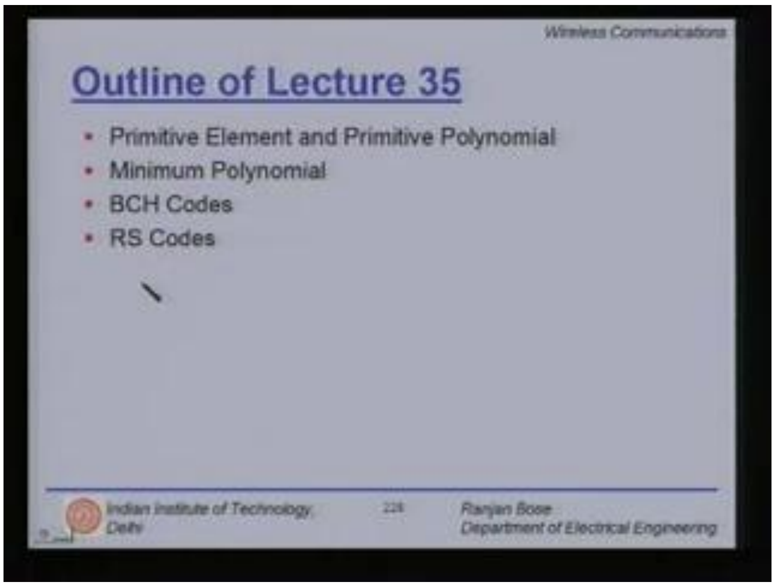
Wireless Communications
Dr. Ranjan Bose
Department of Electrical Engineering
Indian Institute of Technology, Delhi
Lecture No. # 35
Coding Techniques for Mobile Communications (Continued)

Welcome to the next lecture on wireless communications. Today we will deal deeper into various coding techniques for mobile communications. Let us look at the outline for today's talk.

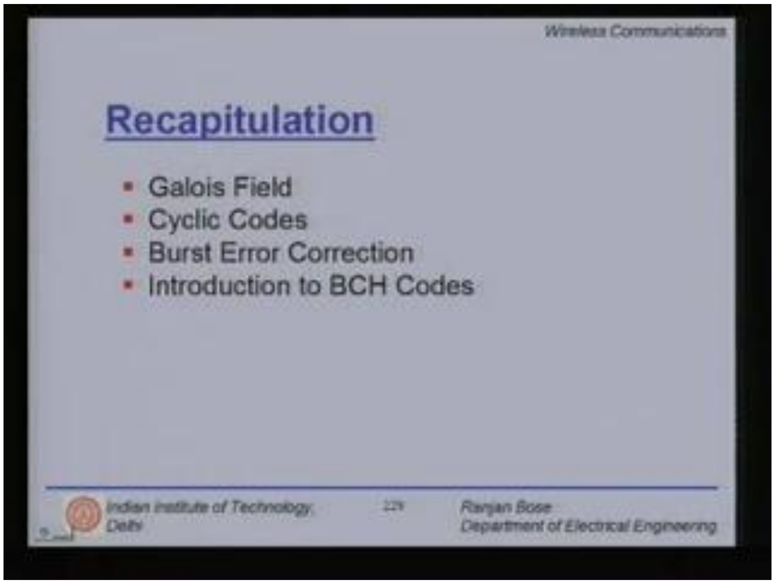
(Refer Slide Time: 00:01:32 min)



We will begin with the study of BCH codes followed by the Reed Solomon codes. We'll also talk about the basics of primitive elements and primitive polynomials required to understand the BCH codes and RS codesas well as talk about minimal polynomials required to construct the generator polynomials for BCH codes and RS codes. So this is the brief outline for today's talk; of course we'll start by summarizing what we have learnt so far. So let us recap.

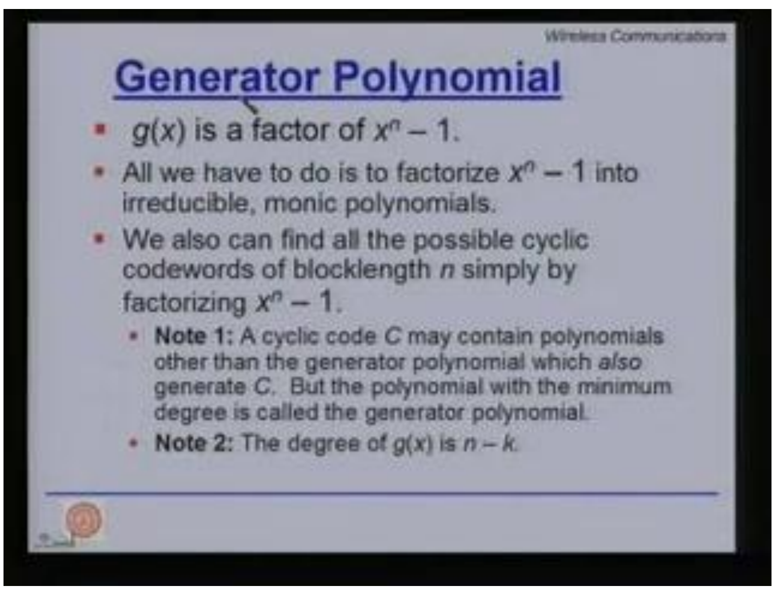
In the previous lectures we have studied the Galois field followed by an understanding of cyclic codes. What we learnt last time was cyclic codes was a subclass of linear block codes except that any cyclic rotation of a valid code word forms another valid code word. We then moved into the domain of burst errors which are very likely in wireless communications, when we get into deep fades we encounter errors which are not randomly distributed but occur in bursts and we also realized that cyclic codes are a very powerful class of error correcting codes which can be used for burst error correction.

(Refer Slide Time: 00:02:07 min)



We finally moved into the introduction of BCH codes which is a subclass of cyclic codes. Today we will take a brief mathematical detour to develop some tools to understand BCH codes and then finally come up with a generator polynomial for BCH codes.

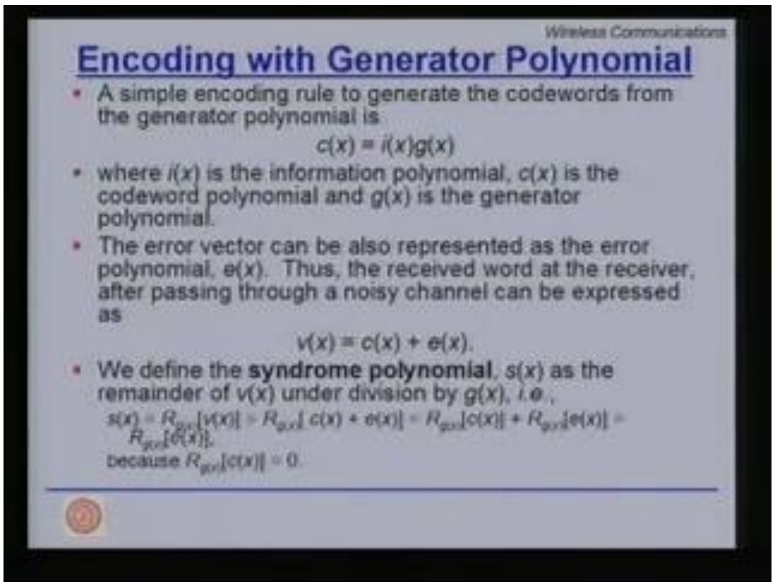
(Refer Slide Time: 00:03:21 min)



So recapping from last time we realize that the generator polynomial g(x) must be a factor of x raise to power n minus one for a cyclic code. Here this n represents the block length so if I have to find out the generator polynomial for a cyclic code of block length 7, all I have to do is take x raise to power 7 minus one and factorize it.

All of the factors have the potential to generate a cyclic code. In fact this is one way to come up with all possible cyclic codes of a block length n. So we have to note that a cyclic code C may contain polynomials other than the generator polynomial which also generate C, with the polynomial with the minimum degree is called the generator polynomial. We have also observed last time that the degree of g(x) the generator polynomial is n-k. All this is under the assumption that there is a unique 1-1 correspondence between a polynomial and a code word. So any code word can be represented by a polynomial and multiplying a polynomial by x merely tantamounts to rotating the code word polynomial by one to the right side.

