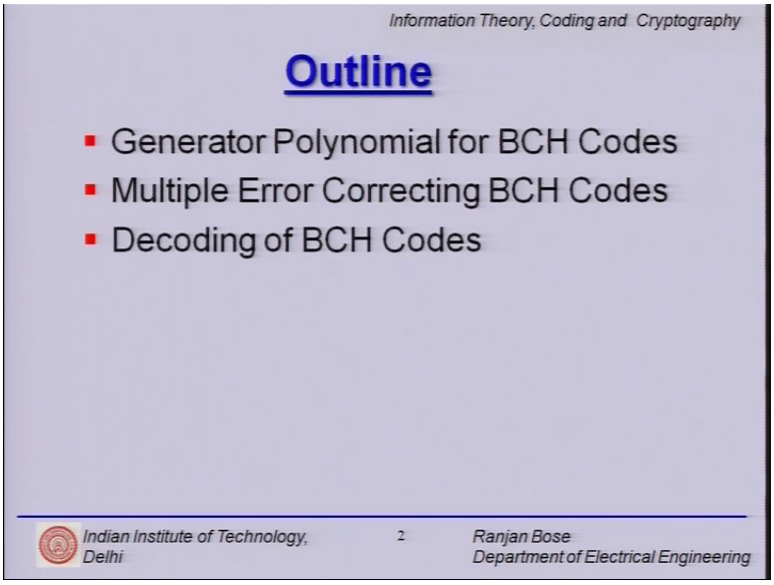
Information Theory, Coding and Cryptography
Dr. Ranjan Bose
Department of Electrical Engineering
Indian Institute of Technology, Delhi

Module – 24
Bose-Chaudhuri Hocquenghem (BCH) Codes
Lecture – 24

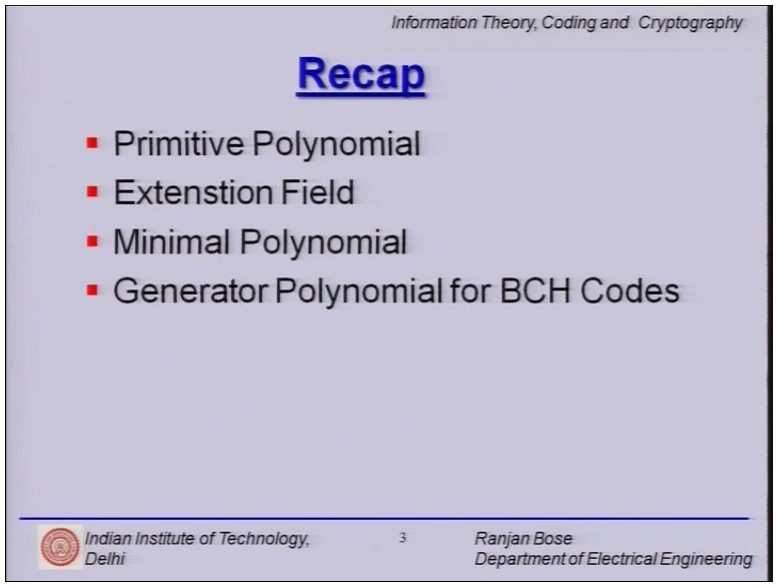
Hello and welcome to our next module on BCH codes let us start with a brief outline of today's talk.

(Refer Slide Time: 00:32)



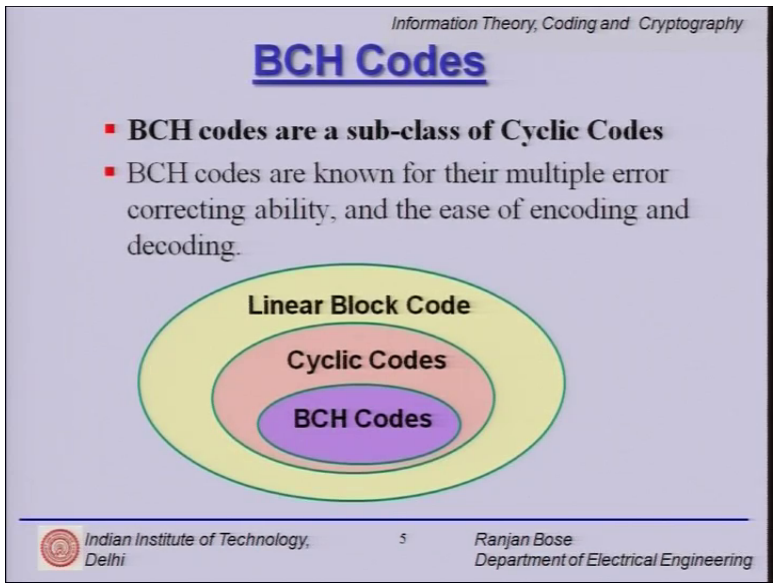
We would start with refreshing our memories regarding the generator polynomial for BCH code and then we would look at how BCH codes can be effectively used to detect and correct multiple errors, because that is what they are famous for and finally, we would talk about efficient decoding of BCH code.

(Refer Slide Time: 00:55)



Let us do a quick recap of to as to what we have done already; we have introduced the notion of the primitive polynomial and then creating an extension field that, we went on to discuss what do we mean by minimal polynomials and then how do we arrive at the generator polynomial for BCH codes.

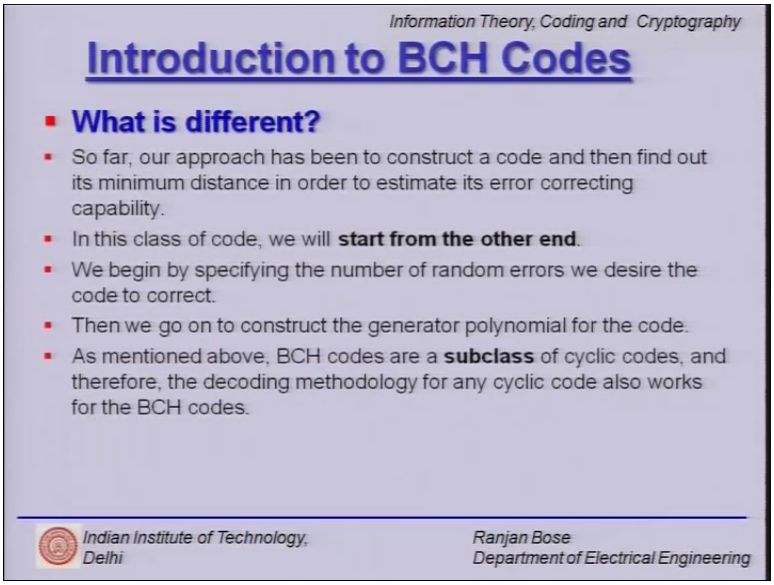
(Refer Slide Time: 01:15)



So, very quickly BCH codes are a subclass of cyclic codes, which are themselves a subclass of the linear block codes. So all the theory we have developed for linear block

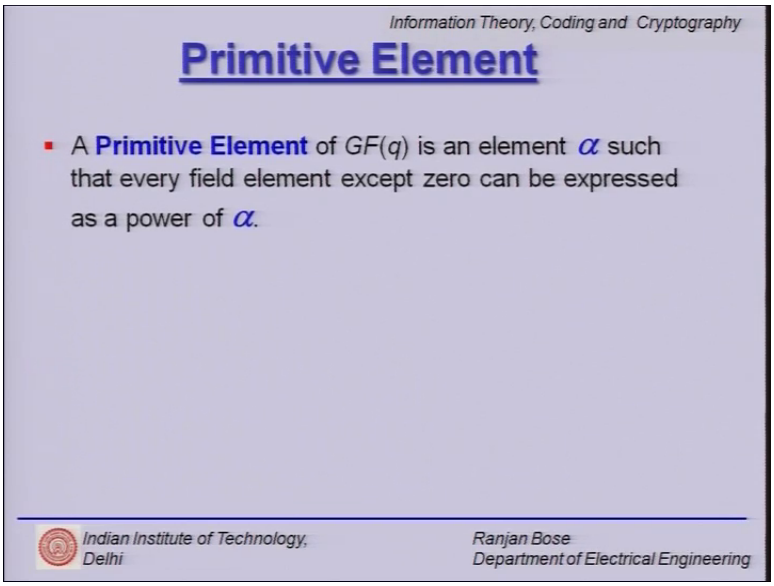
codes and cyclic codes are applicable for BCH codes, at the same time we can have more efficient techniques specifically gear towards cyclic and BCH codes.

(Refer Slide Time: 01:37)



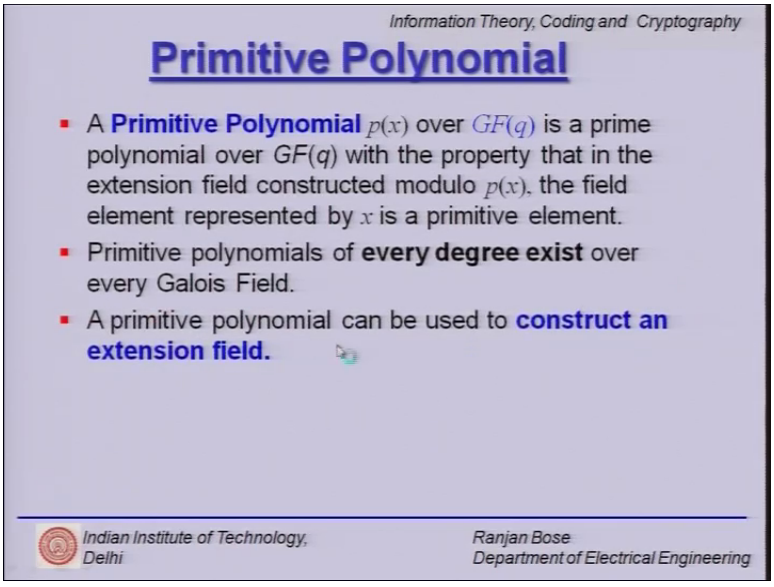
So, please remember the key difference between what we have learnt so far and the BCH codes; in BCH codes we start from the other end we decide how many errors we wish to correct and then we design the particular BCH code for that number of errors being corrected. So t which represents the number of errors, we wish to decode that coupled with and the block length are the starting points for BCH codes; as we mentioned before BCH forms a subclass of cyclic codes.

(Refer Slide Time: 02:16)



Now, in order to do that and also to develop further theory for decoding we must recall what we meant by an primitive element; a primitive element of a Galois field GF q is an element alpha such that every field element except 0 can be expressed as a power of alpha.

