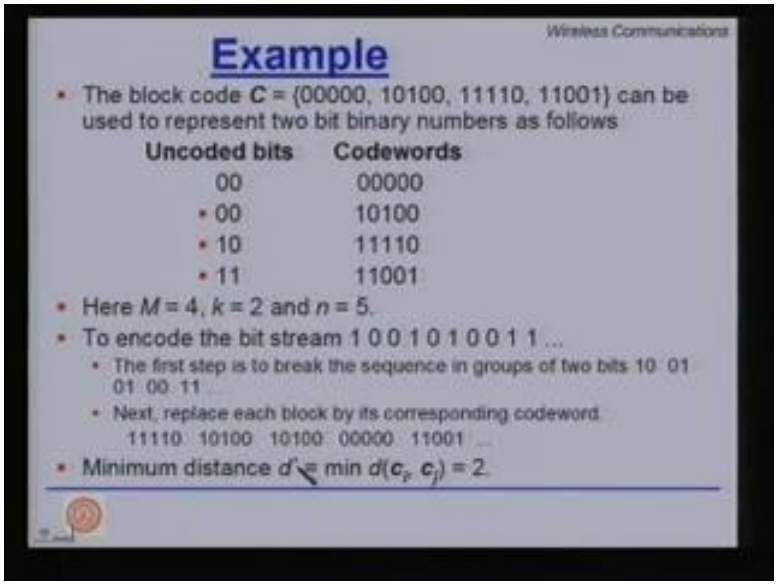


Let's go by some more definitions, the minimum distance of a code is the minimum hamming distance between any two codewords.

So as we know a code is a set of codewords. Now amongst this set of codewords if you compare two codewords at a time some of them will have a greater hamming distance than other pairs. The pair which gives you the minimum distance is called the minimum hamming distance of the code. If the code C consists of a set of codewords C_i where i = 0 1 through M-1. Then the minimum distance of the code is given by d^* and is equal to minimum of distance of c_i, c_j where i is not equal to j. So different pairs of the codewords within the code the minimum distance of that is d^*. Now another term which has something to do with efficiency of the code is called the code rate.

The code rate of an n, k code is defined as the ratio so it's a ratio, it has no units its k over n and reflects the fraction of the codeword that consists of the information symbols. So again the objective of a code which is n, k code is to take k bits at a time and converted into n bits. Now k bits are the information bits and n bits are the codeword bits so k over n which is definitely a fraction because n is larger than k because you add n - k redundant bits, k over n is the code rate. It should be less than or equal to one, equal to one means you are not doing any coding at all. The closer k by n is to one the more efficient is the code.

(Refer Slide Time: 00:13:07 min)

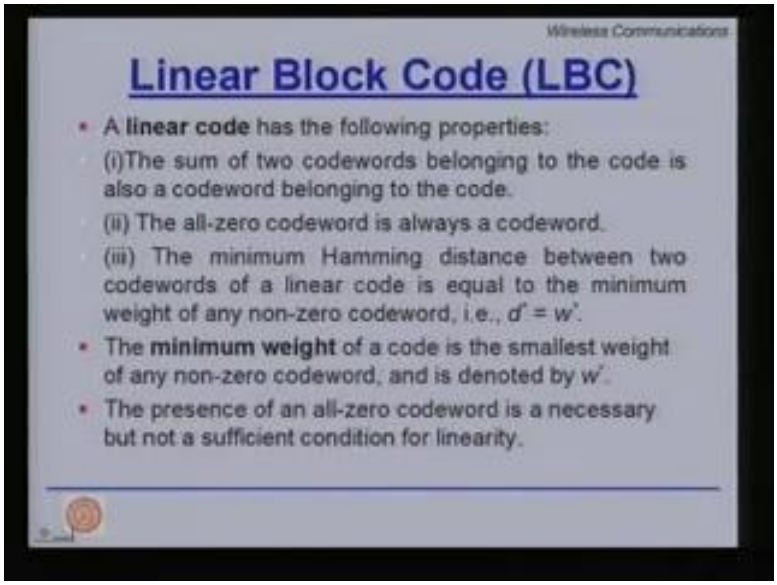


Let us look at an example. Let us look at a simple block code which has 4 codewords. Now look at this, the block code C can be used to represent two binary numbers as follows. So we have uncoded bits and their corresponding codewords as follows. So if we have 0 0 you represent it like this, if you get 0 1 you represent it like this, 1 0 you represent it like this and 1 1 you represent it like this. So for 4 possible two bit inputs you have 4 possible outputs, here M is equal to 4, k is equal to 2 and n is equal to 5, n is the length of the codewords. Note n is 5 and is equal to the block length of the code, each of the codewords is of length 5. Now suppose you have an incoming bit stream 1 0 0 1 0 1 0 0 1 1 and so and so forth.

Now suppose our job is to encode this raw bit stream coming in, using the following code. So first step is should break up the sequence in groups of two bits because we can only take two bits at a time and assign a codeword to it. So if you break it up into two bits it will look like 1 0 first two bits then 0 1 and then 0 1 and then 0 0 and then 1 1 and so and so forth. Now for 1 0 you replace 1 1 1 1 0 for 0 1 you replace 1 0 1 0 1 and so on so forth and so you get the following codewords. Clearly for a much shorter input bit stream you get a much longer output bit stream, the code rate of this will be 2 over 5 k over n. So for every two information bits it is putting 3 redundant bits to form 5 codeword bits.

The minimum distance of this code is a minimum distance between any two codewords and here if you look at it the first codeword and second codeword differ only at two places and if you can carry on this comparison because there are four choose two possible pairs which is 6, if we look at all the 6 distances we will find that two is the minimum distance possible. Hence d^* is equal to 2. We will learn soon the d^* has something to do with the number of errors the code can detect and correct. So in some sense d star tells you how strong the code is in terms of error correcting capability.

(Refer Slide Time: 00:16:22 min)

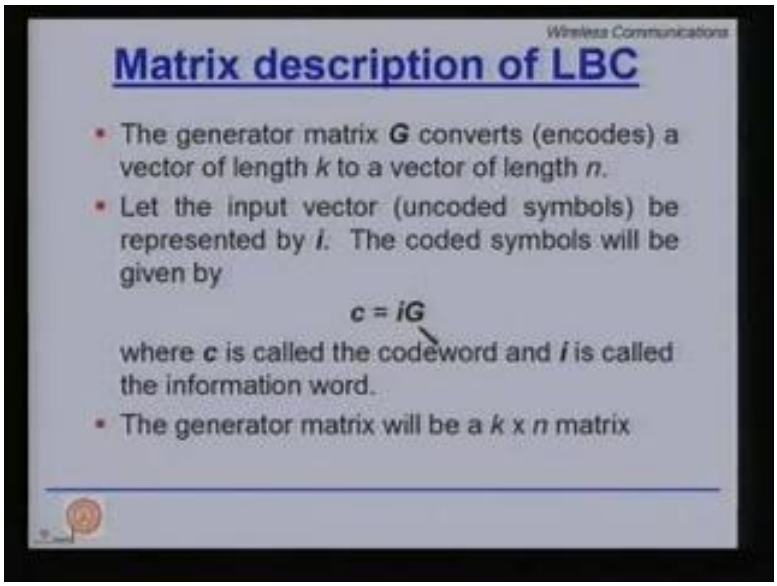


Now let us look at a subclass of block codes called linear block codes. Now what is the philosophy? The basic philosophy is to add more and more structure in our codes because what noise does is breaks the structure and if we have a good structure in place these are mostly algebraic structures then we will able to detect the missing links and hopefully correct the errors. So the first constraint that we put on block codes and constraints means more structure is linearity and we define something called as linear block codes. A linear code must have the following properties. The sum of two codewords belonging to the code is also a valid codeword belonging to the code. So sum of two codewords is also a valid codeword and of course the all zero codeword is always a valid codeword.