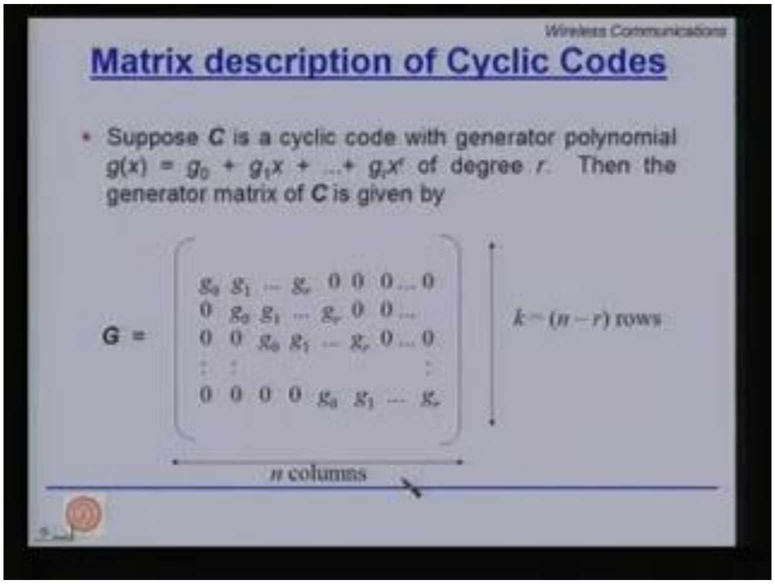
(Refer Slide Time: 00:04:48 min)



Now let us move over to encoding using the generator polynomial for the case of cyclic codes. A simple encoding rule can be c(x)= i(x) times g(x) where i(x) is the information polynomial just like you have the information word you can represent it using a polynomial so i(x) is an information polynomial, g(x) we know is the generator polynomial and c(x) is nothing but the code word polynomial. Now please note that g(x) has been designed in such a manner that for all possible i(x) the c(x) comes out as a valid code word that is it has been ensured that the highest power of x does not exceed that of the block length.

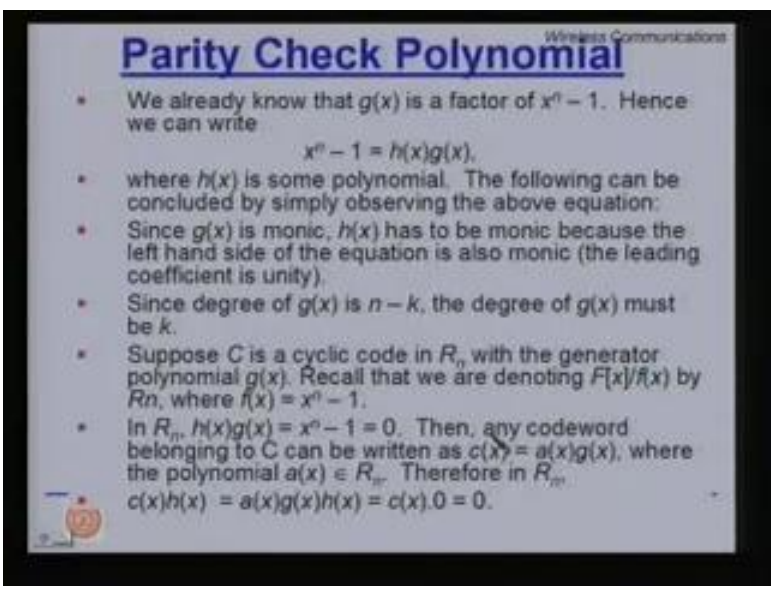
Now if you move forward and represent the received word v(x) as c(x) the code word plus e(x) the error polynomial, you can also represent the error by a polynomial in that case we can define something called as a syndrome polynomial s(x) as the remainder of v(x) under the division by g(x). Clearly if e(x) is zero, your v(x) will be equal to c(x) which can be exactly divided by g(x) because c(x) has been created by a multiplication of g(x) with i(x). For any other case when the error is nonzero, you would get some remainder and that pertains only to that error pattern hence the syndrome for that error.

(Refer Slide Time: 00:06:42 min)



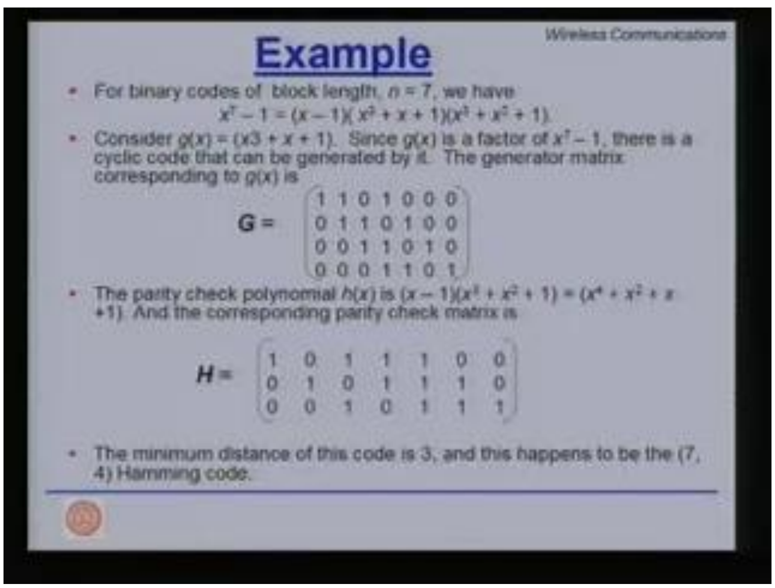
Now we also know that cyclic codes are a subclass of linear block codes that means it is possible to represent them using matrices, a generator matrix and a parity check matrix. Suppose you have your g (x) represented as g_0+ g_1 x + g_2 x squared so on and so forth till g_rx where r is the degree of g(x). Then the generator matrix can be represented as follows, the first row will be g_0, g_1 so on and so forth until g_r followed by zeros and then the second row is nothing but a shifted version and the third row is yet another shifted version and so and so forth. Please note that this first row and second row all are shifted with respect to each other by one and the number of rows here is n-r=k.

(Refer Slide Time: 00:07:45 min)



Just like we had a parity check polynomial we also have a parity check polynomial analogous to the parity check matrix in linear block codes. What is the philosophy? g(x) is a factor of x raise to power n minus one which means that x raise to power n minus one can be represented as h (x) times g (x). So h (x) is the polynomial which will denote as the parity check polynomial so since the degree of g (x) is n-k, the degree of h (x) must be k. Suppose C is a cyclic code in R_n?What is R_n? R_n is capital F(x) by f(x) where small f(x) is nothing but x raise to power n minus one? So doing all the operations modulo x raise to power n minus one, this will ensure that if you multiply a codeword by any power of x, it will pertain to a cyclic shift but the highest power will not exceed that of the block length. So that the highest coefficient moves back into the first place. Hence we take modulo x raised to power n minus one.

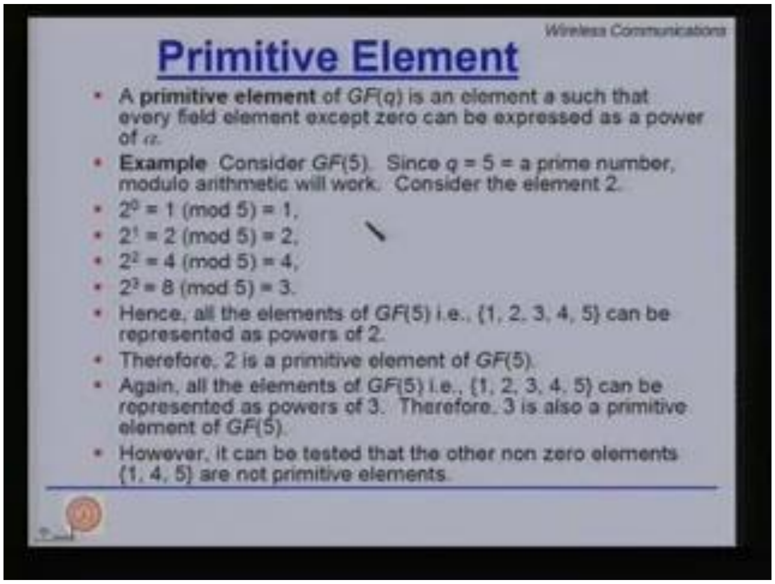
Any codeword belonging to C can be written as c(x) is equal to a(x) times g(x). However c(x) into h(x) can then be represented as a(x) times g(x) which is c(x) times h (x) and if you take modulo x raised to power n minus one, you have this as zero.This means h (x) has this unique property that if you multiply it with any valid code word polynomial, you end up with the zero polynomial. It’s a very simple yet elegant way to check whether a codeword is a valid codeword. (Refer Slide Time: 00:10:05 min)



Let us look at an example. For binary codes of block length n =7 and that is all I need to specify once I said for block length n =7 find for me the cyclic codes. All I have to do is take x raise to power 7 minus one and factorize it which I have done here. They have three factors, clearly individually all three can generate a cyclic code but product of any two will yet still be a factor of x raise to power 7 minus one and they can themselves generate a cyclic code and so and so forth. So 1, 2, 3 product of these two, product of these two and product of these two,6; product of all 3 which is x raise to power 7 minus one is the seventh and unity eight. There are 8 possible cyclic codes some of them may be trivial. All of them will have a block length n =7 and we are talking about binary, if you talk about non-binary cases then the factorization will be different because the multiplication and addition tables are different in different Galois fields.