(Refer Slide Time: 02:37)

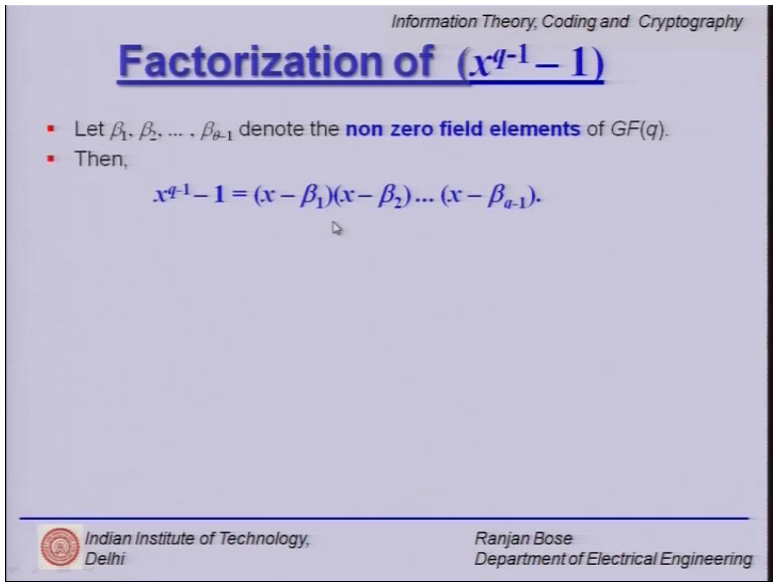


Then we went on to define what is the primitive polynomial; a primitive polynomial p of x over GF q is a prime polynomial, prime if you remember means you cannot factorise and it is also monic; so it is a prime polynomial over GF q with the property that in the

extension field constructed modulo p of x the field element represented by x is a primitive element; so you can construct the entire field using powers of x.

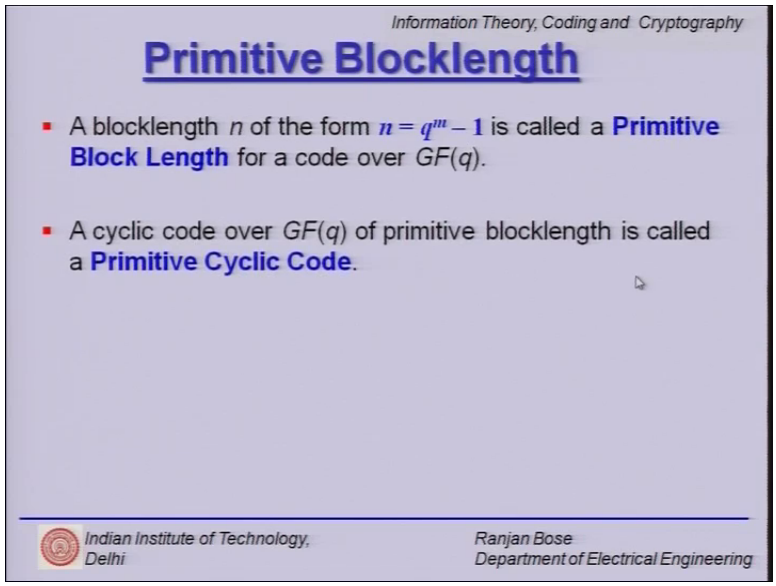
Now, the good news is primitive polynomials of every degree exist over every Galois field and primitive polynomial can be used to construct an extension field.

(Refer Slide Time: 03:15)



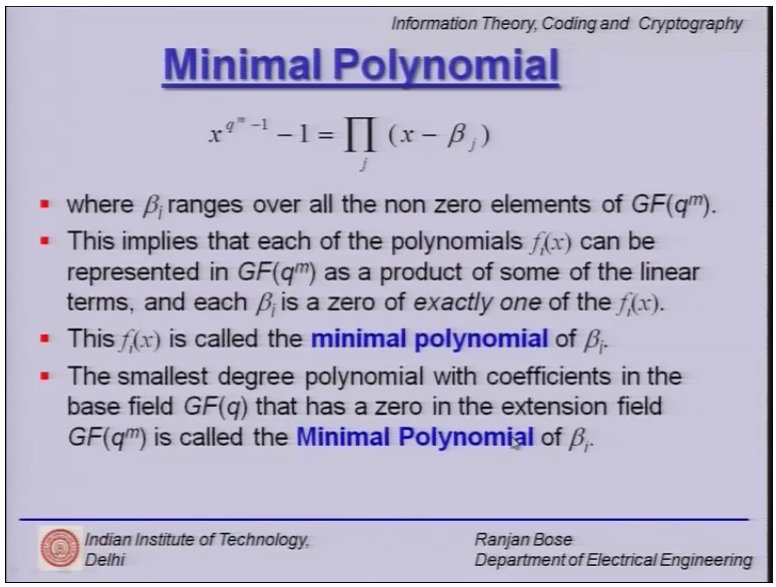
Now why would we doing all of that? That is because we wish to factorise x raise power n minus 1 where n here is q minus 1 so what we have already looked at is if beta 1 beta 2 etcetera are the nonzero elements of GF q then you can write x raise power q minus 1 minus 1 a simply the linear factors of x minus beta i.

(Refer Slide Time: 03:43)



Then we introduce the notion of primitive block length n equal to q raise power n minus where m is some integer and GF q is the field over which we are working; so that is a primitive block, block length for the code over GF q. A cyclic code over GF q of primitive block length is called the primitive cyclic code.

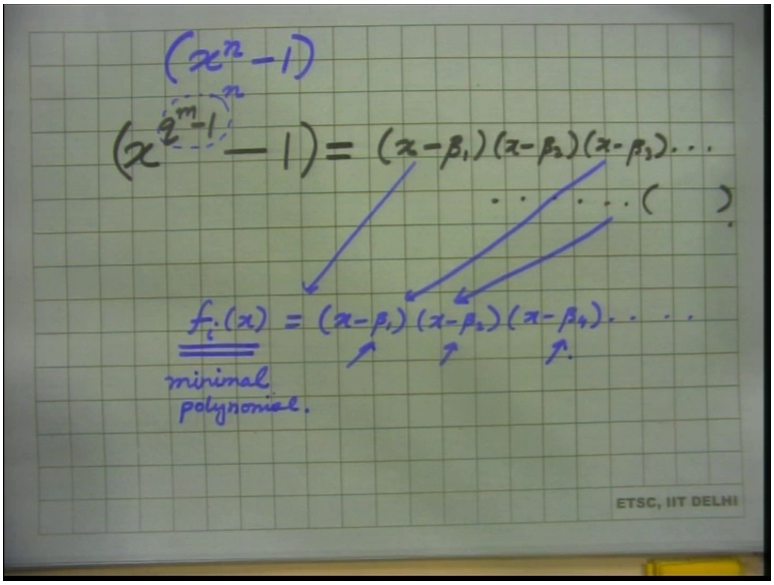
(Refer Slide Time: 04:06)



And we are doing all of this such to factorise are x raise power n minus 1 so we have this x raise power q raise power m minus 1, this is the primitive block length is nothing but the linear factors x minus beta j.

So, we have done all of this earlier and we have found out that that each of the polynomials f i x can be represented in GF q m as a product of some linear terms and each b i is a 0 of exactly 1 of the f i's. So please note that this left hand side is written as factors and some of them correspond to the minimal polynomial right; so if you want to look at it a little bit more carefully we have x is square q m minus 1 minus 1.

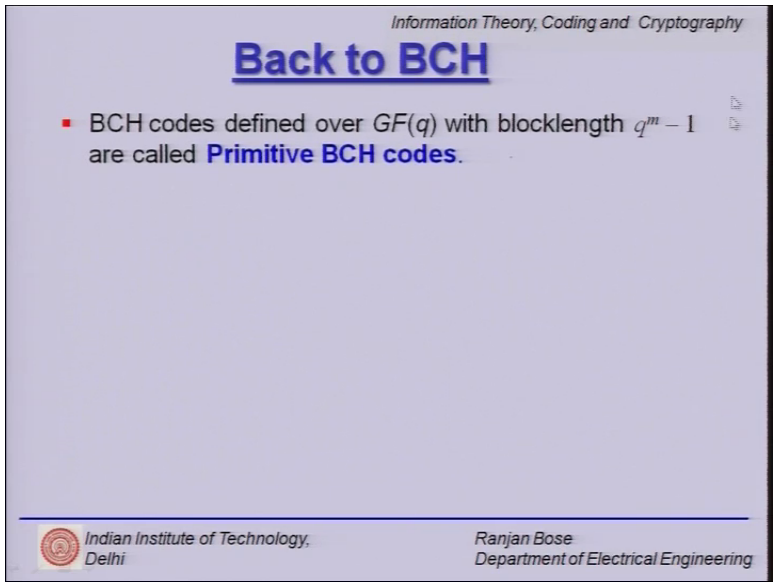
(Refer Slide Time: 05:08)



So, this is very conveniently my n so we are still looking at x n minus 1 factorization, because we are in the domain of cyclic codes. So this can easily be written as a product x minus beta 1 x minus beta 2 and so on so forth right. Now what is interesting is some of them when you multiply together you get so you can have sum of them, so this is the minimal polynomial of this.

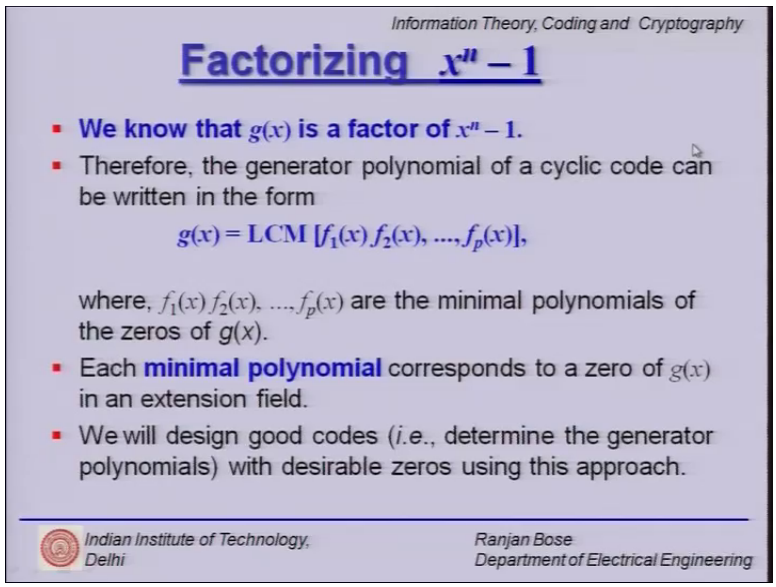
So there are several beta i's which correspondence to the same minimal polynomial and how to choose them we will discuss in the subsequent slides. Now the smallest degree polynomial with coefficients in the base field GF q that has a 0 in the extension field we are looking at the slides again the smallest degree polynomial with coefficients in a base field that has a 0 in the extension field GF q is called the minimal polynomial of beta i.