If these two conditions are satisfied then you can say that the block code is also a linear block code and the other property that comes is that the minimum hamming distance between any two codewords of a linear code is also equal to the minimum weight of any non-zero codewords. That is you don't have to compare all the pairs and find the distance between them, you used compute the hamming weight of each of the codewords except the non-zero codeword and whichever is the minimum you take it and it will be the w^* equal to d^*.

So the minimum weight of a linear code is a smallest weight of any non-zero codeword and is denoted by w^*. Please note that the presence of an all zero codeword is a necessary however not a sufficient condition for linearity. So the first thing that we must check if you are asked to verify whether the block code is linear or not is the presence of the all zero codeword. If it is present we go further otherwise we say look this cannot be a linear block code. Now it is very easy and efficient to define linear block codes by matrices.

(Refer Slide Time: 00:19:09 min)

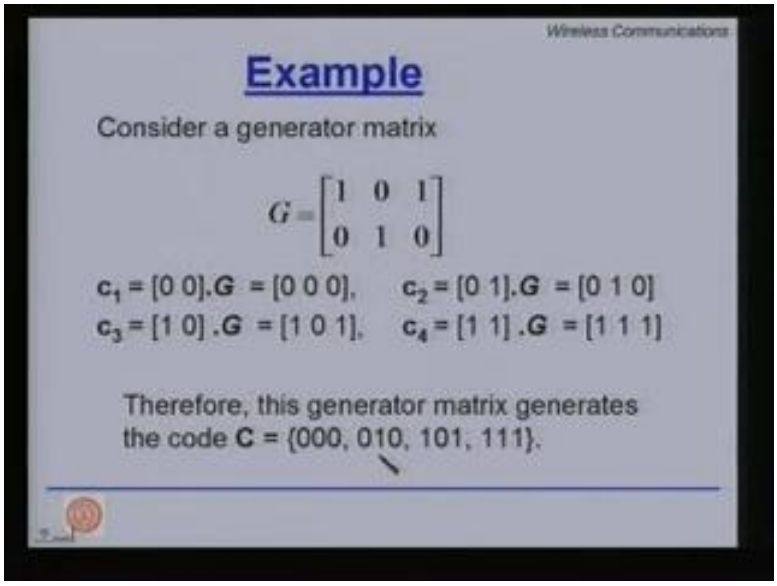


So far we have seen that linear block code is nothing but a table which is a look up table. So you have input bit streams and for any combination of k input bit streams you have a combination of n codeword bits. It is difficult, lengthy and more time consuming to store and recall your linear block codes in terms of look up tables is much more efficient to implement them as matrices. How do we do that? So we have the matrix description of linear block codes. We define the generator matrix G which converts or it encodes a vector of length k to a vector of length n. If we can have this magical matrix G which can take a vector of length k and make it a vector of length n and do it correctly for all possible k bits then we have a generator matrix which can generate the entire code hence the name generator matrix.

Let the input vector which is the uncoded symbols we represent it by vector i so bold face represents it's a vector. In that case the coded symbols will be given simply by c is equal to i times G. So c is a one cross n vector, i is a 1 cross k vector and G is a k by n matrix. So if you multiply 1 by k with k by n you get 1 by n which is c.

Here c is the codeword and i is the information word, the generator matrix clearly will be a k by n matrix. So if G can truly generate your entire code then you only need to save and store k into n bits and define the whole codeword. So a lookup table which would have been very memory expensive would not be required to store in that fashion we can just store the generator matrix.

(Refer Slide Time: 00:21:38 min)

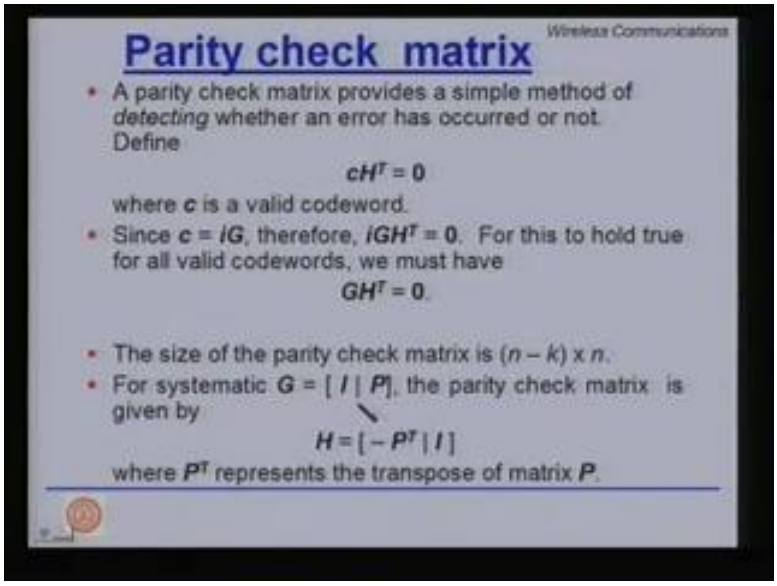


Let us look at an example of a generator matrix. Consider the following generator matrix G the first row 1 0 1 second row 0 1 0. It is k by n 2 by 3 matrix so this should be able to take two bits at a time and converted into 3 bits. So if you have the following representation, c_1 if you take 0 0 and multiply it with the generator matrix you will get c_1 equal to 0 0 times G is equal to 0 0 0. If you take the other two bit input 0 1 and multiplied with G you get 0 1 0 and so and so forth. So for the four possible 0 0, 0 1, 1 0 and 1 1 inputs which are two bit inputs you have the 4 possible outputs.

Therefore this generator matrix generates the code 0 0 0, 0 1 0, 1 0 1 and 1 1 1. Since it is a linear block code if you are asked to find the minimum distance of this code it will be the minimum weight of this code which is the number of non-zero elements in any of the non-zero vectors either here or here and clearly this one has unity one so the minimum distance d^* is 1.

Now we define a dual of the generator matrix called the parity check matrix. It has some interesting properties, a parity check matrix provides a simple method of detective whether an error has occurred or not. So the first job is to detect whether an error has occurred and then the second step is to see whether you can really correct the error. So the job of the parity check matrix is to detect whether an error has occurred. Let us define a matrix H such that codeword times H^T is equal to the zero matrix.

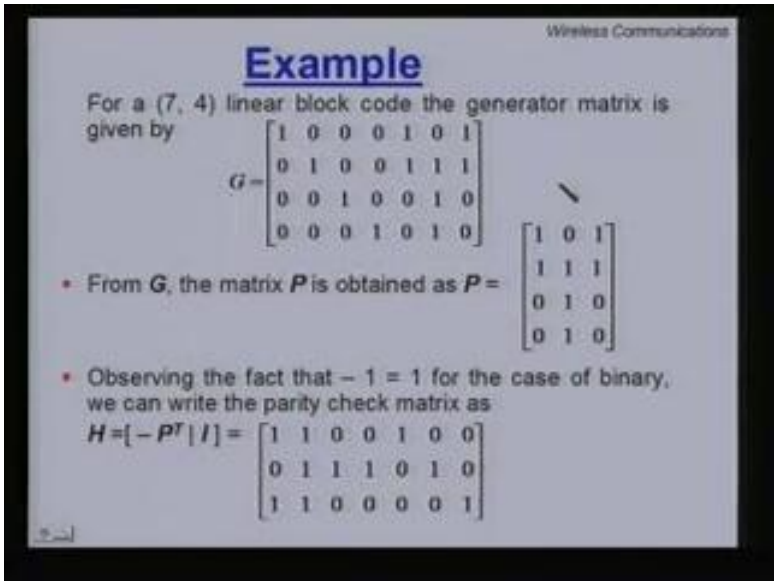
(Refer Slide Time: 00:23:22 min)



This H is your parity check matrix, so for any valid codeword if you multiply it with the transpose of the parity check matrix you will get a zero matrix. This will actually be a vector 1 times n- k so if it is not zero then clearly c is not a valid codeword and hence you have detected an error because coding is about transmitting and receiving valid codewords. Since c is equal to i times G by definition therefore we can write i GH^T is equal to zero. For this to whole true for all valid codewords that is i can be anything we must have GH transpose is equal to zero which means if you have the G matrix which is your generator matrix you can find out the H matrix and vice versa.