So only for binary code if you designate x^3+ x +1 as our generator polynomial then based on that we can write the generator matrixand whatever is remaining that is if you have the product of the other two that will represent your h (x) which is given by x raise to power 4+x^2+x+1 and the corresponding H matrix parity check matrix is given by the following. We also made an observation that this happens to be the 7, 4 the Hamming code. So hamming code is also a cyclic code, the minimum distance of this code is 3, it is incorrect, one error so it is a single error correcting code.

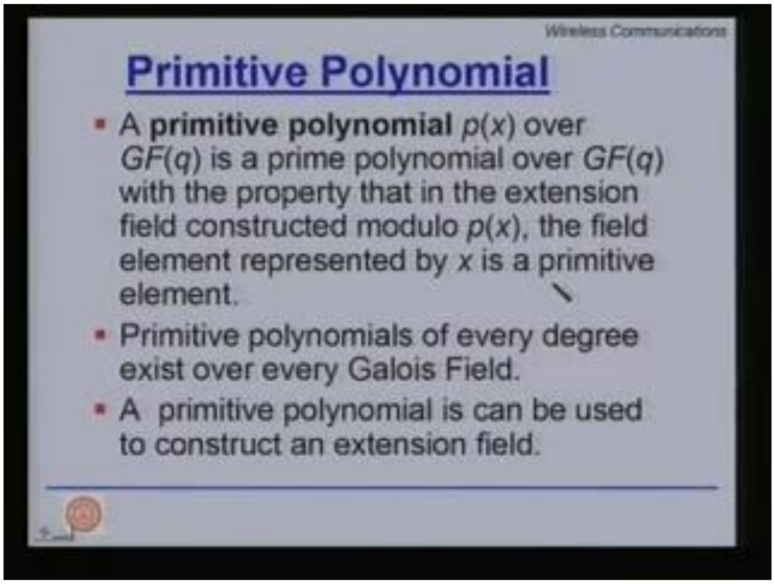
(Refer Slide Time: 00:12:11 min)



Now let us continue with our mathematical detour and let's talk about something called as a primitive element. A primitive element of GF (q) is an element such that every field element except the zero element can be expressed as a power of alpha, alpha being the primitive element. So example consider GF(5), 5 is prime so we know that a Galois field can exist and also since it is a prime number, modulo arithmetic will work. So let's consider the elements 0, 1, 2, 3 and 4 as the 5 elements of GF (5) then we have 2 raise to power zero is 1 but you have to take everything modulo 5 so it's 1,2 raise to power one comes out to be 2 if taken modulo 5;2 squared is 4,2 cubed is 3 hence all the elements of GF(5) that is 0, 1, 2, 3, 4, 5 is not an element;0, 1, 2, 3 and 4 can be represented as powers of 2 here as you can see except zero.So 2has this magical property of being a primitive element, all powers of 2 taken modulo 5 will jump on the different elements one by one.

Therefore two is a primitive element of 5, primitive elements are not always unique. So what happens is if you check for 3 you'll find it is also primitive element but not onebecause any powers of one will remain as 1 or 4,4 you cannot generate all the elements in GF(5). So 2 and 3 can be shown to be primitive elements of GF (5) so you can test that 1,4 and 0 are not the primitive elements.

(Refer Slide Time: 00:14:15 min)

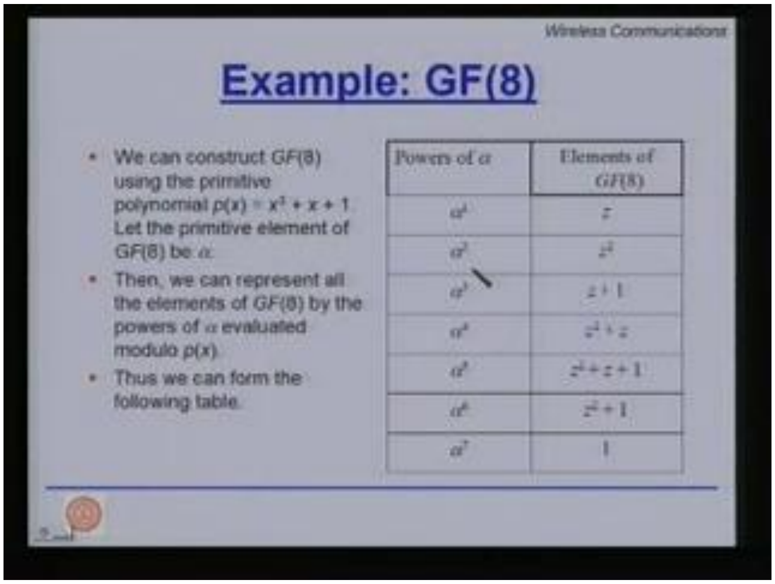


Now what is a primitive polynomial? A primitive polynomial say p(x) again defined over a certain GF (q) a Galois field is a prime polynomial. We know prime polynomial is that which cannot be factorized over GF (q) with the property that in the extension field constructed modulo p (x), the field elements represented by x is a primitive element. So let's go over this definition again, it appears complex so let's break it up. We are talking about a primitive polynomial p (x), it must have certain properties. First of all it is irreducible over GF (q) it's also monicso it's a prime polynomial over GF (q). Now we also know that it is possible to create an extension field from a sub field but what is the property of this primitive polynomial is that in the extension field constructed modulo p (x) the field element represented by x is a primitive element.We'll look at an example to illustrate the point.

Primitive polynomials of every degree exist over every Galois field, a primitive polynomial can be used to construct an extension field. We will realize this importance because once you're dealing with BCH codes and Reed Solomon codes, the notion of a sub field and an extension field is important. The philosophy is that if you have GF (2), you can construct GF (4),GF (8) from GF (2). If you have GF (3) you can construct the other fields like GF (9) from here.

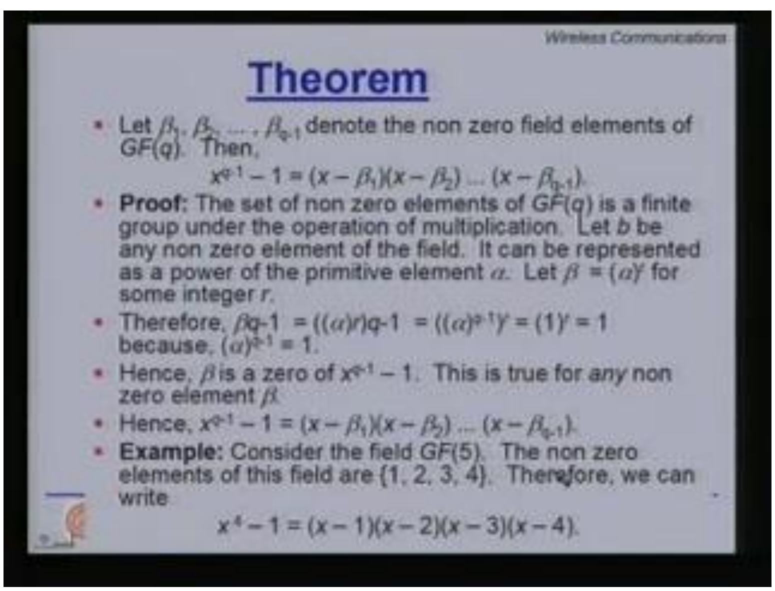
Let's look at an example. The example is of GF (8) so let's say the primitive polynomial p(x) is x^3+x +1. This is clearly non factorizable over GF (2) because you can substitute 0 and 1 and check that x-1 and x-0's are not the factors. Now let alpha be the primitive element and so all the powers of alpha must pertain to elements in GF(8) provided they are taken modulo p (x). So what you do is alpha lets represent it as z, alpha squared is z squared, alpha cubed taken modulo p (x) so if you take z cubed and divide it by z^3+z+1 you'll be left with z+1.So modulo p(x) that is alpha cubed taken modulo p(x) will be z+1 the remainder.

(Refer Slide Time: 00:16:11 min)



Similarly alpha 4 so take z raise to power 4 because alpha is represented by z take it modulo p (x) that is divide by z^3+z+1 and you'll be left with z^2+z and so and so forth.So if you carry down this exercise you will realize that you have 7 elements except the zero element which are the elements of GF (8) and it can be verified with all these follow the 8 axioms of a field.Hence we have just constructed using a primitive polynomial p (x), the field GF (8) from GF (2) so we have this following table.