(Refer Slide Time: 06:55)



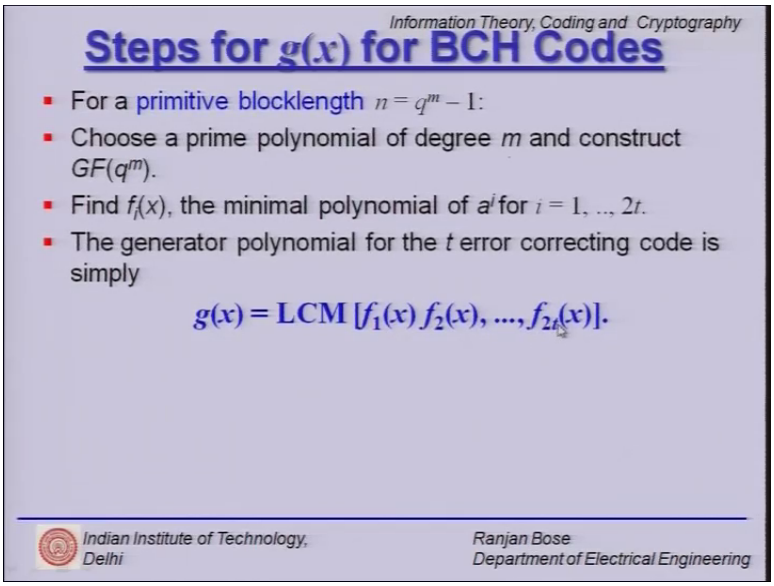
Now, we have talked about primitive BCH codes with block length q, raise power m minus 1 because it is very easily factorizable.

(Refer Slide Time: 07:04)



And it is the factor that we are going to deal with and we went on to define g of x as a factor of x raise n minus 1 and we can take g of x as an LCM of certain number of minimal polynomials; please note each of the minimal polynomial f 1 x f 2 x f n x f p x corresponds to 0 of g of x in the extension field.

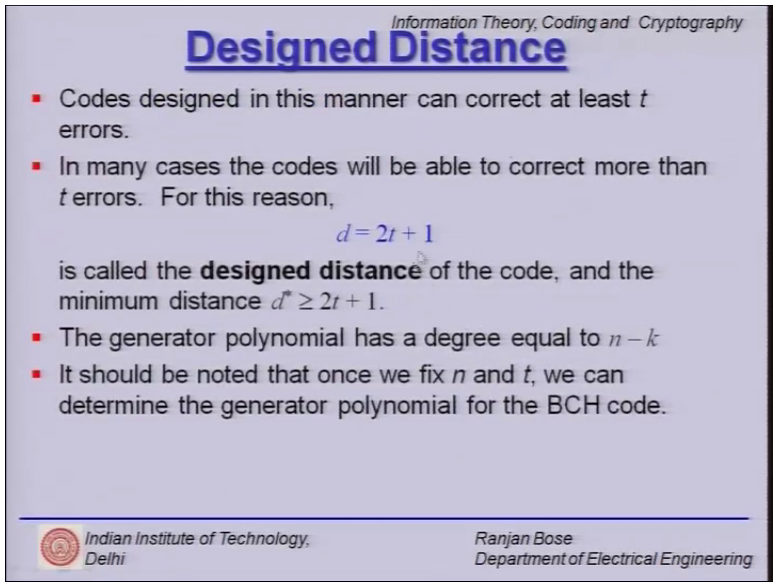
(Refer Slide Time: 07:36)



So, how do we get a g of x for a BCH code; well we first set the primitive block length n equal to q raise power m minus 1 that is the starting point, choose a prime polynomial of degree m and construct GF q m. So you construct the extension field and then find the minimal polynomial of alpha, alpha squared, alpha cubed and so on so forth and finally, you find out g of x is the LCM of f 1 x and f 2 x up to f 2 t x and we did a theory in the last class why 2 t is required so that you can have enough number of equations to exactly correct the required number of errors.

Please note that the design distance is t, so the g of x so designed will guarantee you that will correct at least t errors it can be over designed it can interpreting more than, but this minimum number of errors we can always correct.

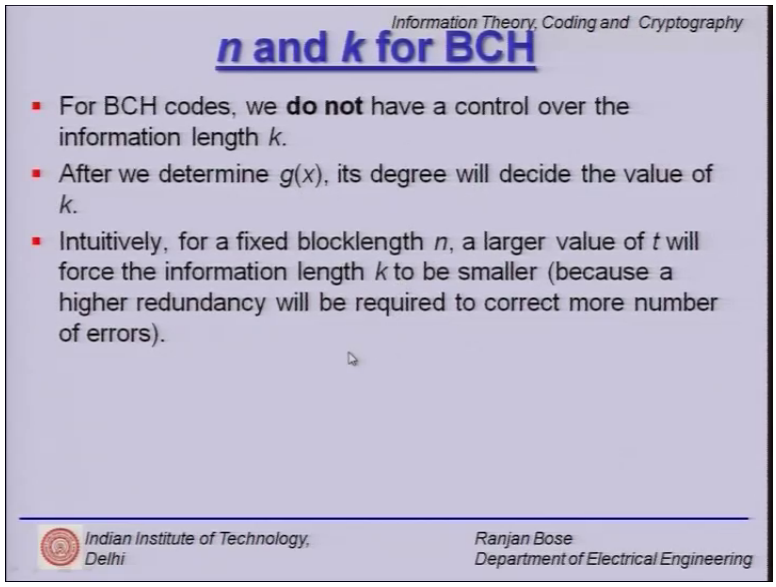
(Refer Slide Time: 08:45)



We will look at some examples and therefore, t is called the designed so d equal to 2 t plus 1 is called the designed distance of this BCH code.

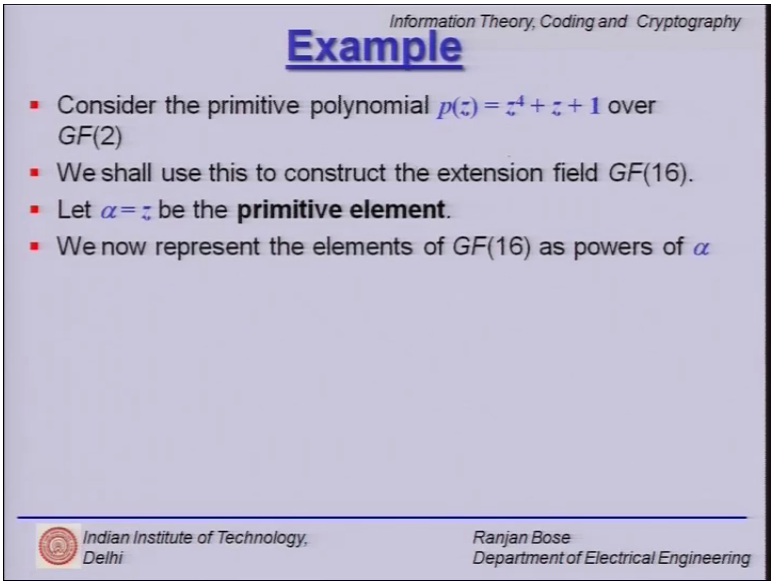
Now, we should note that once we fix n and t we can determine the generator polynomial for the BCH code, because k will automatically get determined, the degree of the generator polynomial is n minus k; so once we know n and once we know the degree of g of x, we must get (Refer Time: 09:17) on to k.

(Refer Slide Time: 09:18)



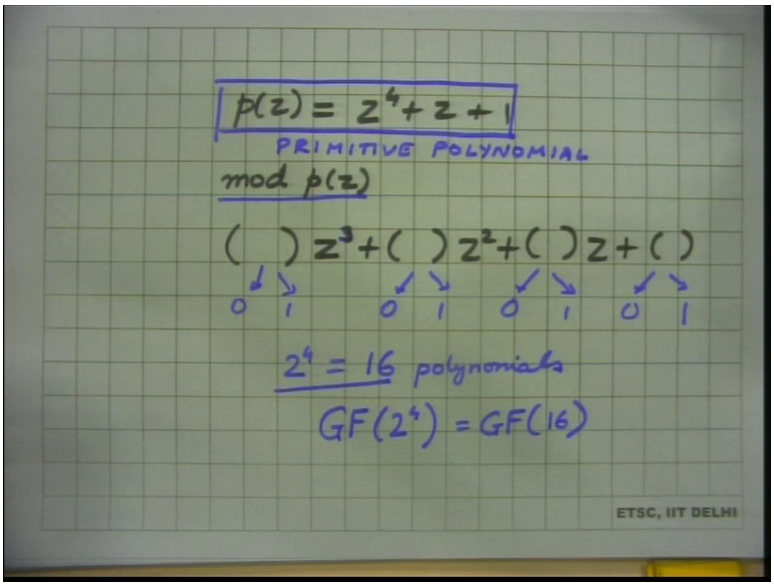
So, for BCH codes we do not have a control on the information length k so once we get g of x that will determine the k.

(Refer Slide Time: 09:29)



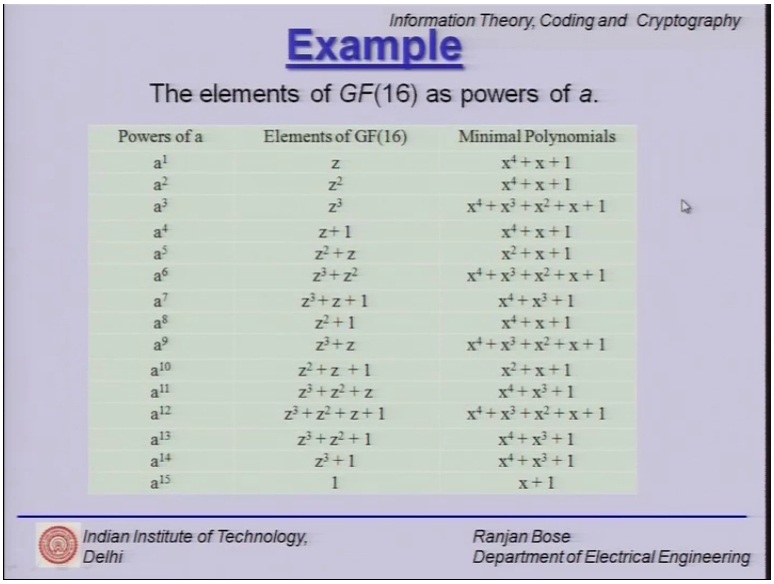
So, let us start with some examples; so consider a primitive polynomial p z is equal to z raise power 4 plus z plus 1 over GF 2 and we wish to construct GF 16 modulo p of z. Now please note if you go back to the drawing board we have p of z equal to z raise power 4 plus z plus 1

(Refer Slide Time: 09:56)



Now we will perform operation modulo p of z, so all the polynomials when we take modulo p of z which will have some coefficient z cube plus some coefficient z squared plus some coefficient z plus coefficient. So please note that since we are taking modulo p of z any remaining polynomial after taking modulo p of z can have the highest degree 3. So we have here a 0 or 1 0 or 1 0 or a 1 0 or a 1 coefficients, so you have 2 raise power 4 is equal to 16 unique polynomials, and these will represent your 16 elements over GF 2 raise power 4, but the only way we would be able to construct this is going with modulo p of z because p of z is the primitive polynomial.