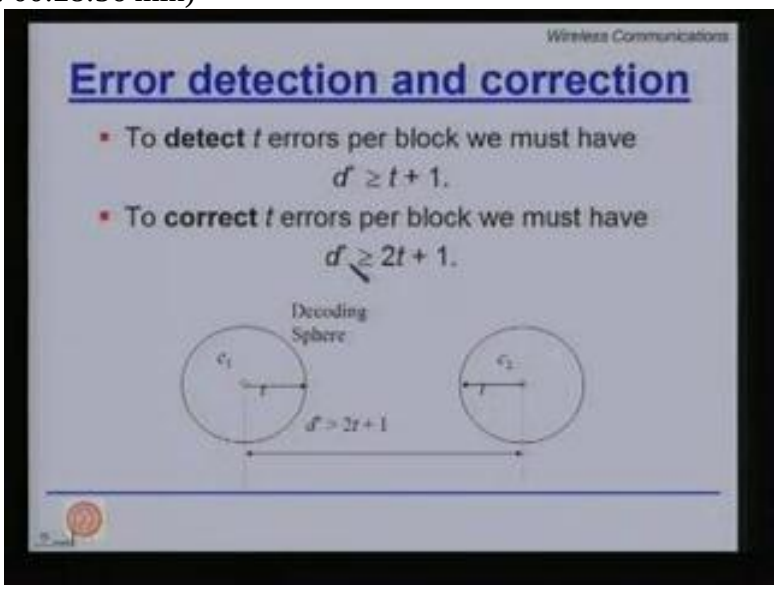
Please note that for any G matrix, H matrix may not be unique that is you can have several H matrices which satisfy this condition. No problem, any one of those H matrices would be able to work with your cH transpose equal to zero and will be able to tell you whether an error has occurred or not. The size of the parity check matrix is n - k times n, so this is the size of the parity check matrix and for any systematic G. So we define a systematic generator matrix G as one which can be represented in a matrix were the first part is a k by k identity matrix followed by k by n - k parity matrix. So if your G has been defined as I partitioned P then H matrix is simply minus P transpose partition I here I is an identity matrix of the size n – k. Here I is the identity matrix of size k, PT represents the transpose of the matrix P so this tells us one way to generate an H matrix from a G matrix and vice versa. If you have the G matrix or the H matrix you uniquely define the code.

(Refer Slide Time: 00:27:12 min)



Let us look at an example. So for a 7, 4 linear block code the generator matrix is given by the following equation. Now please note the first k rows and k columns form an identity matrix here, so this is of the type I partition P. So P matrix is k by n - k so your P matrix is just the last 3 columns of the G matrix here because it's already in the systematic form. Now for a binary case minus one is equal to one because 1+1 is zero binary addition is a modular to arithmetic. So for the case of binary we can write the parity check matrix as again you will have the last 3 columns the identity matrix and this is minus P transpose. So take this P and just transpose it, minus P transposes P transpose and you have a G matrix and H matrix. So GH transpose will be the zero vector, here is H is equal to minus P transpose I. Given this you can get a G, given G you can get the H, this format of G is called the systematic generator matrix.

(Refer Slide Time: 00:28:56 min)



Now let us come to the objective for which the error control codes were designed for error detection and correction so the objective is first to see an error has happened at all or not and then if you have detected that an error has happened then can we correct it. If so how many errors can be correct. So to detect t errors per block we must have d^* which is the minimum distance of the code greater than equal to t +1. What does it mean? It means that if my t is less than d^* -1, I take this on the other side of the unique quality then I can still detect the error which means that if the number of errors is d^* then the problem is that one of the two codewords which form the pair for minimum distance might get changed into the other codeword if you have d^* errors.

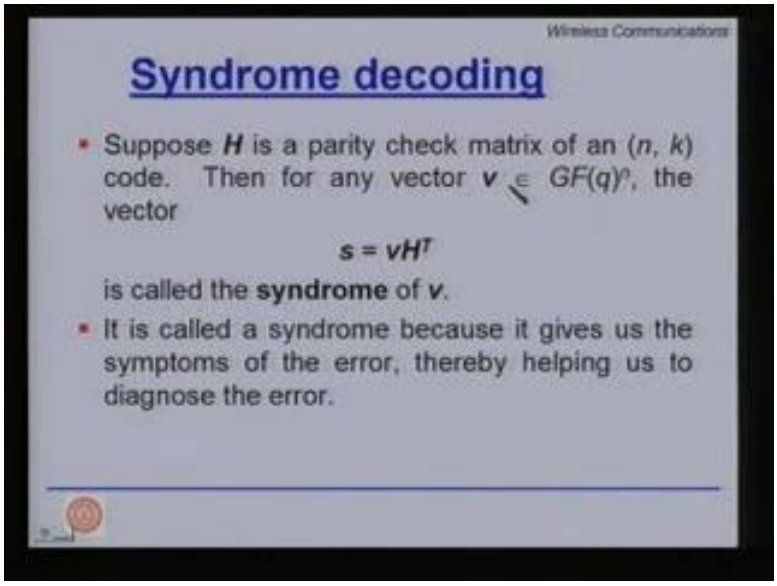
Any errors fewer than d^* can be detected because it will not make any codeword into another codeword. It takes d^* changes since the hamming, minimum hamming it is d^* it requires d^* changes of one codeword to go and transform itself into another codeword. So d^* -1 is a maximum errors that can be permitted so that our detection algorithm can work. So in order to have a detection for t errors we must have the condition d^* greater than or equal to t +1. To correct t errors per block so we are talking about block codes, we now want to correct t errors we must have the condition d^* greater than or equal to 2t +1. Now why is that? Consider the following diagram, we have the codeword c_1 here and codeword c_2 here and they are separated by the minimum distance d^* is a distance notion so we have separated them here spatially.

Now if an error happens, if the number of error is one we will move one unit away from the codeword because now the hamming distance will be one, if they are two it will move further away three, four and so and so forth until we have d^* because d^* steps will make c_2 look like c_1 in the worst case. Why? They are only different at d^* places because the distance between these two is d^*. I am talking about the minimum distance pair codewords. So it takes d^* errors to make c_2 appear as c_1 and vice versa. Now let's say it is a decoding sphere because it is distant from c_3, c_4, c_5, other codewords also so you can define a sphere it is called the decoding sphere. If t errors make this sphere boundary here and t's errors make it here and these two circumferences just touch each other. Then we have the condition that 2 t is equal to d^*, in that case the spheres will not intersect that means t errors.

If any of the t errors happen then they can be decoded back to c_2, if any of the t errors happen the erroneous codeword could be mapped back to c_1 here. So it is any codeword or any word appearing within the decoding sphere of c_1 will be decoded as c_1, any erroneous word occurring within the sphere of c_2 can be mapped back to c_2. There is no ambiguity, only in the case when the spheres intersect we will have an ambiguity and we will not be able to correctly decode. So in order to ensure that t errors can definitely be corrected, I must ensure that the spheres do not intersect.