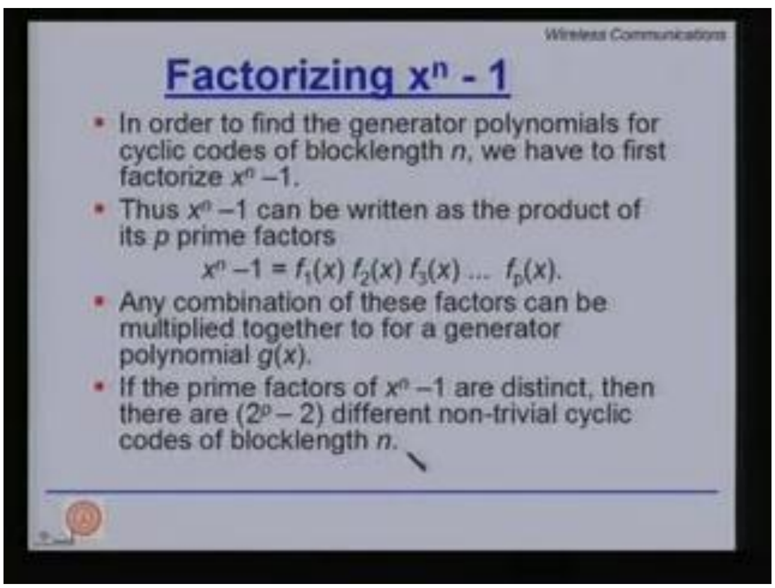
(Refer Slide Time: 00:18:15 min)



So let $beta_1$, $beta_2$, so and so forth up to $beta_{q-1}$ denote the nonzero field elements of GF (q).Then what we have is x^{q-1}-1, if you take this one it can be factorized as (x- $beta_1$)(x-$beta_2$) so on and so forth till (x- $beta_{q-1}$). What is the proof?The set of nonzero elements of GF (q) is a finite group under the operation of multiplication.Let beta be any nonzero element of the field then it can be represented as a power of the primitive element alpha.So what does it mean, we can say that beta is alpha raise to power r for some integer r because alpha is the primitive element.Therefore beta raise to power q-1 is alpha raise to power r whole raise to power q-1 can be represented like that is equal to one raise to power r is one because alpha raise to power q-1 is 1. We have seen in the previous example that alpha raise to power q is 8, minus one will ultimately come down to one. It'll look through all the elements and alpha raise to power q minus one will be one.

Hence beta is a zero of x^{q-1}-1 and this is true for any nonzero element beta, hence we can write x^{q-1}-1 as the product of (x- $beta_1$) (x- $beta_2$) so and so forth till (x- $beta_{q-1}$). So if you consider the GF (5) you can close your eyes and simply write as x^4-1, q is 5 so q-1 is 4 as (x-1)(x-2)(x-3) and (x-4).Now let's go back to our problem because why are we taking this mathematical detour. We are trying to find easier ways to factorize x^n-1 and also try to see if you can come up with a constructive technique for coming up with g (x) with certain desirable properties.

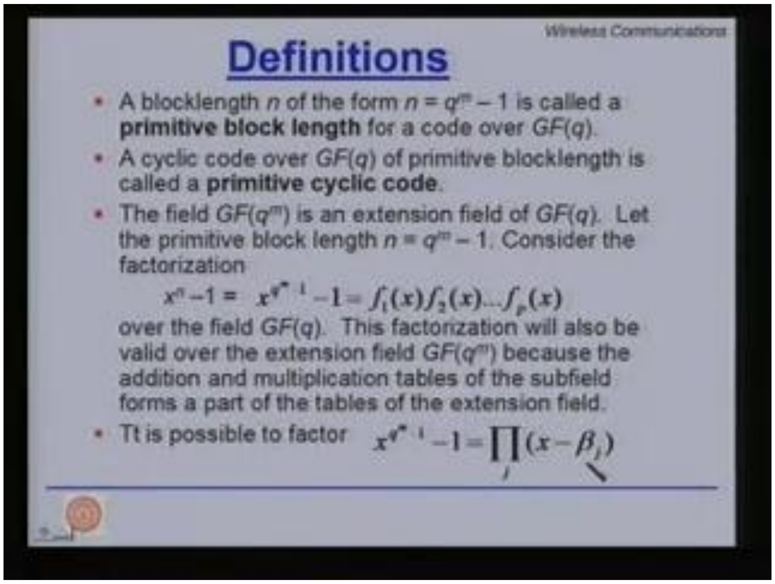
(Refer Slide Time: 00:20:28 min)



What are those desirable properties? The number of errors, the code can correct per block because the strong or weak code depends on how many errors it can correct and that depends on the minimum distance of the code. What we want to do is having got some kind of a constructive technique, can we pre specify how many errors the code generated by the generator polynomial can correct. So we know in order to find the generator polynomials for cyclic codes and BCH code forms a subclass of cyclic codes we have to first factorize x^n-1.Now x^n-1 can be written as a product of p prime factors like this, any combination of these factors can be multiplied together for a generator polynomial g (x).

We have done a previous example where we had taken to factorize this and then come up with. The question is can we directly write this factorization from the mathematical tools we have developed. If the prime factors are x^n-1 are distinct then there are 2^p-2 different non-trivial cyclic codes for block length n, this you have seen earlier.

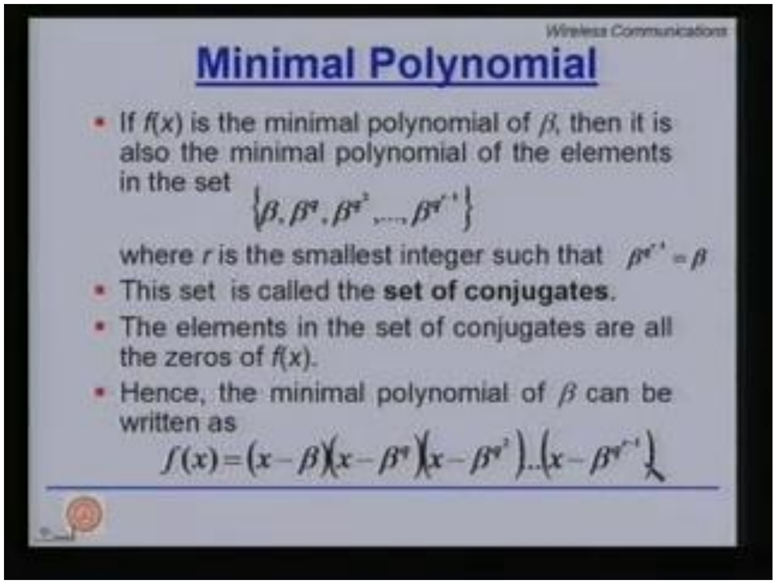
(Refer Slide Time: 00:22:14 min)



So let's go over the basic definitions. A block length n of the form n =q^m-1is called the primitive block length for a code over GF (q). This definition is particularly important from the perceptive of BCH code. So a special block length, what is it? It is q raise to power an integer m-1, so if q is 3 and m is 2 so n can be 3^2that is 9-1, 8. A cyclic code over GF (q) of primitive block length is called primitive cyclic code. The field GF (q^m) is an extension field of GF (q). For this let the primitive block length n be equal to this one (Refer Slide Time: 23:20). So we consider the following factorization x^n-1 this has been our most famous polynomial x^n-1 which has to be factorized in order to get the generator polynomials. Now n clearly is of the type q^m-1. So I substitute it here and this can be represented say as f_1(x) times f_2(x) so on so forth till f_p (x).

This factorization will also be valid over the extension field that we have just generated GF (q^m) because the addition and multiplication tables of the sub field form a part of the tables of the extension filed. This is an important observation that the sub field tables, so if you talk about the sub field as GF (8) and extension field as GF (8) then you will see that the addition and multiplication table for GF (8) contain within itself the addition and multiplication tables of GF (2). Since this is valid then you can write one possible factorization x^n-1 and substituting n for q^m-1 minus one as nothing but the products over j x raise to power minus B_j. So we have a direct way of factorizing.