(Refer Slide Time: 11:37)

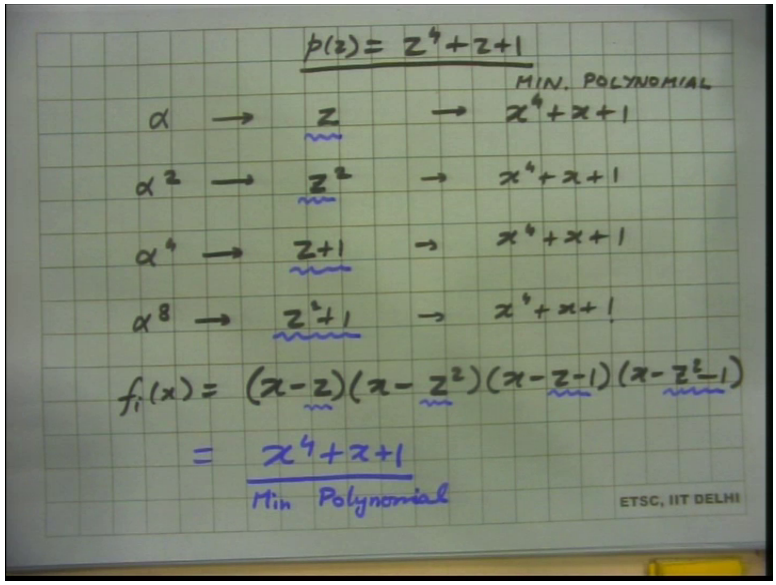


So, let us complete this exercise so we go back to our slides and if we write down the powers of alpha a then alpha is equal to z, alpha squared is equal to z squared, alpha cube is equal to z cube and alpha 4 is z raise power 4, but it has to be taken modulo p z which is z raise power 4 plus z plus 1. So if the moment you divide z raise power 4 by z raise power 4 x plus 1 you end up having z plus 1 and so and so forth you can continue and you can construct the 16 nonzero elements of GF 16.

Now, if you take alpha, alpha squared alpha 4 right and take the elements and you multiply x minus z into x minus z square into x minus z minus 1 and so and so forth which are the conjugates you end up with the minimal polynomials. So for example, let us take this example a little forward so we go back to the drawing board so you have

alpha corresponding to z and you will realise that I have written minimal polynomial x raise power 4 plus x plus 1.

(Refer Slide Time: 12:52)



Similarly alpha square is z squared because we are taking p z is equal to z raise power 4 plus z plus 1 and again it has the same minimal polynomial why because they are conjugates, but why should we stop here we look at alpha raise power 4 and when you take it modular p of z you end up with z plus 1 again and then you go back and you look at alpha 8 and alpha 8 is z square plus 1 same, but the moment you go alpha 16 it reverts back so these are the 4 elements.

Now, if you want to look at it your f i x is nothing, but x minus z into x minus z squared into x minus z minus 1 into x minus z square minus 1. Please note what we have done, we have taken the first element, second element, third element, fourth element and we stop here because going beyond we repeat back. Now if you multiply this out magically all the z's will cancel out and you will left with x raise power 4 plus x plus 1 which is your minimal polynomial. And similar exercises can be done for alpha cubed, alpha 6 and so and so forth and so and so forth till you discover all the minimal polynomials so it takes a.

Student: What is the combination behind all the z considered?

So, the minimal polynomial will have coefficients in the base field that is how a definition comes and we had shown it in a theorem in the previous class therefore, z is an element of the extension field, but these field is GF 2 so the coefficients can be 0 or 1. So therefore, it is a beauty of maths that it will cancel out and you can show that the minimal polynomial will ultimately have coefficients in the base field.

Student: Sir.

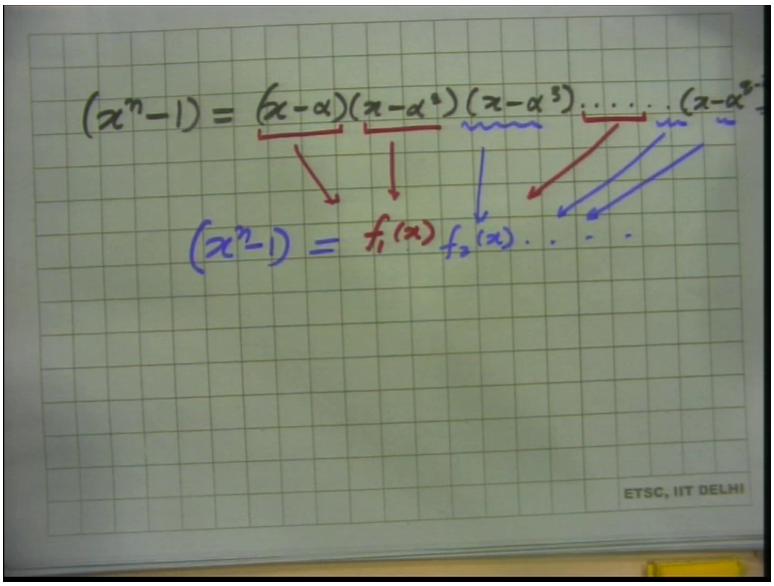
Yes.

Student: If there are going to write minimal point polynomial looking directly (Refer Time: 15:54).

Question that been is asked is; can we write the minimal polynomial directly looking at p z? The answer is no. There is a standard way to solve for minimal polynomials; you first create the extension field as you have done in the table we showed earlier and then from the table you have these elements, now we also prove last time that certain powers of alpha when you have linear factors then you will end up having the products leading to the minimal polynomial.

So this is basically the so if you want to continue this logic forward you know that x raise power n minus 1 is nothing, but x minus alpha x minus alpha squared x minus alpha cubed so and so forth right till all the elements are there and that is the factorization i is closed you can just continue doing this

(Refer Slide Time: 16:43)

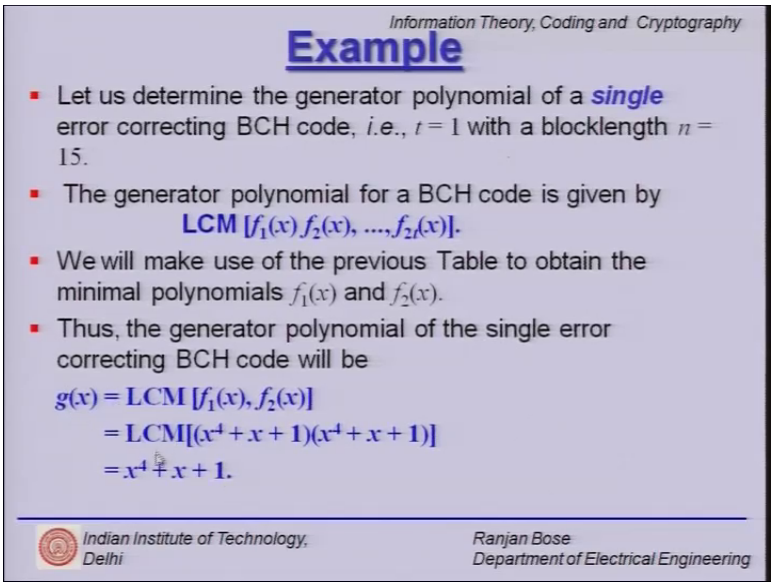


But the interesting part is if you take sum of them and multiply you get f 1 x, if you take some other ones here and multiply them you get f 2 x and so and so forth and they are still equal to x raise power n minus 1. So in fact, we are factorising the left hand side elegantly using all the coefficients from the base field.

So, we continue with our example so we go back to our slide and we have suddenly a nice table of GF 16 with consecutive minimal polynomials and in little bit of observation says that look at alpha cubed right. The minimal polynomials for alpha cubed is the same as alpha raise power 6 is the same as alpha raise power 9 is the same as alpha raise power 12. So these are the conjugates so exhausted 1, alpha raise power 2, alpha raise power 3, alpha raise power 4, then you look at alpha raise power 5.

So the minimum polynomial of alpha raise power 5 should be the same as alpha raise power 10 right and so and so forth. So very soon you will have those conjugates and you can multiply them out to constitute the minimal polynomial. Now needless to say here there only the highest power is x squared so only 2 of the elements have been multiplied out right, so once we have this table we are ready to go back to our BCH codes.

(Refer Slide Time: 19:04)

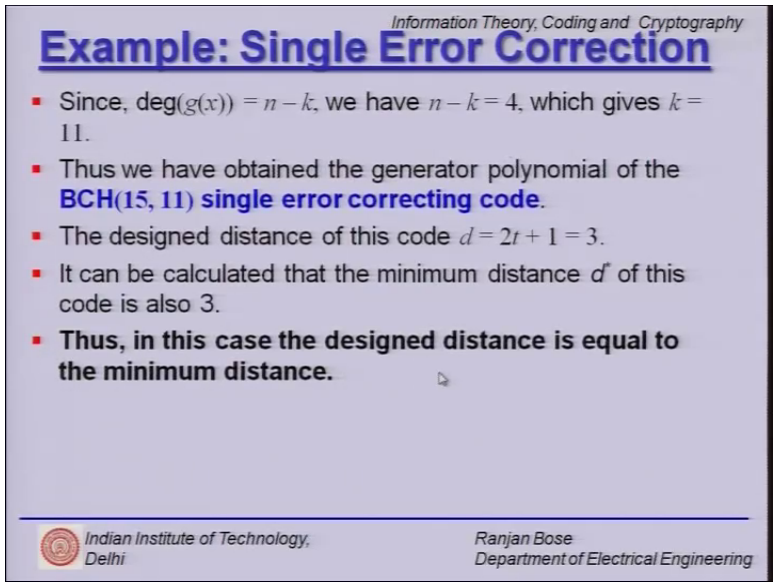


So, let us say we are interested in finding out a single error correcting code; now single error correcting means t is equal to 1 and let us fix n is equal to 15 because without fixing the n we cannot proceed the two things we need to fix is n is equal to the block length and t is equal to the number of errors you need to correct and their we will design our code.

What is designing a code finding the g of x that is what we mean by getting a BCH code, but we already have the; for it G of x is nothing, but LCM of f 1 x, f 2 x; what are these? These are the minimal polynomials when do you stop f 2 t, but here our case t is equal to 1 so we stop at f 1 f 2. So f 1 x and f 2 x are to be multiplied from the previous table, so if you go back to the previous table we have the f 1 and f 2 right here and now you understand the value of the LCM because they are identical here so we do will only consider them only once right.