Now the minimum condition required is d^* is equal to 2t and there also we have this condition that if an erroneous word is exactly t from this one and t from this one I cannot make a decision. So I put a safeguard and I say d^* should be greater than or equal to 2t +1. So if you have this condition that d star exceeds to t +1, t errors can be corrected as well. So it clearly tells us that in order to have your block code correcting more number of errors you have to increase the distance.

(Refer slide Time: 00:34:30 min)

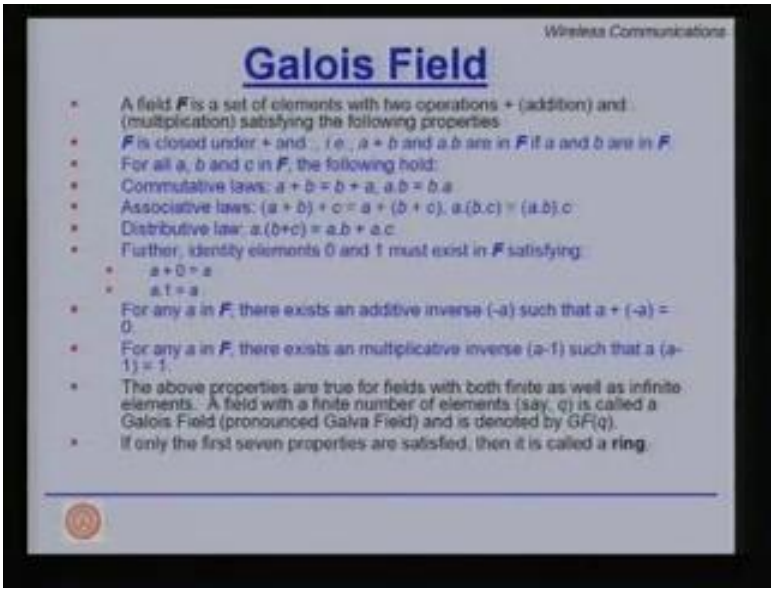


Let us talk about syndrome decoding. Suppose H a parity check matrix of an n, k code is defined by capital H then for any vector v which is an element of Galois field q^n, we will talk about Galois fields very soon the vector s is equal to vH^T is called the syndrome vector. What is s? S is nothing but v the received word multiplied by H transpose. Now we have seen earlier that if c word, if this vector v were the actual valid codeword c then c H transpose is zero. So the syndrome will be zero. In any other case when v, the received word is not equal to a valid codeword c you will not get a zero you will get something.

Now if that something can point to the specific error pattern then we are in business that is if this vector can be the syndrome to the disease which is our error here then we can actually decode it back. If there is a unique syndrome for every error that we can encounter then we can uniquely map back and remove that error and hence do an error detection and correction. So this s is called the syndrome of v, it is called a syndrome because it gives us the symptoms of the error thereby helping us to diagnose the error. Now let's take a very brief mathematical detour and talk about something called as the Galois field.

This is name after a French scientist Galois hence the pronunciation Galois, Galois field we will realize is something which follows a set of properties and we need to do this because we want to move into the next domain of linear block codes which is called the cyclic codes. Cyclic codes we'll realize are much easier to define once we understand a little bit of Galois field theory. So what is a field? A field F is a set of elements with two basic operations the addition and the multiplication and these elements of the field must satisfy the following properties.

(Refer Slide Time: 00:36:32 min)



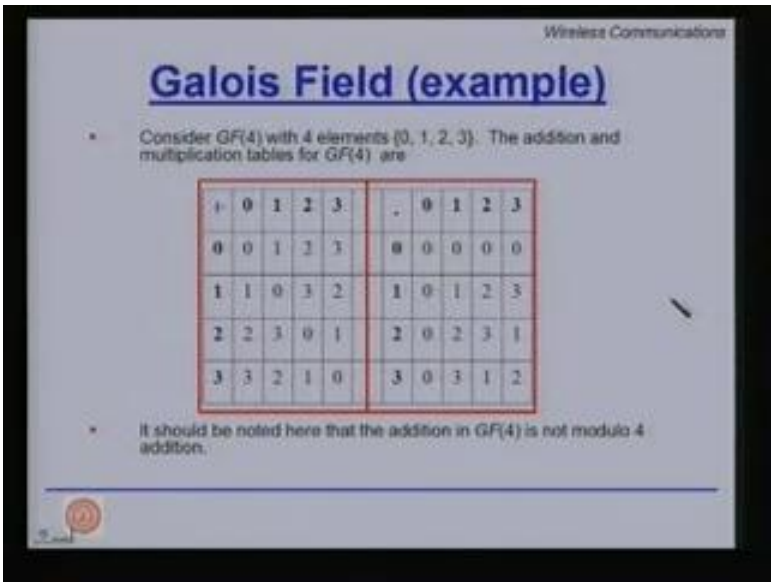
So if you can find a set of elements which have the basic addition and multiplication defined and satisfying the following properties, you have found a field and we will define our codes over certain fields. Normally we should say Galois field but in short we would just mention it as a field, we imply the same thing. So F is closed under addition and multiplication that is a + b and a times b are in F, if a and b are in F. So the resultant elements if you do a + b or a into b, if a and b belong to the set here then the addition and the multiplication yields another element should also belong to the same set. For all a, b and c in the field F the following should hold. The commutative laws that is a + b is equal to b + a, a times b is equal to b times a and so and so forth.

Associative laws a + b bracket plus c is equal to a plus bracket b + c or a times b times c in a bracket is equal to bracket a dot b dot c and then the distributive law a times b + c is a times b + a times c. Further the two identity elements which are 0 and 1 must exist in the field F satisfying a + 0 is a and a times 1 is a. Then we define two more things for any element a in the field, there must exist an additive inverse which is an element such that a + a minus is zero, this called the additive inverse of a. This must exist and for any element in a, there should be a multiplicative inverse also such as a times a inverse multiplicative is also one, except for zero. So it must be that the additive inverse and multiplicative inverse must; the above properties are true for fields with both finite as well as infinite elements.

Question: [Conversation between Student and Professor – Not audible ((00:39:53 min))] So this minus one has shifted down, it should a inverse, a raise to power minus one and here it's not a minus one, it is a raise to power minus one which is giving you the multiplicative inverse. A field with a finite set of elements say q is called the Galois field and is denoted by GF (q). So for all practical purposes, we are considering a field of finite elements and will talk about Galois field GF (q).

If q is equal to 2 then we have GF (2) the binary field, if only the first 7 properties mentioned here that is no multiplicative inverse other than that all the properties are satisfied then that sub set is called a ring. So a ring with an additional property that a multiplicative inverse exist becomes a field. So if we can ensure this then we can define our field and a ring. Today we will learn to construct Galois field as well, let us now look at an example.

(Refer Slide Time: 00:41:13 min)

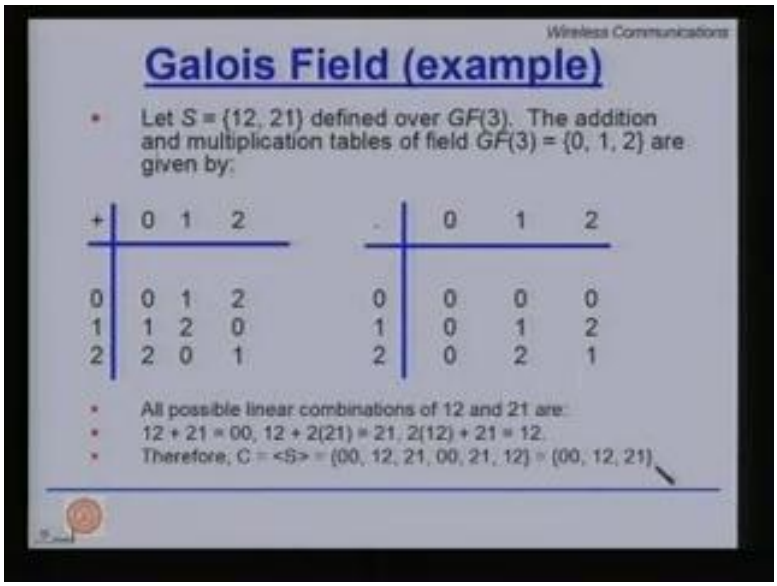


Let us consider GF (4) that means it's a Galois field with 4 elements. Let's denote this elements by 0 1 2 and 3. Now please note that there is nothing sacrosanct about defining it as 0 1 2 3 I could have defined it as 0 1 because 0 and 1 must exist and apple and an orange it still will form a set of elements but please note that this set must follow certain rules which we have mentioned in the previous slides and hence we should be able to construct an addition table and the multiplication table. If we have that we will ensure that all of the properties are satisfied. So we can define a field by just defining the addition table and the multiplication table.