(Refer Slide Time: 00:24:46 min)



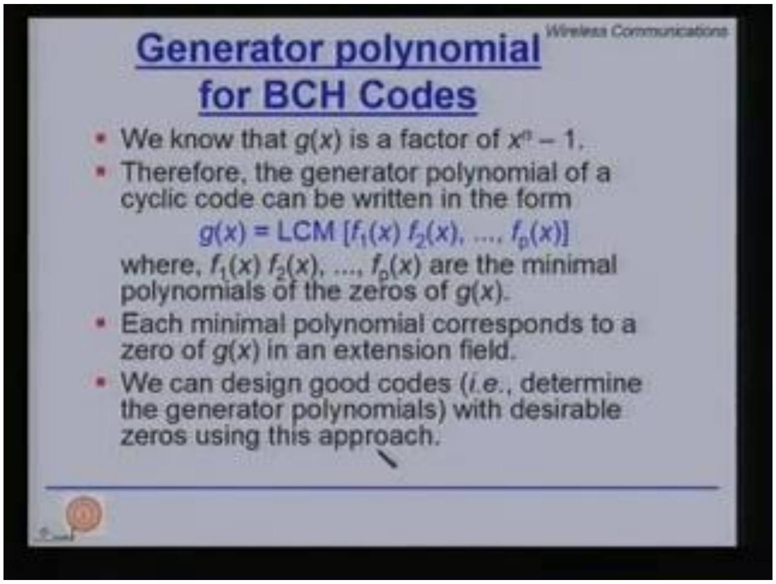
What are minimal polynomials? In order to find the generator polynomials for the cyclic codes of block length n, we have to first factorize x^n-1. We have seen it can be written as the product of p polynomials and any combination can give you a factor.The smallest degree polynomial with coefficients in the base field GF (q) that has a zero in the extension field, GF (q^m) is called the minimum of beta j. We are talking about a smallest degree polynomial where the coefficients are in the base field GF (q) but that is a zero in the extension field. It must have a zero because we intend to factorize it as (x- $beta_1$)(x- $beta_2$) and so forth.

Let us consider an example, consider a sub field GF (2) and its extension field GF (8).Here clearly q =2 and that integer m =3 hence we have q^m is 2^3which is 8.The factorization in the sub field or the extension field yields x^7-1,7 is the primitive block length is nothing but (x-1)(x^3+ x+1)(x^3+ x^2+1)but next consider the elements of the extension field of GF (8). These are the following, these element if you remember we had just constructed from GF (2) using modulo operation, modulo operation over primitive polynomial which was the primitive polynomial x^3+ x+1. Therefore we can write very conveniently the factorization x^7-1 is nothing but (x-1)(x-z)(x-z-1) and so and so forth till (x- z^2 –z-1). That is (x- $beta_1$)(x- $beta_2$)(x- $beta_3$) and so and so forth. What is magical is when you multiply this out all the terms, all the z terms will cancel out and you will be left simply with x^7-1.

Now if this is true then we already have a very elegant way to pick and choose the factors and also create a cyclic code as strong as we want. Clearly if we have more number of factors in my g (x) the highest power of g (x) increases that is n-k increases because the degree g (x) is n-k. So if n-k increases we are putting in more and more redundancy. You can build stronger and stronger codes, exactly how strong; we'll just mention. If f_x is the minimal polynomial of beta then it is also the minimal polynomial of the elements in the set beta, $beta^q$, beta raise to power q squared and so and so forth. This can be shown where r is the smallest integer such that beta raise to power q raise to power r minus one is beta.

It's a property this set is called the set of conjugates; the elements in the set of conjugates are all zeros of f(x). Hence the minimal polynomial of beta can simply be written as f(x) is equal to (x-beta)(x-beta^q) so and so forth till (x- beta raise to power q raise to power r minus one).

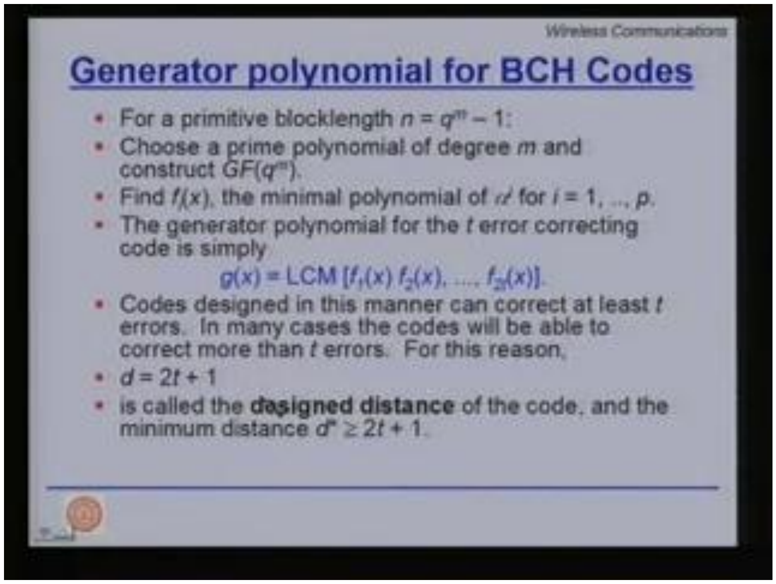
(Refer Slide Time: 00:29:01 min)



Now we come back to our BCH codes and try to think of a way to create generator polynomials for BCH codes. Again it is a cyclic code, we have to have a factorization of x^n-1 but this time we'll put a constraint on n, it cannot be any arbitrary block length, it has to be primitive block length. So what we do is we can write g(x) as a generator polynomial is simply the LCM, the least commonmultiple of f_1(x), f_2(x) so and so forth till $f_p$xwhere f_1(x), f_2(x) and so and so forth till f_px are the minimal polynomials of the zeros of g (x). We have established this, so each minimal polynomial corresponds to zero of g(x) in an extension field. We can design good codes that determine the generator polynomials with the desirable zeros using this approach.

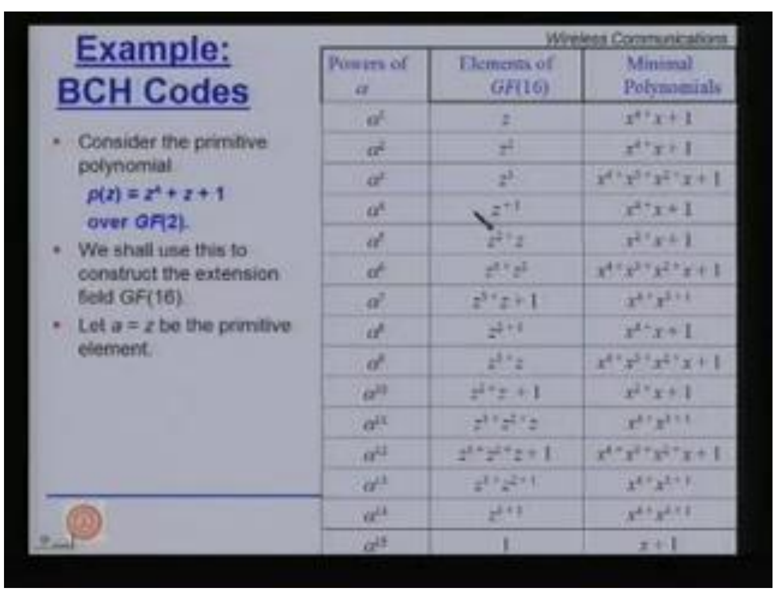
So with the primitive block length n is equal q^m-1, choose a prime polynomial of degree m and construct GF (q^m). Let us now look at the recipe for getting the generator polynomial for any BCH code. We start with a primitive block length because the factorization that we have talked about holds good only for this special block lengths.Choose a prime polynomial of degree m and construct GF (q^m) so keep in the back of your mind that we are constructing for example GF (8) from GF (2) where q is 2 and m =3. Find f_i(x) the minimal polynomial of an alpha^{j}for i =1, 2, 3, 4 up to p. The generator polynomial for the t error correcting code is then simply given by LCM f_1(x) times f_2(x) up to f_{2t}(x).

(Refer Slide Time: 00:30:10 min)



Codes designed in this manner can correct at least t errors that is we have a recipe for coming up with a generator polynomial that give us a cyclic code which is BCH which can correct at least t errors, you can come up with a generator polynomial which can over do the design, it can correct more than t errors but it will guarantee you that will correct at least t errors. For this reason d =2t+1 is also called the designed distance of the code and the minimum distance d^8 could be less than this one. So you can over design your code using this technique.