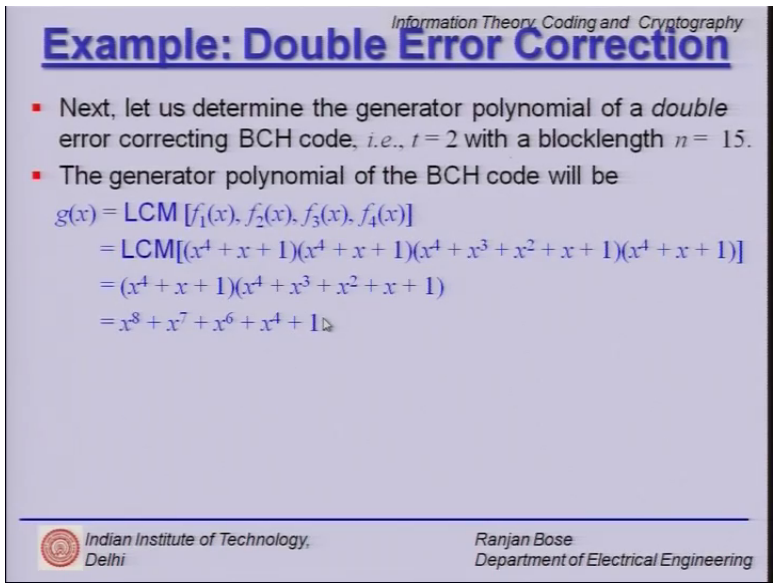
So, we take LCM of these two and so any one of them and the answer is g of x is nothing, but x raise power 4 plus x plus 1 that happens to be my generator polynomial. Now n minus k is 4 and n is already 15 consequently k must be 11, so continuing that we will get k equal to 11.

(Refer Slide Time: 20:35)



So, we have suddenly have developed first BCH code single error correcting 15 comma 11 BCH code which is a binary code ok. So the design distance is 3 and you can verify using the generator polynomial that it is indeed the minimum distance is 3 leading to the fact that it can correct a one error code, thus the minimum distance is equal to the design distance.

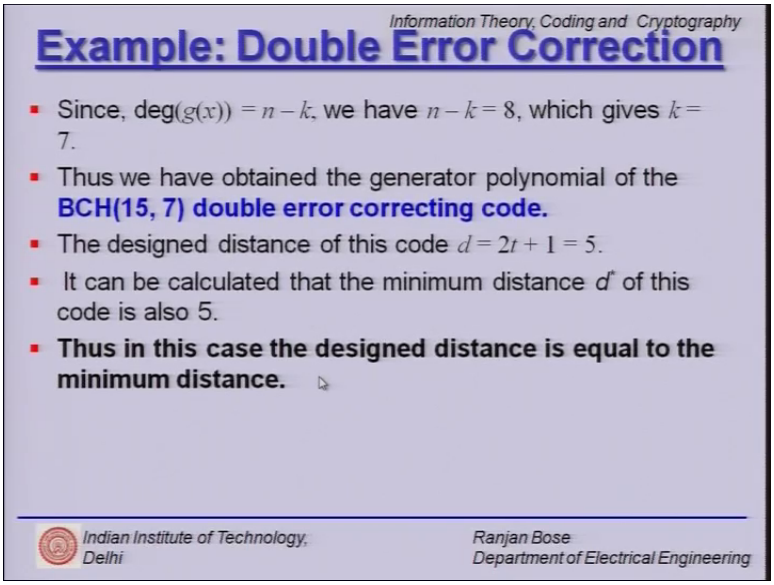
(Refer Slide Time: 21:11)



Now, we do not have to do much hard work if we have to graduate to the double error correction double error correcting code BCH code all we have to do is now increase the

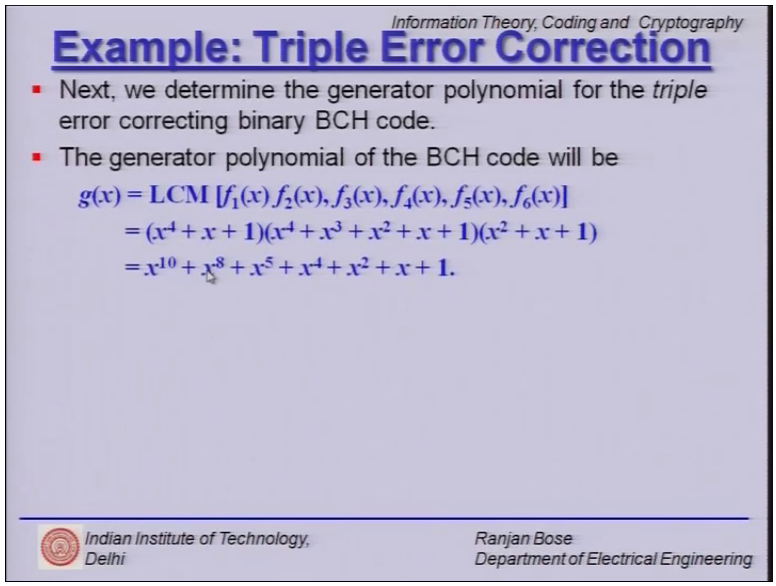
value of t is equal to 2 and now the generator polynomial is nothing, but the LCM of f 1 x, f 2 x up to f 2 t, but 2 t is 4 so just the first 4 minimal polynomials and take the LCM. So from the table we have written down this thing again LCM comes handy because f 1 x, f 2 x and f 4 x are identical and so we will choose only one of them and multiply it with the second guy and therefore, we are left with this following g of x. So we have designed our second BCH code in a (Refer Time: 21:59) because we simply have the value of t is equal to 2.

(Refer Slide Time: 22:03)



So, what do we conclude the degree of g of x is n minus k, but here n minus k equals 8 consequently k is equal to 7 therefore, k gets determined our BCH code is now 15 comma 7 which is a double error correcting code ok. So if the design distance is 5 and you can also do the hard work and look at the either the parity check polynomial or you can look at the generator polynomial, either which way and find out what is the minimum distance, it is also equal to 5, again in this case the design distance equals the minimum distance.

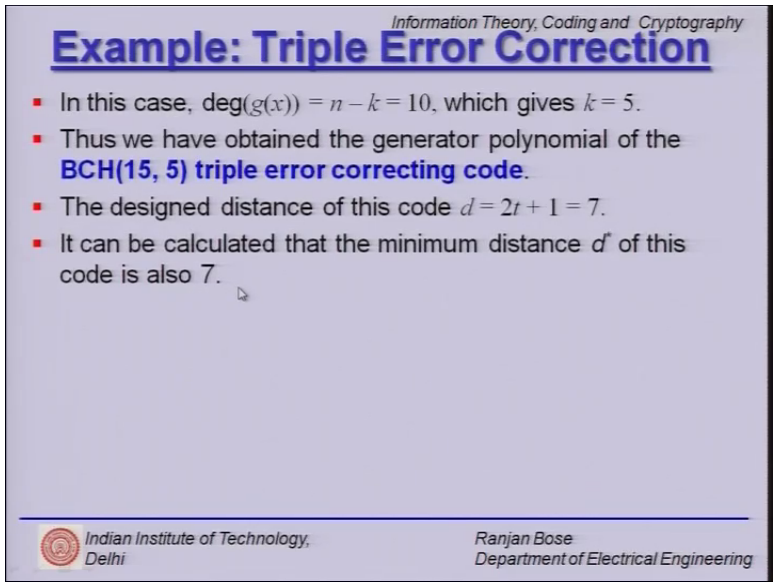
(Refer Slide Time: 22:46)



Since this is a pretty mechanical process we will go ahead and continue to the triple error correcting code this also shows how versatile is designing BCH code. Because for example, a certain application requires us to put in a much stronger code we can immediately go back and come up with the stronger triple error correcting BCH code. The g x is the same process, but this time the first six minimal polynomials need to be picked up once we do that and find out the LCM we get the following.

What is to be noted is the more number of polynomials we pickup the higher is the degree of g of x, but higher degree means n minus k is larger n minus k, but n is fixed so k must go down ; that means, my redundancy is being added so g of x becomes a stronger polynomial generator polynomial at the cost of higher overheads.

(Refer Slide Time: 23:52)

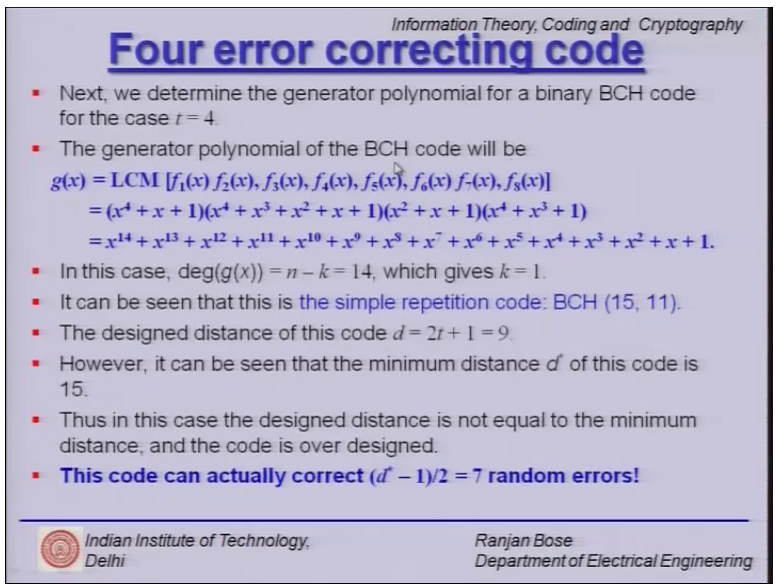


Again k is equal to 5 so we have BCH 15 comma 5 which is a triple error correcting code again one can check that it is indeed the minimum distance is 7 leading to number of errors to be.

Student: (Refer Time: 24:09) always happens that minimum designing distance will be equal to (Refer Time: 24:13).

So, question is being asked whether the design distance is always equal to the real minimum distance; the answer is no as we will see in the next slide.

(Refer Slide Time: 24:24)

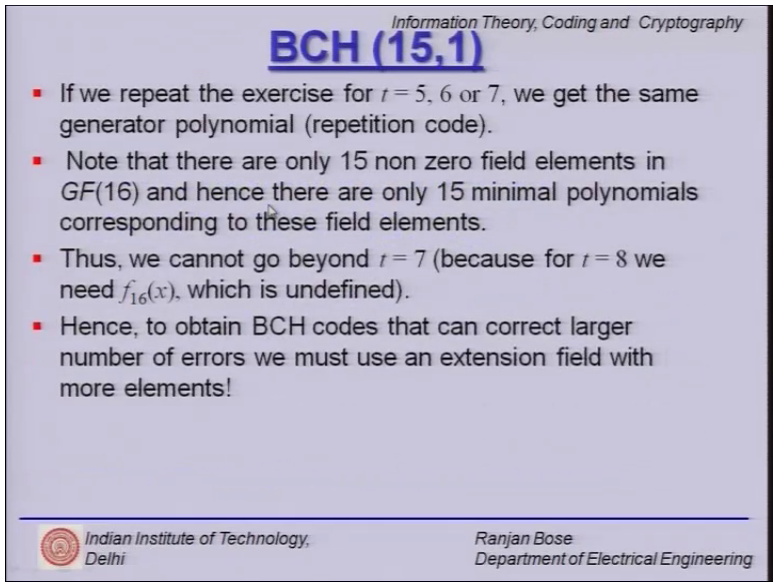


Let us go to 4 error correcting code because it is so easy to go from one type of codes each one is are independent BCH code that we have come up with. So we should appreciate the beauty with which we can immediately graduate to the larger and larger error correcting capability of BCH code. So we are getting greedy we have gone to t is equal to 4 and we quickly go to the first 8 minimal polynomials. So you only do the hard work once in terms of creating the table and once if the table ready we go all the way and once we multiply it out and get the LCM we get a very interesting g of x it contains all the x, x raise power 14 plus x raise power 13 plus x raise power 12 so and so forth up to x plus 1.