So let's first look at the addition table. Clearly if you add something with the zero, you get back the same element so this two rows and columns are fixed and then you have for example 2 + 2 is 0. So this is the definition of a addition table and similarly you can have the multiplication table. So please note that here since you have zero's here it tells you that there is one element for every element present which is an additive inverse, all elements have an additive inverse, if it each element is its own additive inverse and similarly you will find a_1 somewhere in every row and column except that of the zero which tells you that there is a multiplicative inverse as well. So this is your GF (4) or Galois field with 4 elements. It should be noted that in the addition we are not carrying out any modulo four addition. Only when GF (q) where q is a prime number, will you find that the modular arithmetic works. Otherwise you have to construct the addition and the multiplication tables. Also it is interesting to note that Galois field for any q will not exist, it will exist only for prime and prime power.

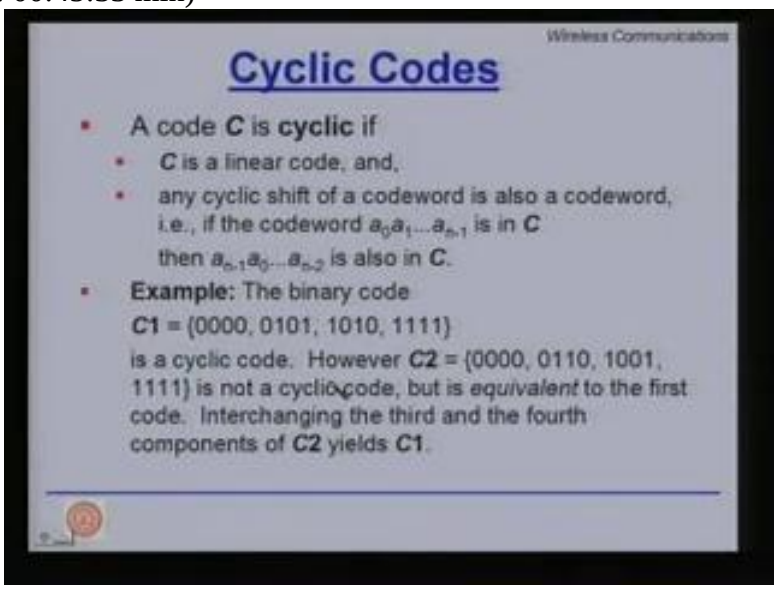
So GF (2) will exist, GF (3) will exist, GF (4) will exist, GF (5) will exist but GF (6) will not exist because 6 is neither a prime nor a power of a prime whereas GF (4) is 2^2 hence a prime power. Again G (7) will exist, G (8) will exist, G (9) will exist the G (10) will not exist. So not all Galois field or any arbitrary number of elements may not exist, it has to be either a prime or a prime power.

(Refer Slide Time: 00:44:38 min)



Let's look at another example. Let us talk about GF (3) so there are 3 elements since 3 is a prime number we actually have the modular arithmetic being valid here. So what we have here is the addition table and a multiplication table and if we want to construct a simple code which is a 2, 1 code then we can have a set of elements defined like this. So we can have the 0 mapped to 0 0, 1 mapped to 1 2 and 2 mapped to 2 1.

(Refer Slide Time: 00:45:33 min)

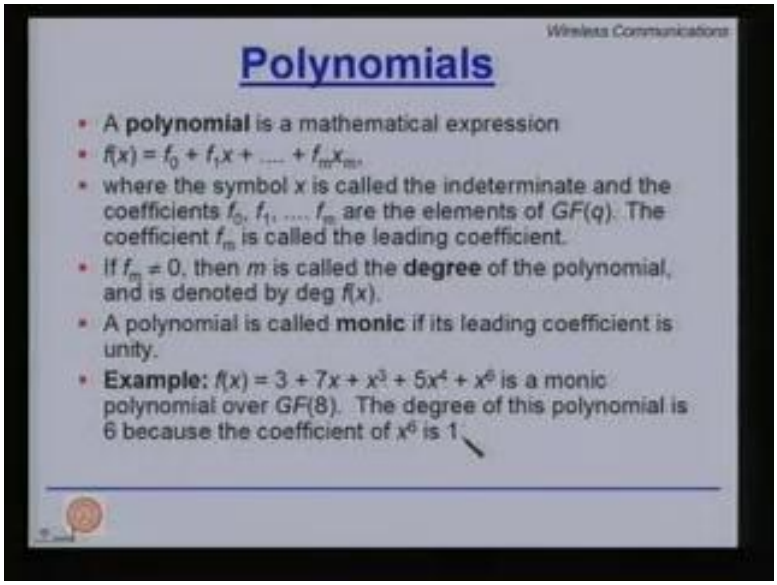


All this mathematical detour was important to define another important subclass as we said of the linear block code, call the cyclic codes. So cyclic codes is a special class of linear block codes with another constraint, another algebraic constraint and please remember the whole game encoding theory is to learn how to put more and more structure into your codes because any break in the structure due to noise can be easily detected and corrected. So a code C is cyclic, if C is a linear code and on top of that any cyclic shift of a codeword is also a codeword. That is if the codeword a_0, a_1, so and so forth a_{n-1} is in C then if you do a cyclic shift that is you put the last element front and shift each one by one so a_{n-1}, a_0, a_1, a_2 up to a_{n-2} is also in C.

Any cyclic shift; shift by one, shift by two any number of cyclic shifts will result in another vector which must also be a valid cyclic codeword. Please understand that if this is true, if we can really construct cyclic codes then they are very hardware friendly. We can generate them using shift register without any problems. So let's look at an example. Consider the following binary code 0 0 0 0, 0 1 0 1, 1 0 1 0 and 1 1 1 1. The block codes, all zero codeword is a valid codeword, you can check that some of any two codewords is also valid codeword so then it means that this is linear code the first property is satisfied but next look at any cyclic shift. If you rotate this by one so put this one here and right shift each one of them, you get 1 0 1 0 which is this one and if you right shift one again you get this one. If you do a cyclic shift of this you get back this one.

So any cyclic shift gives you any one of the valid codewords already present in the set. Hence this is an example of a cyclic code. Now let's look at a small example here, C_2 which is 0 0 0 0, 0 1 1 0, 1 0 0 1 and 1 1 1 1 is not a cyclic code, will be few shift it here you will get 1 1 0 0 which is not already present in the set of codewords. However C_2 is equivalent to the first code because all the distance properties are similar but one is a valid cyclic code, the other one is not.