(Refer Slide Time: 00:32:05 min)

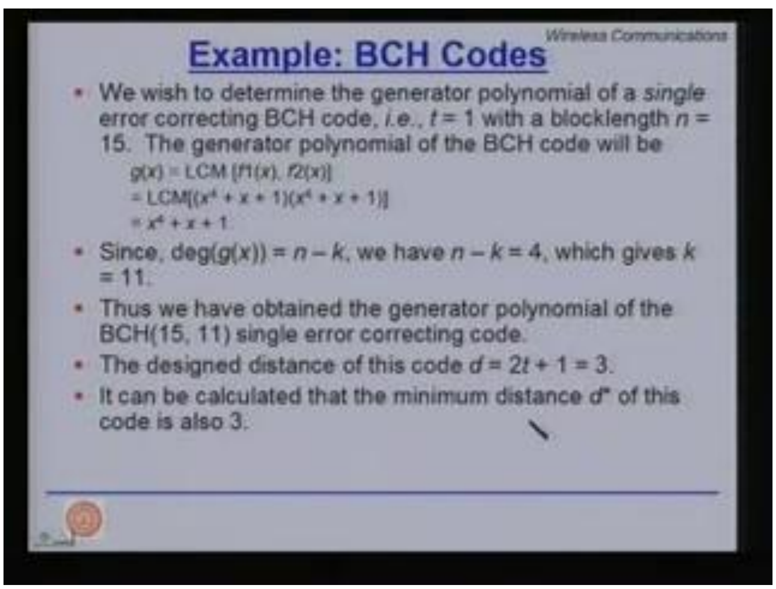


Let's look at an example. We have to start with the primitive polynomial let's say a primitive polynomial is z^4+z +1 over GF (2).

Quickly check that it is a prime polynomial because you cannot factorize it and its monic. We shall use it to construct the extension field GF (16) this time. Clearly I can construct with an appropriate choice of a primitive polynomial any extension field. So GF (2) is the base field and GF (16) is the extension field, let alpha =z be the primitive element then you take powers of alpha and keep on taking modulo p (z). So alpha = z, alpha squared = z squared, alpha cube = z cube but the moment you have alpha^4= z^4 what you do is take modulo of this one so we will get z+1. Again alpha^{5}is z squared + z and so and so forth. Again please note alpha raise to power q minus one alpha^{15} will give you one.

On the right hand side of the table we have the corresponding minimal polynomials. So what is interesting to note is that minimal polynomials are not unique, see you have an x^4+x+1 here then again you have x^4+x+1for alpha^2 and then for alpha^4 again you have the same thing and magically for alpha^8 again you have this.Consider alpha^3 you have this term which is all the powers of x^4+x^3+x^2+ x+1 but it occurs for alpha^3, again it occurs for alpha^6 then again it occurs for alpha^9 and again it occurs for alpha^{12}. These are definite pattern, the pattern we observed last time beta, beta^2 and so and so forth.

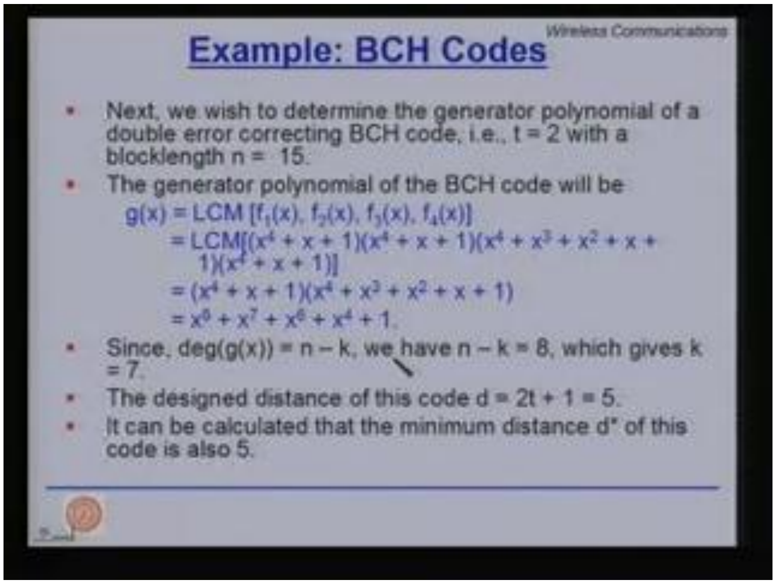
(Refer Slide Time: 00:34:31 min)



So now let us look at an example of constructing the generator polynomial for a particular BCH code. Here let us start with a block length n =15 and we wish to construct a single error correcting code that is t =1 then the generator polynomial for the BCH code can be written as g(x) is equal LCM of f1(x), f2(x) so and so forth till f2t(x) but t =1 so we stop at f2. Clearly it is LCM for f1(x) and f1(x) but from the previous example what we have seen is f1(x) and f2(x) are identical. So this LCM comes to our rescue and we just retain one of them x^4+ x+1 simple, this is our generator polynomial. We have just found the generator polynomial of a single error correcting code. Now please observe the following, the degree of g (x) is n-k but here n-k is 4 which gives k =11. Why, n =15 the starting point thus we have obtained the generator polynomial for the BCH (15, 11) single error correcting code.

The designed distance of this code is 2 t +1 =3, it can be also calculated the minimum distance of this code also happens to be 3. If the minimum distance is 3 it is definitely a single error correcting code. In this case the designed distance is actually equal to the minimum distance of the code. Let us now become more ambitious and say no we are not happy with t =1 but I wanted double error correcting code.

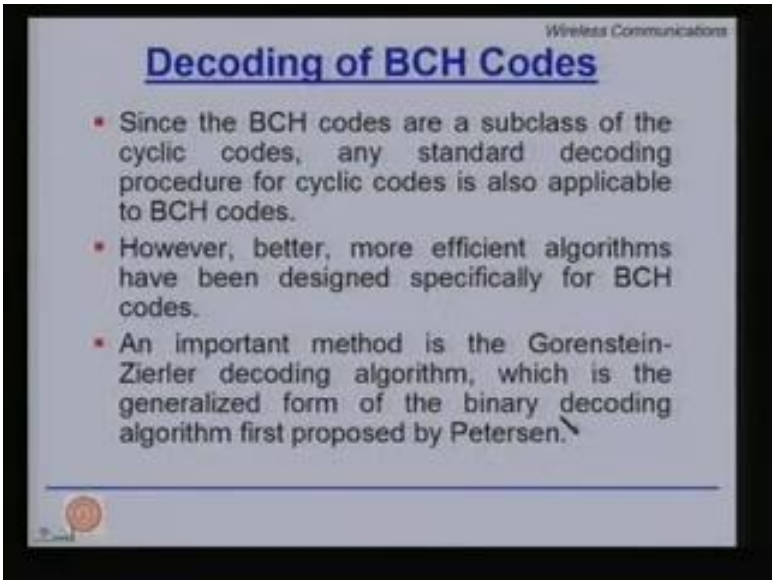
(Refer Slide Time: 00:36:35 min)



It should be able to correct two random errors within the block length n=15. To do so we carry out the same exercise but this time g (x) is LCM f_1 (x), f_2 (x), f_3 (x) and f_4 (x). Again we take the primitive polynomials like this, take the LCM and after the product we find $x^8 + x^7 + x^6 + x^4 + 1$ as the g(x). Again the degree g(x) = n-k is equal to 8 in this case which gives us k =7 because n=15 this means we have been able to design a 15, 7 code. The design distance of this code is 5 and it can be also seen that the minimum distance is also 5. However if you keep on proceeding beyond t =4 you will see that the design distance exceeds the minimum distance that is you will start over designing your code.

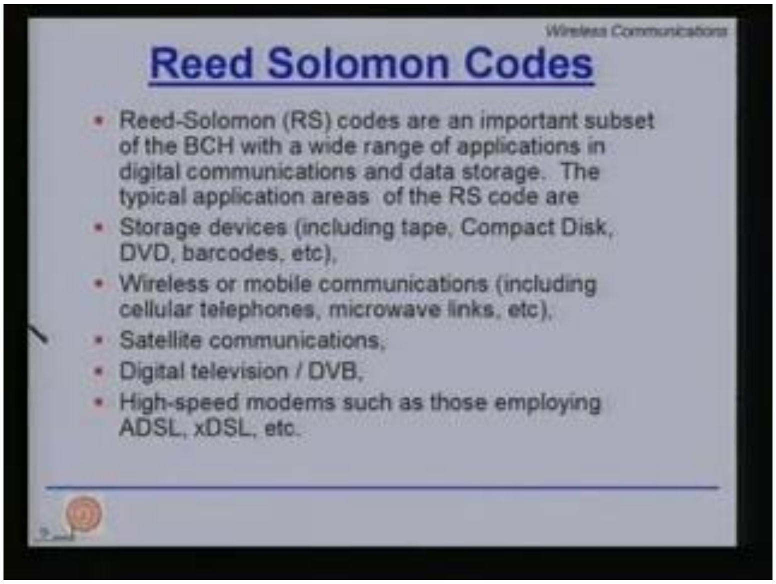
Now the other part of any good coding scheme is how efficient you can make the decoder. Fortunately very fast algorithms exist for decoding BCH codes. It should be remembered that’s since BCH codes are a subclass of cyclic codes which are subclass of linear block codes, any of the previous decoding techniques will work but we have better more efficient algorithms that have been designed specifically for the BCH codes looking at the structures. An important method is the Gorenstein-Zierler algorithm which is the generalized form of the binary decoding algorithm which was first proposed by Petersen.