Now, in this case the highest power is 14 n minus k equal to 14 consequently k is equal to 1 because this is the n is equal to 15 block length code. So we have come up with a 15 comma this is misprint it is a 15 comma 1 and this is nothing, but a repetition code so this is a 15 comma 1 repetition code and in this case the design distance is nine because we wanted a 4 error correcting code.

But if you go ahead and look at it carefully this one has a minimum distance of 15 consequently it can correct much much more than what we have designed for this time we have over designed this 15 comma 1 code right; so actually it can correct up to 7 errors; that means, our substituent effort to go from 4 error correcting code to 5 6 or 7 we will get the same answer because we have run out of minimal polynomials right.

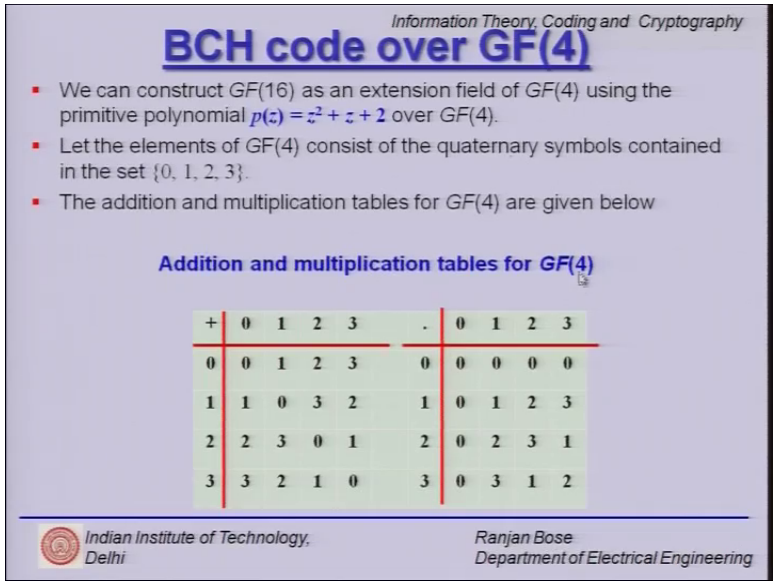
(Refer Slide Time: 26:27)



So for t is equal to 5 6 or 7 right we have the potential we will always end up getting the repetition code because if already multiplied out the minimal polynomials available to them. So BCH code can correct larger number of errors than we have designed for; so if you have to do better we need to go to a Galois field which is higher.

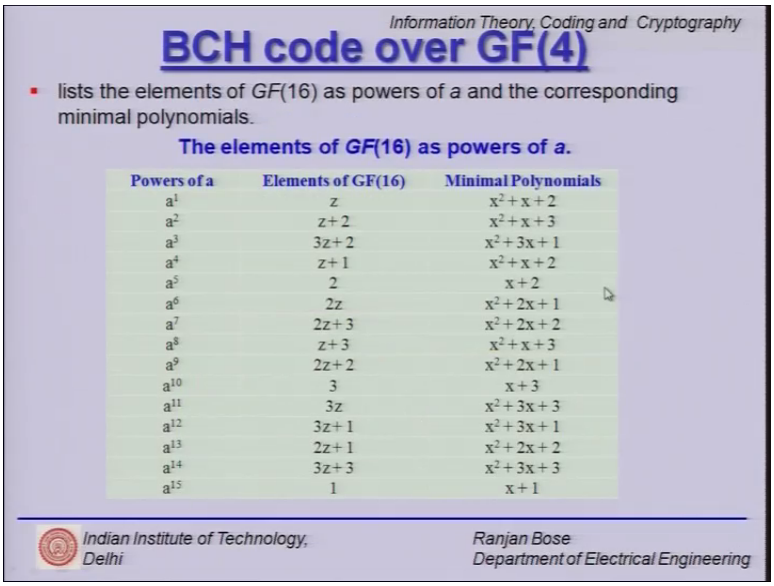
So, so far this 15 comma 1 repetition code, can correct 4 5 6 or 7 errors.

(Refer Slide Time: 27:13)



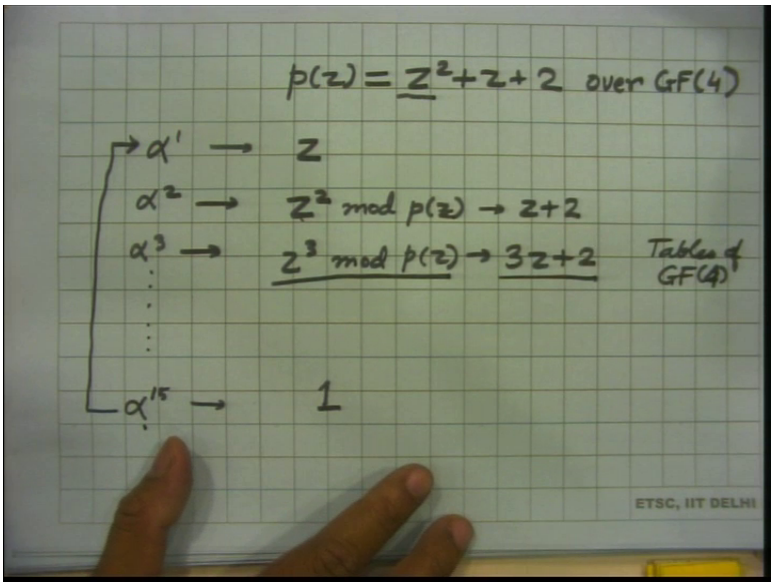
So, let us now extend this philosophy over a non binary code, so we are looking at a BCH code over GF 4, now how do we do this we are going to use a primitive polynomial p z is equal to z square plus z plus 2 over GF 4. So since it is over GF 4 it can have coefficients 0 1 2 and 3 2 is already been used so 1 and 2 are being used here and the extension field will be GF 16. So just to recollect we will do a arithmetic over this GF 4 which is the addition table and multiplication table given right here please note GF 4 does not follow modulo 4 arithmetic because it is not a prime.

(Refer Slide Time: 28:07)



So, we do the same exercise, but this time we have again elements of 16 GF 16 constructed modulo p z z squared plus z plus 2 so let us go back to a drawing board and we have this time p z is equal to z squared plus z plus 2 and note it is over GF 4.

(Refer Slide Time: 28:28)

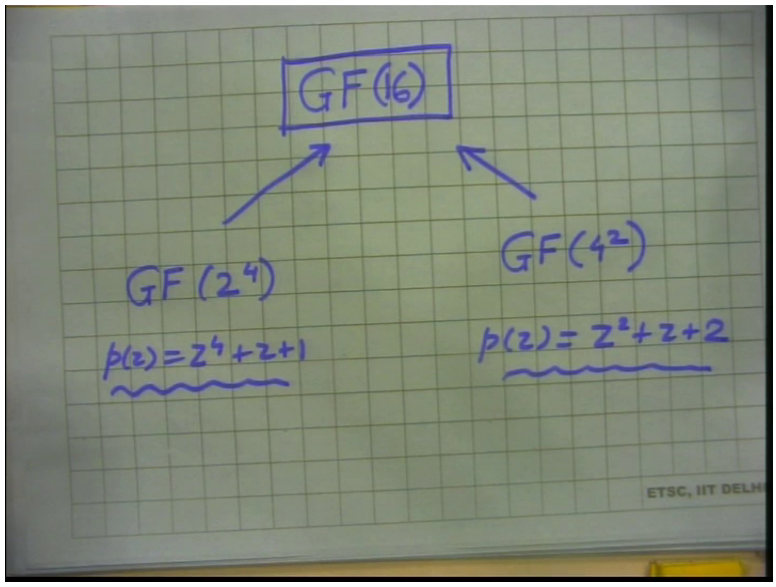


Now, if you see that the elements which are taken always modulo p z so if you have alpha raise power 1 this is nothing, but z no problems alpha squared is nothing, but z squared modulo p z. The moment you have modulo p z here z square highest power you divide it and you are left with z plus 2 and then you have alpha cubed z cubed again you

have to take modulo because the degree of z is higher than z squared. So you take modulo z square and you get this time 3 z plus 2; now in order to execute this operation we have used the tables of GF 4 the multiplication and addition tables of GF 4 has been used to obtain this therefore, we get this and the coefficients lied between 0 1 2 and 3 and so and so forth. You can continue up to alpha 15, which will give you 1 and then it will repeat again if you keep multiply.

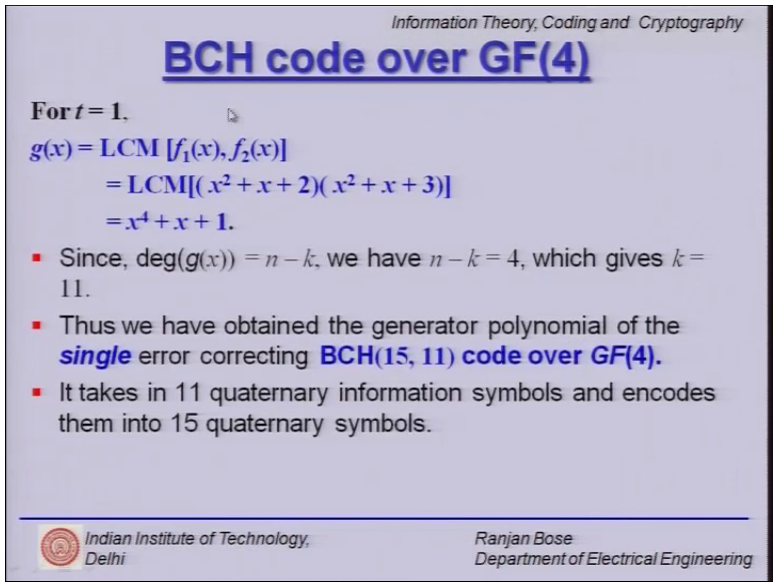
So, all the powers of alpha can give you the 16 elements, but please note that the elements obtained in this table look different than the once we obtained using GF 2 so GF 16 can be constructed in both ways.