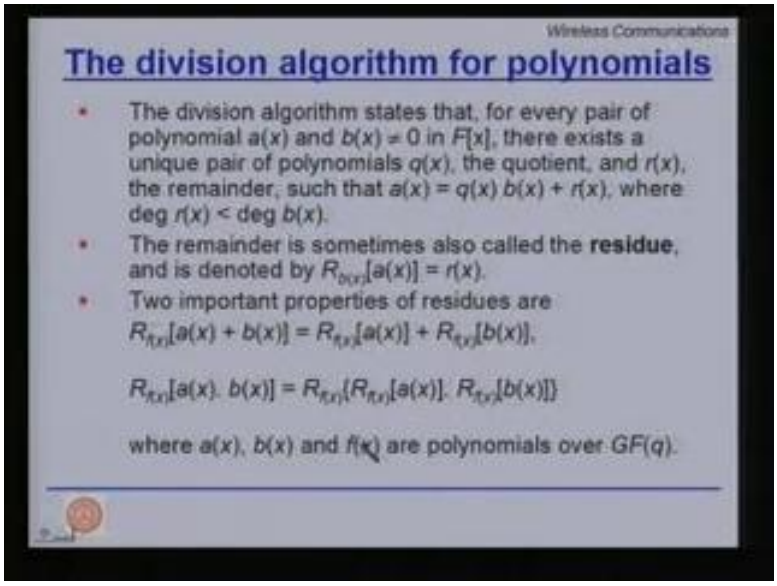
(Refer Slide Time: 00:48:50 min)



Continuing with our mathematical detour, let's talk about polynomials because we will soon see it is very easy and efficient to represent cyclic codes using polynomials. A polynomial is a mathematical expression for example f (x) is equal to $f_0 + f_1$ (x) + f_2 (x) squared so and so forth till f_m (x^m) so it should be a superscript where the symbols x is called the indeterminate and the coefficients f_0, f_1 up to f_m are called the elements of GF (q). So if I am going to define a polynomial over GF (q) then these coefficients of x must be taken from GF (q). If it is GF (2) binary case then f_0, f_1, f_m should be 0 or 1, if it is from GF (3) it should be 0 1 or 2 and so and so forth. The coefficient f_m is called the leading coefficient which belongs to the highest power x^m here. If f_m is not equal to zero then m is called the degree of the polynomial and is denoted by degree of f (x) that is the highest power of x.

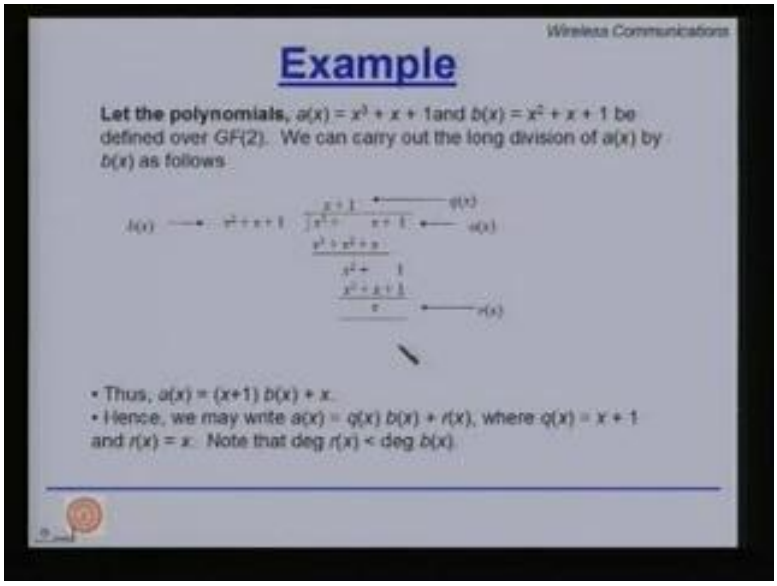
A polynomial is called monic if its leading coefficients f (m) is unity. So for example this is a monic polynomial over GF (8). Why is it over GF (8)? Please note all the coefficients belong to elements of GF (8) 0 1 2 3 4 up to 7 but the highest power is x^6, it's a degree 6 polynomial the coefficient is 1. Hence it's called a monic polynomial.

(Refer Slide Time: 00:50:50 min)



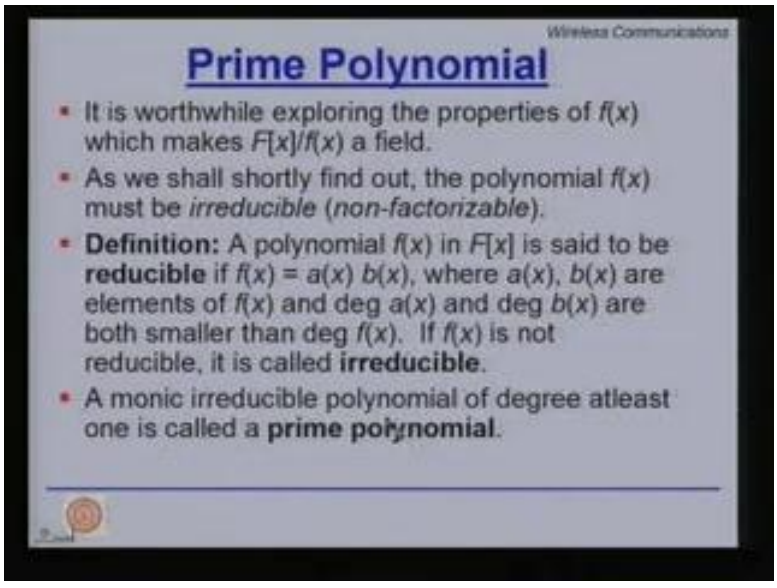
Let's talk about the division algorithm for polynomials. The division algorithm states that for every pair of polynomials a (x) and b (x) where b (x) is not equal to zero in the set of polynomials F (x), there exists a unique pair of polynomials q (x) the quotient and r (x) the remainder such that a (x) is q (x) b (x) + r (x) where the degree of r (x) is less than the degree of b (x). The remainder is sometimes also called the residue and is denoted by $R_{b(x)}$ [a(x)]. So it means that you take the polynomial a(x) and you divide it by b(x) you may get some remainder r(x) that remainder is called the residue. It's like the long division that we have learnt in our school earlier. Two important properties of residues R residue a(x) + b(x) is nothing but residue a(x) + residue b(x) and residue a(x) times b(x) is the product of residues a(x) and residues b(x). Here a(x), b(x) and f(x) are polynomials over GF (q); we will use these properties later on. Let's look at an example.

(Refer Slide Time: 00:52:18 min)



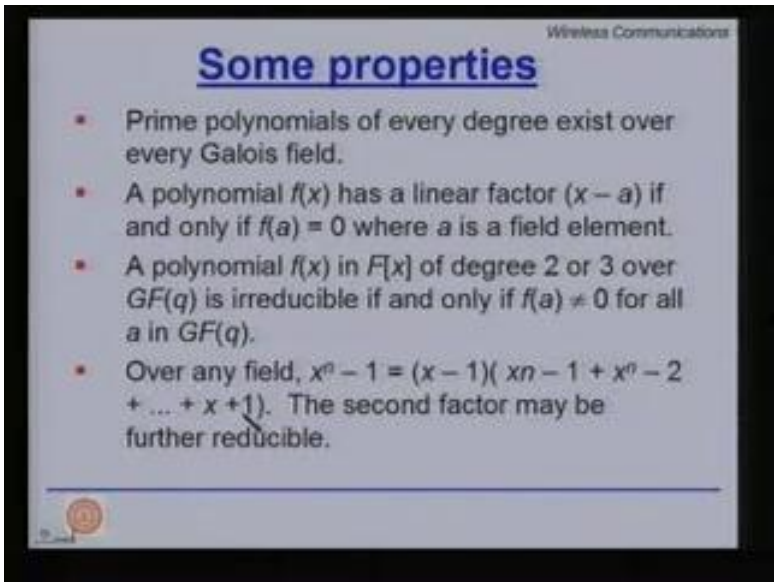
We would like to divide this a(x) which is $x^3 + x + 1$ by b(x) which should be non-zero $x^2 + x + 1$ and we need a quotient and a remainder, you see the remainder is the more interesting part so you carry out the basic division which is a long division and we finally obtain a residue. Thus a(x) here is x + 1 times b(x) + x. This is all the division algorithm of the polynomials state. Please note there is a degree of r(x) must be less than the degree of b(x) which is two that is all there is to it. Now let's do a couple of more definitions, it's worthwhile exploring the properties of f(x) which makes capital F(x) by f(x) a field. So what is this capital F square bracket x? It's a set of polynomials divided by f(x) of field so this is again a set of elements just like you have seen that earlier GF (2) has 0 and 1, GF (3) is 0 1 2 elements. You can have polynomials as the elements in the field.