(Refer Slide Time: 00:37:49 min)

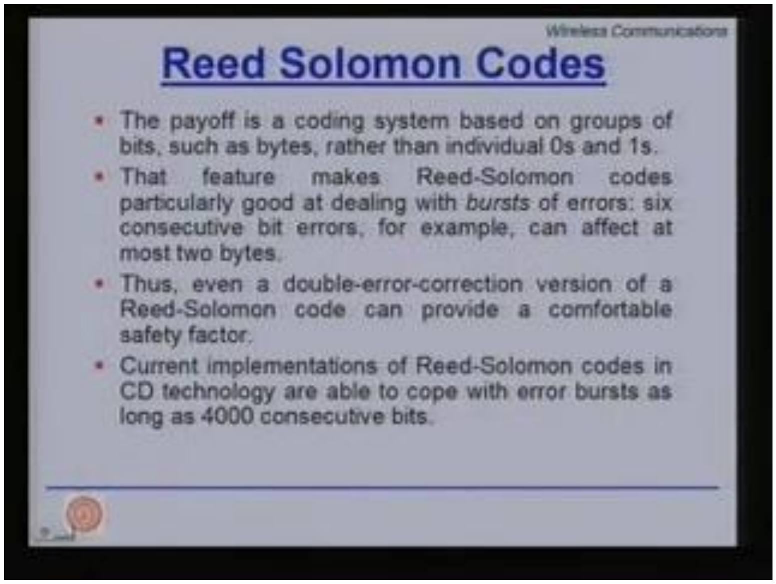


Now let us go into subclass of BCH codes called the Reed Solomon codes or the famous RS codes. RS codes are an important sub set of the BCH codes with a wide range of applications in digital communications as well as in data storage. The typical application areas of RS codes are storage devices including tapes, your music CD's, DVD's, barcodes, wireless and mobiles communications cell phones and microwave links, satellite communications, digital TV, high speed modems. Let us talk more about the Reed Solomon codes, the payoff is a coding system based on groups of bits such as bytes rather than the individual zeros and ones. Here we are actually looking at non-binary codes.

(Refer Slide Time: 00:38:41 min)

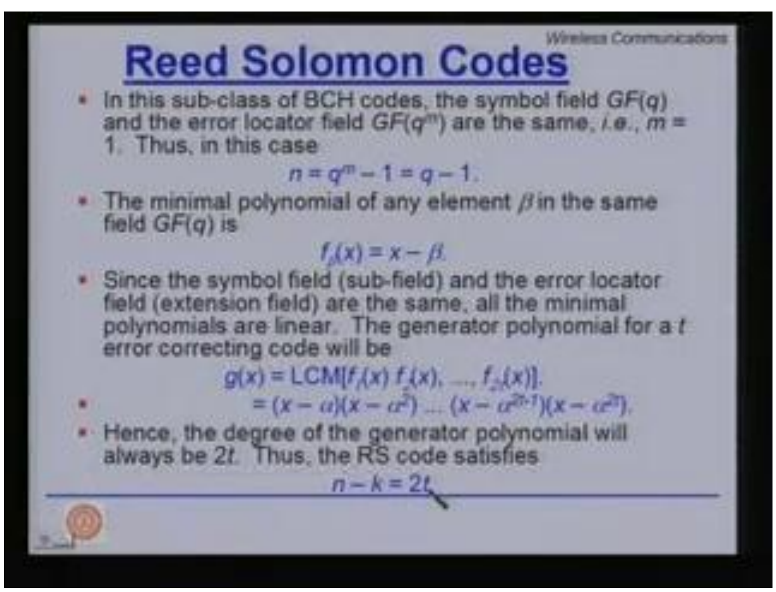


(Refer Slide Time: 00:39:26 min)



If we are now talking about symbols and each symbol represents a set of bits then if you can even make a single symbol error correcting code then eventually you can correct all the burst errors embedded therein because correcting one symbol amounts to correcting all the errors within that symbol but one symbol represents several bits in succession. So we are actually come up with a very strong burst error correcting code naturally. This feature makes the Reed Solomon codes particularly good at dealing with bursts errors, 6 consecutive bit errors for example can affect at most two bytes. Thus even a double error correcting version of a Reed Solomon code can provide a comfortable safety margin that is the inherent advantage. The disadvantage of codes is you have to work in higher Galois fields. The current implementations of RS codes in CD technology are able to cope up with error bursts as long as 4000 consecutive bits.

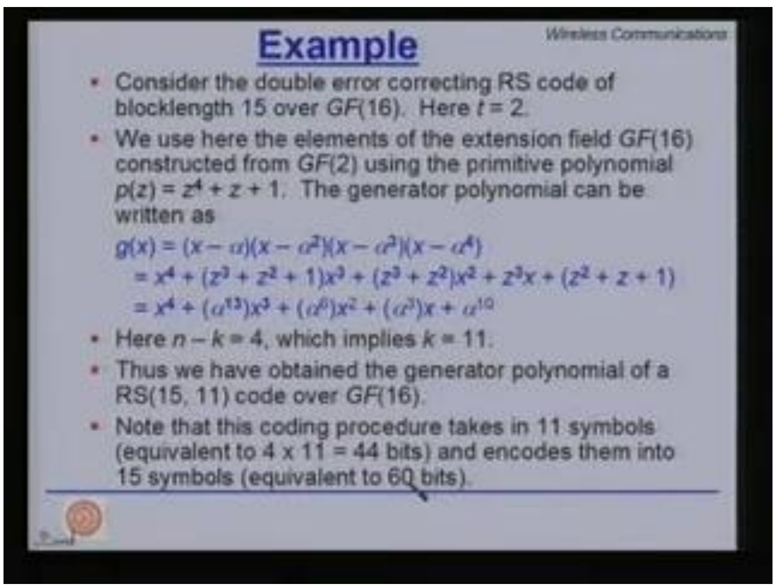
(Refer Slide Time: 00:41:01 min)



So in this subclass of BCH codes, the symbol field GF (q) and the error locator field GF (q^m) are one and the same. That is we are dealing with m =1 so the base field and the extension field that we are talking about for so long are one and the same but in order to make this thing effective I should start with a large value of q. So let's put n =q^m -1 is nothing but q-1 here. The minimal polynomials of any element beta in the same field GF (q) is given by f_{beta} (x) = x- beta. Since the symbol field which is the sub field and the error locator field which is the extension field are one and the same.

All the minimal polynomials are linear, hence the generator polynomial for t error correcting code in this case can be simply written as this, which is true for BCH codes you go from f_1, f_2 up to f_{2T} but here again we have a very simple linear product (x- alpha) (x-alpha squared) so and so forth till (x-$alpha^{2T}$), a really very simple generator polynomial for Reed Solomon codes. Hence the degree of the generator polynomial will always be 2 raise to power t. So you can always comment upon the degree of the generator polynomial, hence n-k is 2 t why, because the degree of g (x) is n-k.

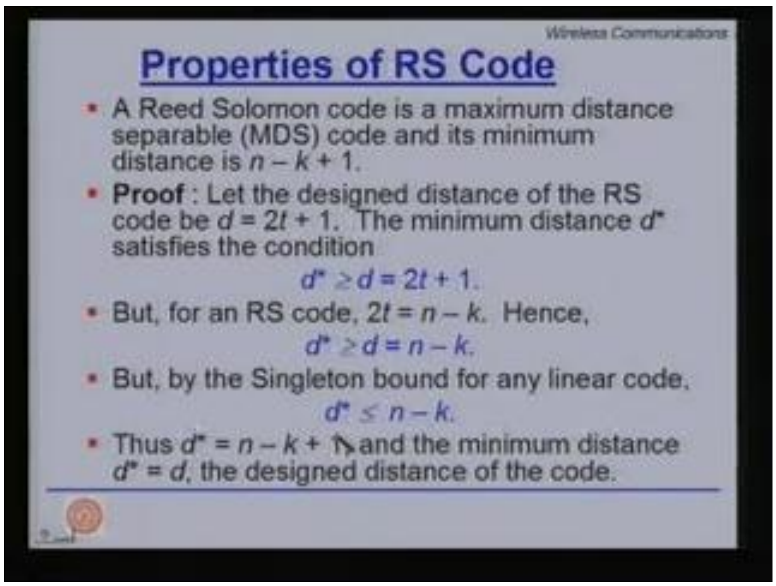
(Refer Slide Time: 00:42:56 min)



Let's look at an example. Consider the double error correcting Reed Solomon code of block length 15 over GF (16). Please note m is 1 so q is 15 so q-1 is 15, as we have seen in the previous case n = q^m -1, q is 16, q-1 is 15. So your block length is 15 that we are talking about. Now we use here the elements of the extension field GF (16) constructed from GF (2) using the primitive polynomial z^4 + z+1. We have done this just before so the generator polynomial can be written as (x-alpha) (x-alpha squared) (x- $alpha^3$) (x- $alpha^4$). Why because t =2 so the maximum power of alpha is 2 t is equal to 4 as simple as this, we immediately have the generator polynomial. We have to expand it out and get a standard polynomial, so if you multiply it out you get this and if you solve this and try to write it in this form you get this one. Please note we are working in GF (16), we have all the elements alpha one, alpha squared, alpha cubed and so and so forth. So you have these representations.