(Refer Slide Time: 30:26)



We have constructed GF 16 using 2 methods either you take GF 2 and to prime power or you have GF 4. So you can construct this of course, you have used here p z is equal to z 4 plus z plus 1 and here you have used p z is equal to z square plus z plus 2. So we have used different kind of primitive polynomials to construct GF 16, but GF 16 is unique so the tables the addition and multiplication tables formed either which way will show identical properties, but we go back to our slide and now we look at the elements of GF 16 as powers of alpha and you have again the corresponding minimal polynomials. We do a quick observation alpha raise power 1 has this minimal polynomial alpha squared same minimal polynomial alpha 4 same right and so and so forth you look at alpha 8 with the same minimal polynomial.

(Refer Slide Time: 31:51)

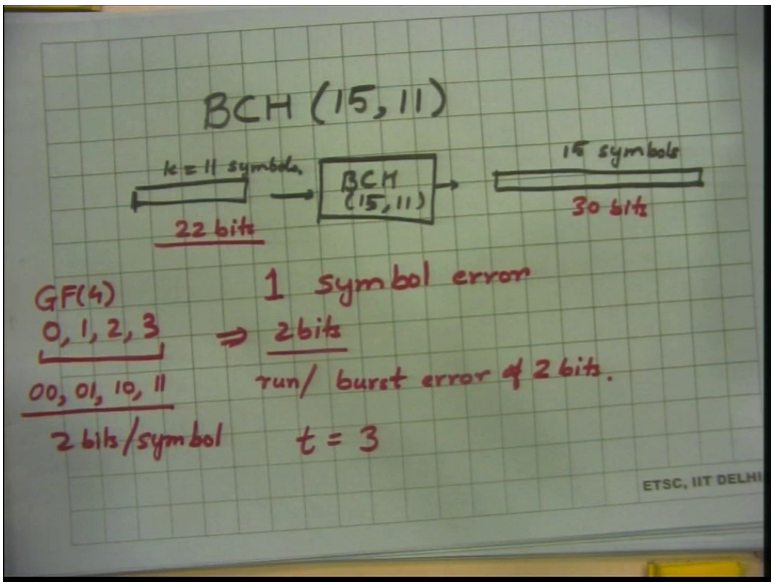


So, consequently you can now start looking at the error correcting capability of this non binary BCH code please note in the early example we had a binary case here the codes will be designed over GF 4.

So, we are again back to our square 1 we would like to have a single error correcting code t is equal to 1 very mechanical g x is LCM of f 1, f 2, but please note this time the minimal polynomials are changed, they take coefficients from GF 4 please note minimal polynomials have coefficients 0 1 2 and 3. So I multiply it out and I get g of x happens to be x 4 plus x plus 1 you can multiply it out and use the tables of GF 4.

Again degree n minus k I am looking at n is equal to 15 15 minus 4 is 11, so I have got a BCH 15 comma 11 code over GF 4 where a similar 15 comma 11 code over GF 2 also, which is a single error correcting code, but the difference is that this takes 11 quaternary symbols and encodes are into 15 quaternary symbols. So what is the big deal let us look at this example a little bit more carefully, so this is a BCH 15 comma 11 code.

(Refer Slide Time: 33:35)

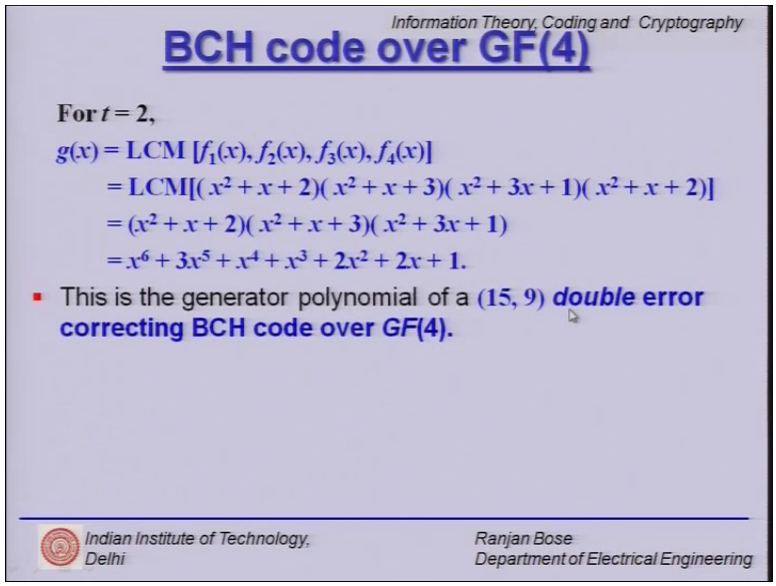


This block BCH 15 comma 11 always takes in k is equal to 11 symbols and it encodes into 15 symbols, please note this symbols belong to GF 4.

So, consequently you can actually use 22 bits to represent this and here you have got 30 bits to represent the symbols, because remember GF 4 can be represented using the 4 symbols 0 1 2 and 3 and they can be equal valiantly written as 0 0 0 1 1 0 1 1. So I can have 2 bits per symbol, so what is the difference is it is a single symbol error correcting code, so it corrects 1 symbol error, but 1 symbol contains 2 bits implies 2 bits. So suddenly bitwise we are correcting 2 bits, but if both the bits are in error then also it can correct because 1 symbol is a 1 symbol so it means that if I have a run or a burst error of 2 bits I can correct it, because for me if I am sending 22 bits and each of the 2 consecutive bits are paid together to form 1 symbol then I am sending 11 symbols.

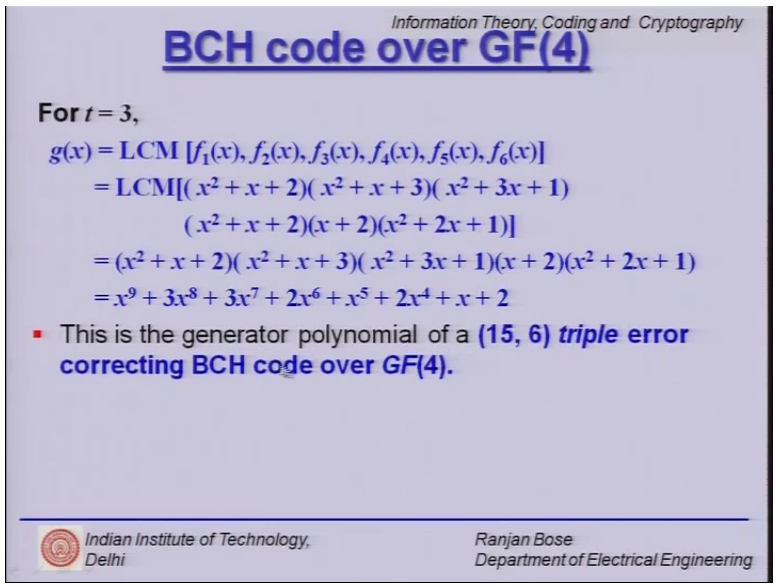
If 1 of the symbol is wrong I mean 2 bits can be wrong then we can always correct 2 bits, but they can be consecutive 2 bits they do not have to be random 2 bits consequently it is actually a bit burst error correcting bit of 2 number 2 bits. So for extend this logic and if I say t is equal to 3; that means, 3 symbols can be corrected and so and so forth, but will come to this later let us continue this example so we go back towards slides; so we have for t is equal to 1 we have already a g of x over g of 4.

(Refer Slide Time: 36:32)



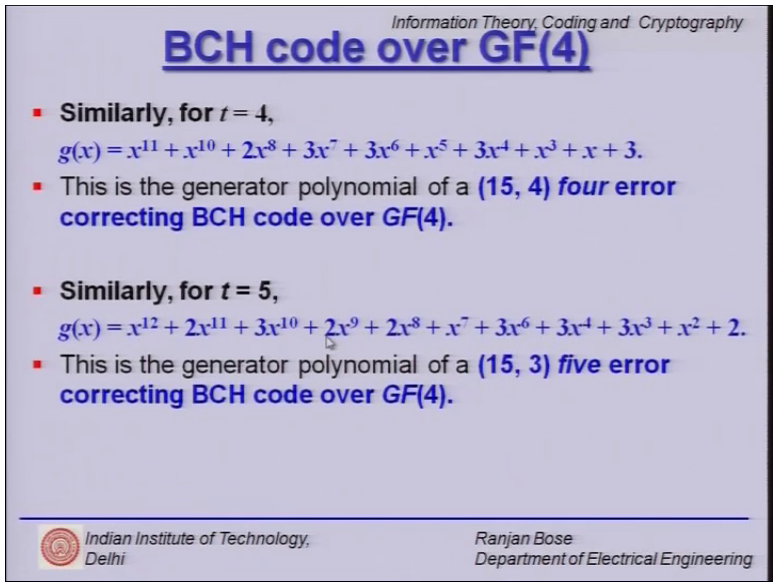
Similarly, for t is equal to 2 we have got the same LCM methodology leading us to g of x which this time has coefficients over GF 4 that is alright because our BCH code is non binary. So this is 15 comma 9 why is this 9 because n minus k is 6 n is 15 so k is 9, so 15 comma 9 double error correcting code, but double error does not mean 2 bits it is 2 symbols and each symbol can be 2 bits.

(Refer Slide Time: 37:08)



And mechanically we continue for t is equal to 3 we have yet another g of x resulting from the product of f 1 x f 2 x up to f 6 x because 2 t, t is equal to 3 so it is f 6 and therefore, we have a triple error correcting BCH code over GF 4.

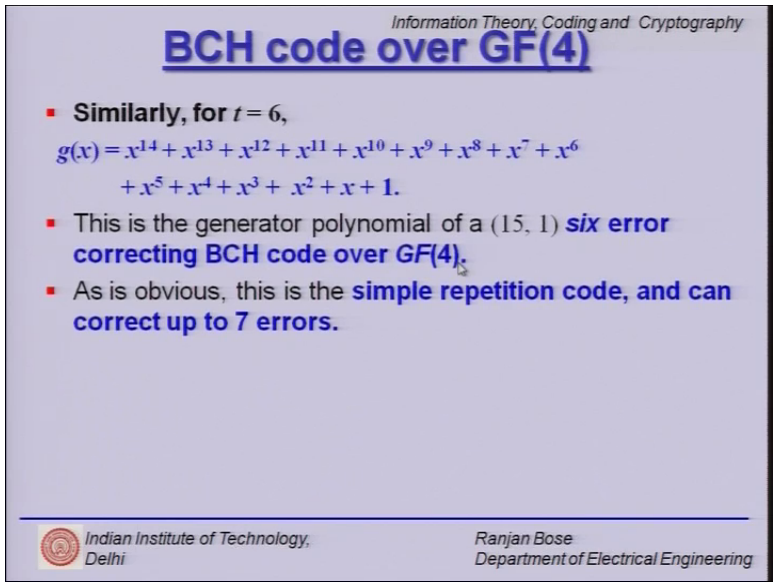
(Refer Slide Time: 37:30)



And we can continue doing this exercise and we get for t is equal to 4 we have this following 4 error correcting BCH code and for 5, we can take the same effort and get 5 error correcting codes.

So, the point that we are making is it is extremely easy to get generator polynomials for BCH code for the designed distances extremely easy.