(Refer Slide Time: 00:53:17 min)



We would like to know when is this a field? A field as you know satisfied certain number of properties. We'll find out shortly that the polynomial f(x) must be irreducible that is non factorizable to make this a field. So let's define a polynomial f(x) in capital F(x) is said to be reducible if f(x) can be written as a product of a two polynomials that is you can factorize it where a(x) and b(x) are elements of f(x) and degree of a(x) and degree of b(x) are both smaller than degree of f(x). However if you cannot do so, if f(x) is not reducible it is called irreducible and if your f(x) is irreducible then this becomes a field. A monic irreducible polynomial of degree at least one is called a prime polynomial, we will use this definitions.

(Refer Slide Time: 00:55:07 min)

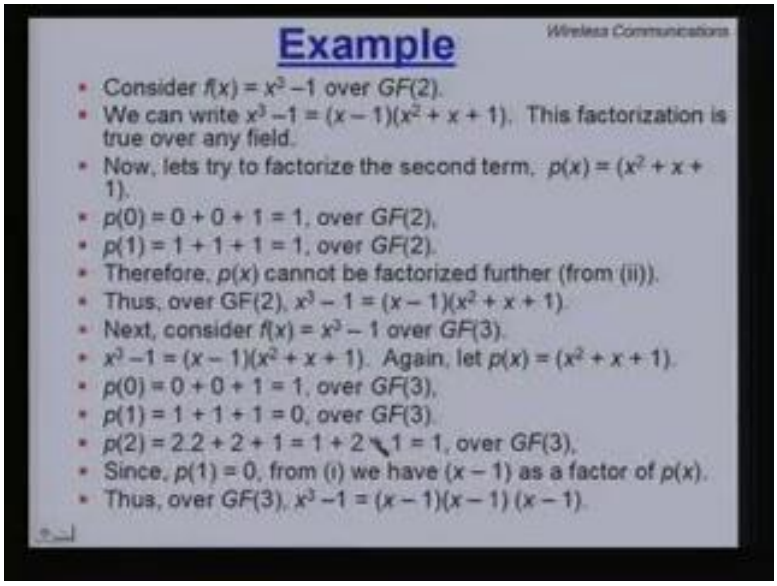


Prime polynomials of every degree exist over every Galois field, a polynomial f(x) has a linear factor x - a if and only if f (a) where a is a field element. A polynomial f(x) in F square bracket x of degree 2 or 3 over GF (q) is irreducible if and only if f (a) is not equal to zero for all a is in GF (q) and over any field we can write this following factorization x^n -1 is x-1 times xn -1+ x^n -2 and so on so forth till x + 1 this second factor may be further factorized.

Let's look at an example of factorizing x^3 -1 over GF (2), so clearly you can write x^3 -1 as x - 1 times x^2 + x +1. This factorization is true over any field but please note as we proceed further if you try to factorize x^3 -1 over GF (2) you may not end up with the same result if you factorize x^3 -1 over GF (3). So continuing over GF (2) let's try to factorize this term x^2 + x+1. First substitute zero because we have only two elements so I am trying to see whether there is an x - 1 factor or not or x - 0 factor or not.

So if you substitute zero you don't end up with a zero so it's clearly x - 0 is not a factor. If you put one again you do not end up with zero so clearly x -1 is not factor. Therefore p(x) cannot be factorized further over GF (2). So the best you can do is to say over GF (2), it is for the binary case x^3 -1 is nothing but x - 1 terms x^2 + x +1 so this is irreducible. We need irreducible polynomials which are monic and hence prime polynomials to do other interesting things for cyclic codes.

(Refer Slide Time: 00:56:04 min)

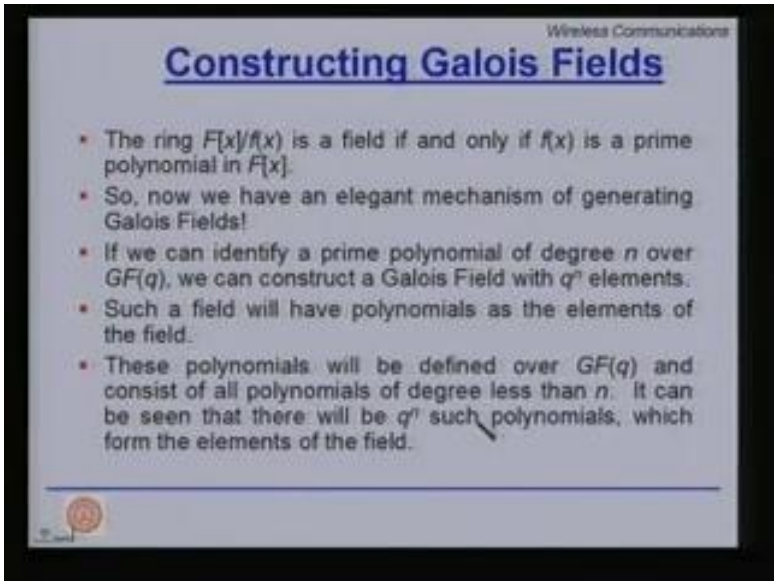


Now consider the same thing of factorizing x^3 -1 over GF (3), again with your eyes closed you can write x^3 -1 is x -1 times x^2 + x +1. Again try to substitute p (0) =1 so x is not a factor put p (1). Now this addition has been carried over GF (3), here additions were carried over GF (2). Addition tables in GF (2) and GF (3) are different and hence x - 1 should be a factor. Again if you substitute 2 you don't get x -2 as a factor. We can conclude that if p(1) is zero p is defined here x -1 must be a factor. So you can write over GF (3) x^3 -1 is x -1 times x -1 times x -1. So clearly the term x^2 + x+1 is irreducible that is non factorizable over GF (2) but easily factorizable over GF (3). It's important to know over which field are you working, we will learn to construct codes over specific Galois fields.

So now we have a very nice way to construct Galois fields if we know that if f(x) is irreducible, if it is a prime polynomial. So the ring F(x) over f(x) is a field if and only if this denominator is a prime polynomial in F (x). What is F squared bracket x, is a set of polynomials. So we now have an elegant mechanism of generating Galois fields, if we can identify a prime polynomial of degree n over GF (q), we can construct Galois field with q^n elements.