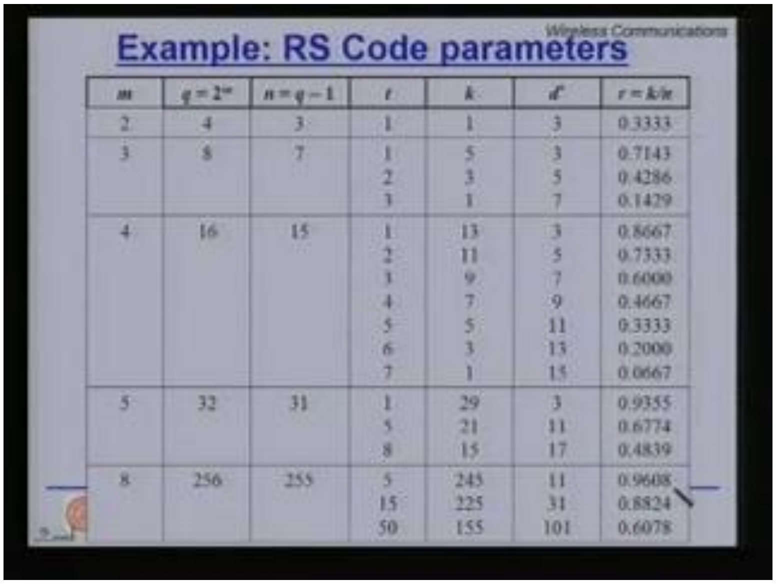
My generator polynomial will have coefficients taken from GF (16) which that the field I am working on but please note by definition g(x) must be monic. So the coefficients of the highest power of x is unity and rest of the coefficients are drawn from GF (16) because these are the elements of each power of alpha is some element of GF (16) since alpha is a primitive element of GF (16). Here n-k is 4 highest power which implies k =11 why, because n =15. Thus we have obtained the generator polynomial of a RS (15, 11) code over GF (16). So clearly this a non binary code, note that the coding procedure takes 11 symbols not bits so it's (15, 11) but it's not dealing with bits its dealing with symbols but we are over GF (16) so each symbol is 4 bits. So we take in 44 bits and throws out 15 symbols which is equivalent to 60.

(Refer Slide Time: 00:45:52 min)



Some properties of RS code, a Reed Solomon code is a maximum distance separable code and its minimum distance is n-k+1. Why? Let the designated distance which is designed for RS code be d =2t+1 then the minimum distance d^* satisfied the condition d^* which is greater than or equal to 2t+1, we know this from basic linear block code theory but for Reed Solomon code we have just seen using the power of g (x) the degree of g (x), n-k=2t. Hence d^* greater than d =2t is replaced by n-k but by singleton bound for any linear code d^* must be less than or equal to n-k, using these two upper bound and lower bound we have been able to say the d^* star is exactly equal to n-k +1 for any Reed Solomon code.

(Refer Slide Time: 00:47:05 min)



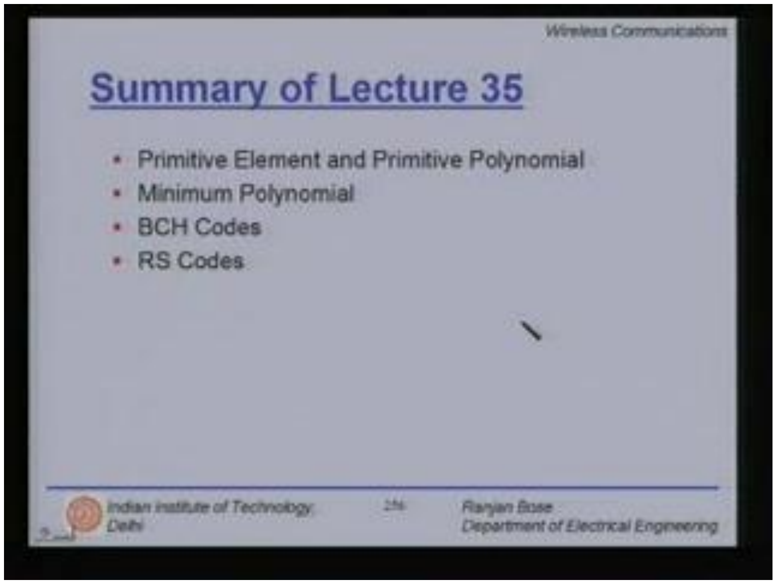
Example: RS Code parameters

m	$q = 2^m$	$n = q - 1$	t	k	d^*	$r = k/n$
2	4	3	1	1	3	0.3333
3	8	7	1	5	3	0.7143
			2	3	5	0.4286
			3	1	7	0.1429
4	16	15	1	13	3	0.8667
			2	11	5	0.7333
			3	9	7	0.6000
			4	7	9	0.4667
			5	5	11	0.3333
			6	3	13	0.2000
			7	1	15	0.0667
5	32	31	1	29	3	0.9355
			5	21	11	0.6774
			8	15	17	0.4839
8	256	255	5	245	11	0.9608
			15	225	31	0.8824
			50	155	101	0.6078

This table shows some standard Reed Solomon code parameters, please note here we have this m, second column we have q which is 2^m and here is the block length. Again this column represents the number of errors that code can correct and here is the k and the d^* minimum distance. Finally the measure of the efficiency of the code which is k over n which is the code rate. So pick for example m =4 and you are talking about q =16 for n =15 if we just talk about a single error correcting code, you have a fairly high code rate. We know that the code rate is less than one but if you go much closer to one, you get a more efficient code.

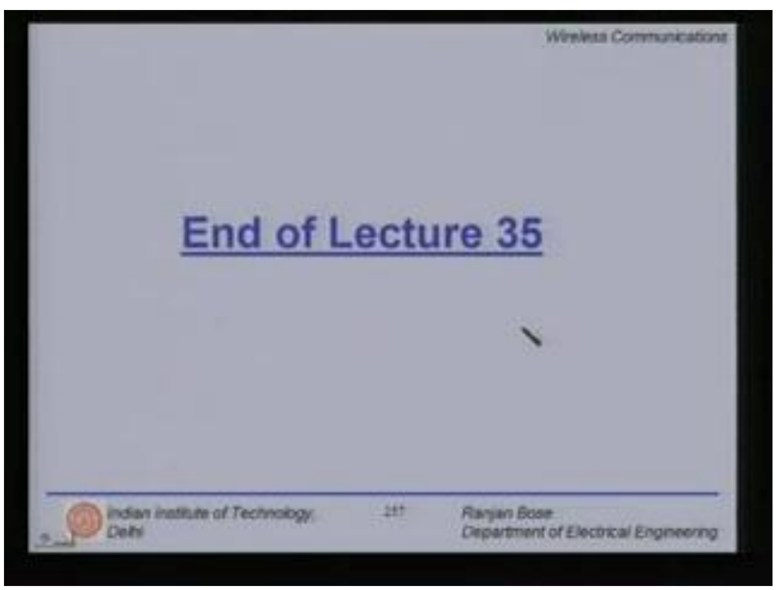
At the same time if you move down the table and if you are at m =8 that is your working at 256 GF (q) is equal to GF (256) and your block length is 255 then you can have t =5 error correcting code but the code rate being very close to 1.96. So you have a five error correcting code and please note this is the symbol, the five symbols can be corrected. So RS codes form a very efficient yet powerful class of cyclic codes, this is the power of Reed Solomon codes.

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



Let us now summarize today's lecture we started off with a mathematical detour and we talked about the primitive element and the primitive polynomial. Then we discussed the definition and the implifications of minimal polynomial, we then discussed BCH codes, how to construct BCH codes followed by the famous Reed Solomon codes.

(Refer Slide Time: 00:49:18 min)



We will conclude our lecture here and in the subsequent lectures will talk about codes with memory that is convolutional codes.
Thank you.