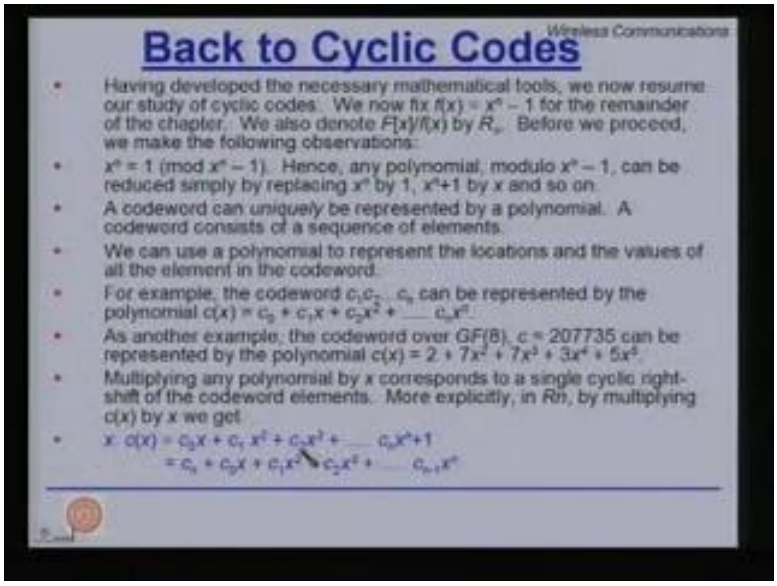
So the job is to identify the prime polynomial. Such a field will have polynomials as a elements of the field. This polynomials will be defined over GF (q) and consists of all polynomials of degree less than n, it can be seen that there will be q^n such polynomials which form the elements of the field.

(Refer Slide Time: 00:59:21 min)



So how do we relate all this to cyclic codes? Having developed the necessary mathematical tools we now resume our study of cyclic codes. We now fix f(x) is equal to x^n -1 for the remainder of the talk, so we denote F(x) by f(x) by R(n) but we make the following observations. First of all x^n =1 provided you take modulo x^n -1. Hence any polynomial modulo x^n -1 can be reduced simply by replacing x^n by 1, x^n +1 by x and so on. This is an important observation it means that if you are multiplying any polynomial by x^n.

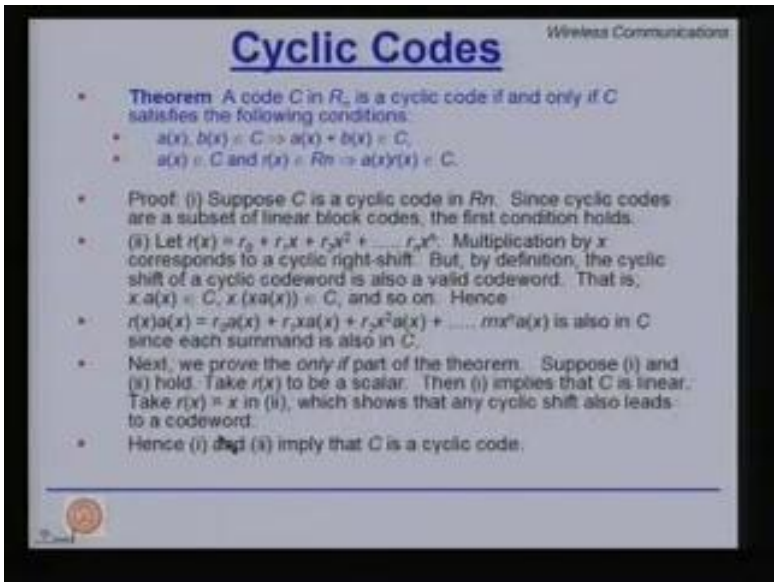
(Refer Slide Time: 1:00:36 min)



We effectively multiplying it by one, a codeword can be uniquely represented by a polynomial. So here is the missing link. Why are we trying to study polynomials in coding theory, because what we do is any codeword can be represented uniquely by a polynomial because what is a codeword, it's a sequence of elements. We can put this sequence of elements as coefficients of the polynomial, so this is a one to one correspondence. So we can use a polynomial to represent the locations and the values of all the elements in the codeword. For example the codewords c_1, c_2 up to c_n can be represented by the polynomial $c_0 + c_1 (x) + c_2 (x^2)$ up to $c_n x^n$.

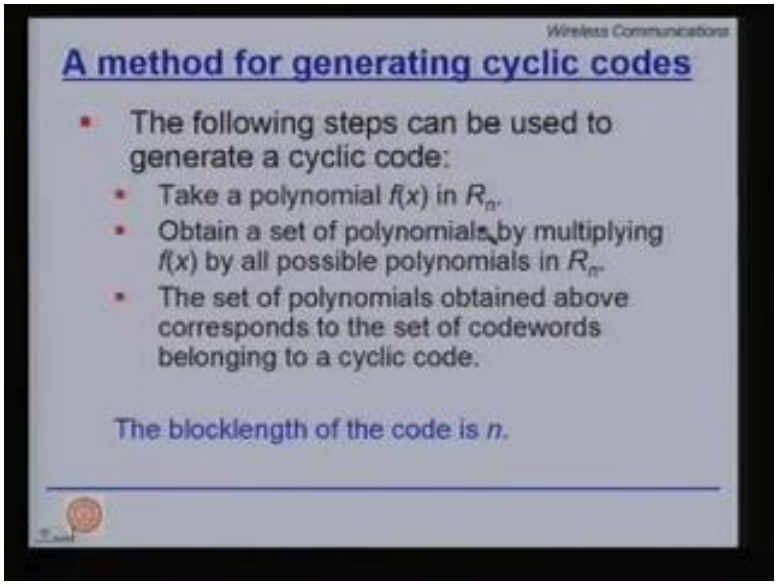
So c_1, c_2 correspondingly form the coefficients of x, x^2 and so on so forth. So there is a one to one correspondence but please note if you multiply this by x, all we do is raise the power of x by 1. So effectively we are shifting the elements to the left but we must ensure that the number of elements do not exceed the block length. So we must do a modular operation to bring back the highest power back to n - 1 and therefore we have to carry out the modulo x^n -1 operation. So here is another example where a codeword can be represented with polynomials. Here again we have represented, if you multiply c(x) by x you just shift the polynomials.

(Refer Slide Time: 1:03:37 min)



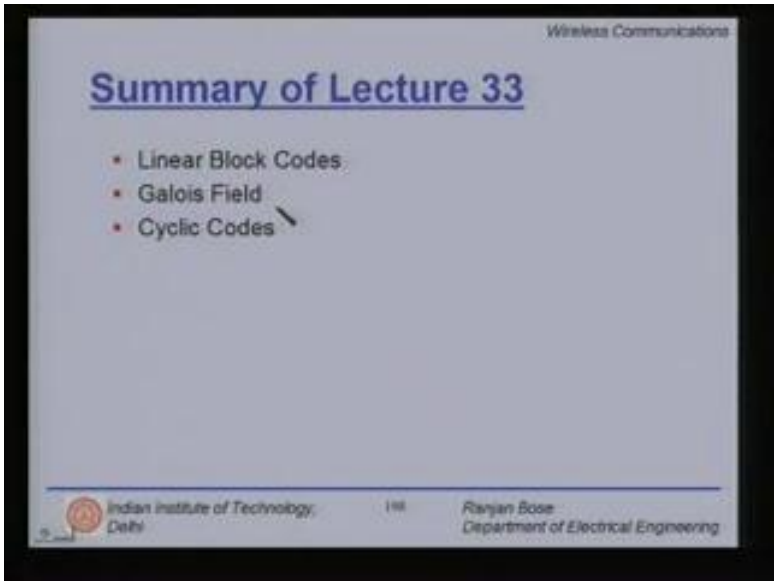
So there are certain theorems for cyclic codes that we would like to take up in the subsequent lectures that it is possible to represent cyclic codes by using polynomials defined using a modulo operations and this will be the starting point for next lectures.

(Refer Slide Time: 1:04:10 min)



So generation of cyclic codes will also be started in the subsequent lectures where it is very easy to define cyclic codes based on the polynomials f(x) defined over R_n.

(Refer Slide Time: 1:04:24 min)



So let's summarize today's lecture. We started off with a brief study of linear block codes. We then talked about Galois field theory and then linked it back to how to work with cyclic codes. We have just had a brief glimpse of cyclic codes. In the next lecture we will start with cyclic codes and look at certain applications for wireless communications.
Thank you