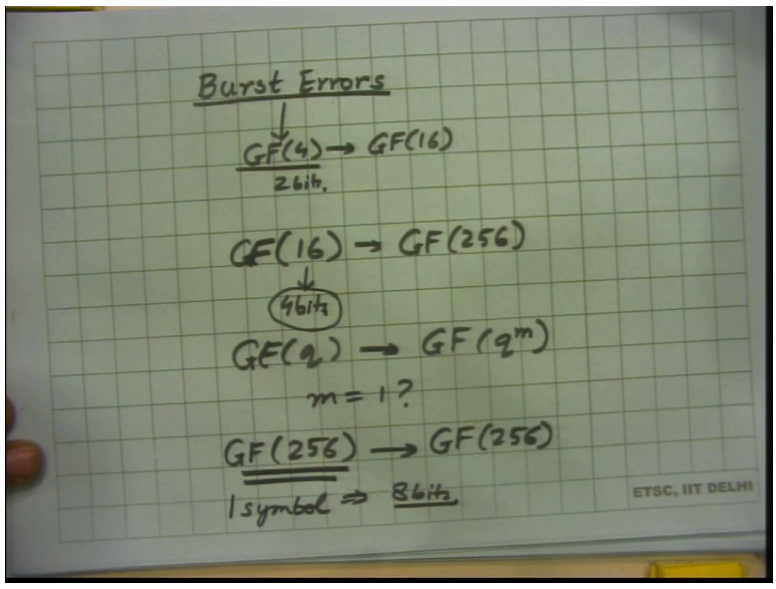
(Refer Slide Time: 38:01)



Student: Sir, can we generalise that for t symbol error correction the BCH code can correct 2 t burst error.

Ok so the question being asked is about the burst error correction no so if we have to say burst error correction we have to interpret it in a different way the burst error correcting comes from it is higher Galva field representation

(Refer Slide Time: 38:25)



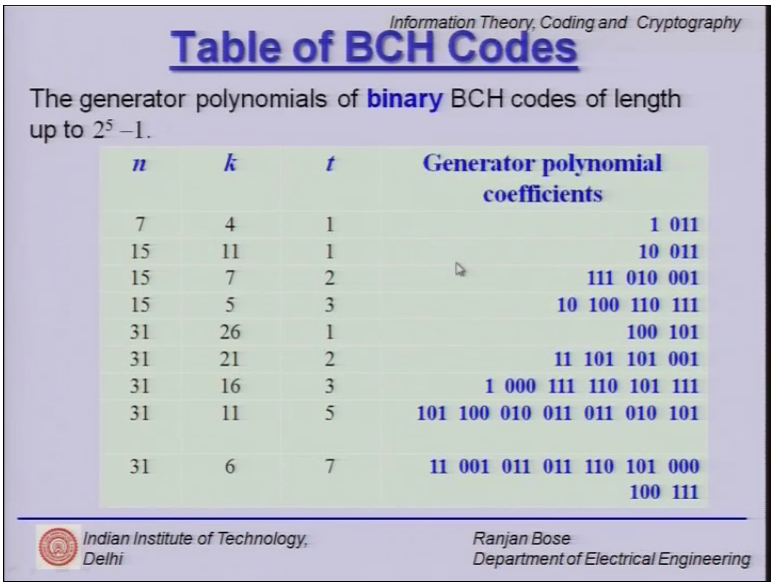
Burst error comes because today we have gone to GF 4 extension field GF 16 this GF 4 is represented using 2 bits therefore, it is a burst of 2, but suppose I had a luxury to go

from GF 16 to GF 256. Here we have 4 bits so process is mechanical, but now I will be able to correct a burst of length 4; so the strength or the burst error correcting capability comes because I am using a larger and larger Galois field to do so fine. So please note that if this is G f q then I am looking at GF q raise power m where m is an integer. If you want to extend this further can I have m equal to 1 and therefore, can we have for example, GF 256 an extension field is GF 256.

In that case here only focusing on burst error correction and this is a special case because in BCH codes we have looked at different values of m we just looked at m is equal to 2 and m is equal to 4, but what if m is equal to 1 GF 2 to GF 2 does not make sense, but GF 256 to GF 256 make sense because again I have the burst error correcting capability here. So even if I have a single error correcting code over GF 256 then the moment I say in can correct 1 symbol single error correcting code right; that means, I can correct 8 bits and I do not care whether all the 8 bits are in error, it is a symbol error. So I will be immediately able to correct a burst of 8 which is significant.

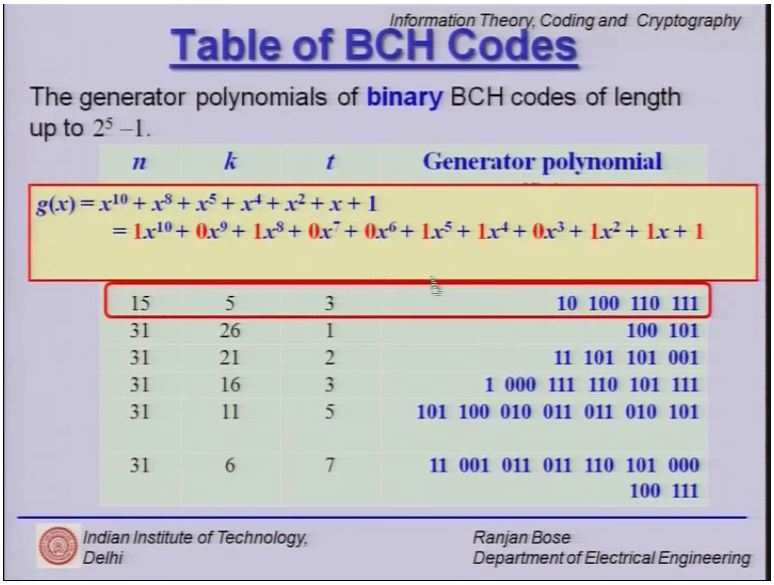
So, the special case when n is equal to 1 falls under the category of Reed-Solomon Codes which we will subsequently study which is the subclass of BCH codes, but we go on and complete this exercise. So we go back to our slides and we have the generator polynomials for t is equal to 4, t is equal to 5 and we can continue for t is equal to 6 wherein we suddenly hit x raise power 14 and if you have x raise power 14, then you know that this is a 15 comma 1 6 error correcting code, but moment it is n minus k is 1 it is nothing, but a repetition code. So; obviously, it can correct up to 7 errors even though you design it so the design distance is more than what the minimum distance you are aim for.

(Refer Slide Time: 41:47)



So, point is the whole process can be mechanized and you can find out for a given n and t k and the coefficients of the generator polynomials so we can always make a table like this. So here are some typical examples given for example, for n is equal to 7 k is equal to 4 you can have a single error correcting code for a 15 these are all binary cases; so this example will looked at 15 11 15 comma 11 symbol error correcting code double, triple.

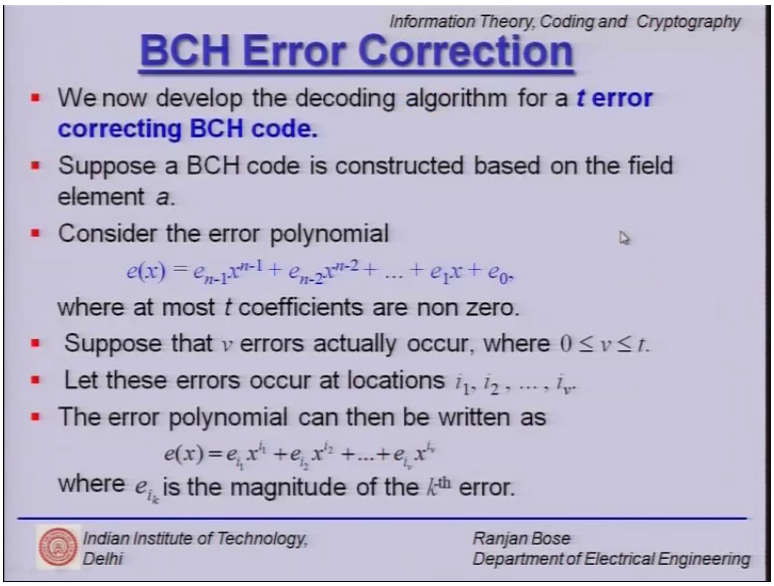
(Refer Slide Time: 42:21)



So, for example, the triple error correcting code how do we interpret this polynomial is that n is equal to 15 we got this g x. Now you write down all the coefficients so some of

them are 0, some of them are 1 and that is exactly what you write so 1 1 1 0 1 1 0 0 1 0 1 so all the coefficients are written up to this. So that is how each one of them can be immediately written in terms of a polynomial, so either I write the polynomials a polynomial coefficients.

(Refer Slide Time: 42:59)

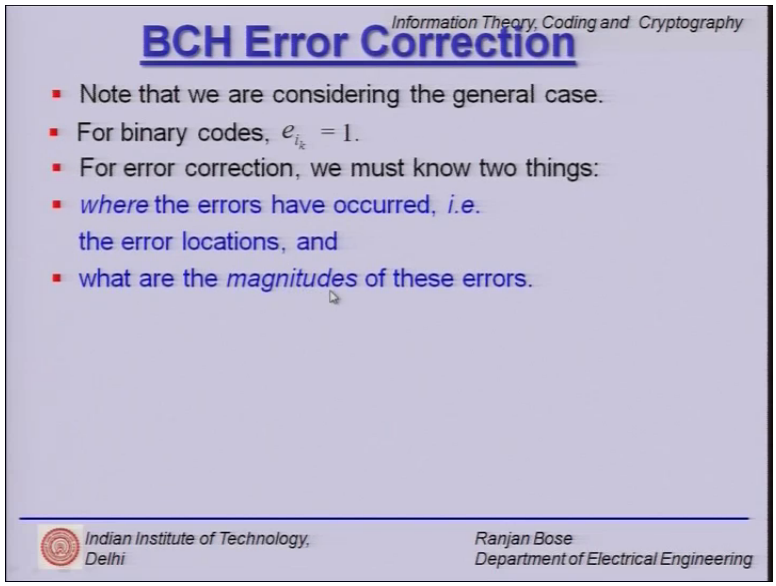


Now, very quickly we will go over the BCH error correction even though it forms a subclass of linear block codes and cyclic codes, we have already looked at the error correction of those let us quickly look over the BCH error correction in a more efficient manner.

So, lot of stuff you have already studied, consider an error polynomial e of x given by the following e n minus 1 x raise power n minus 1 up so and so forth right. So of which suppose new errors actually occur when nu is less than t and t is a number of errors which we wish to correct. So what are we considering suppose you have designed a triple error correcting code well it should be able to correct also two errors so one error so less than fewer than t error should also be corrected, up to t we should be able to correct.

Now, either the errors are happening and where they are happening is also important. If we know what is the error and what is error location then we are home free, so we would introduce this notion of the location and the magnitude of errors; so the error polynomial it is written as follows.

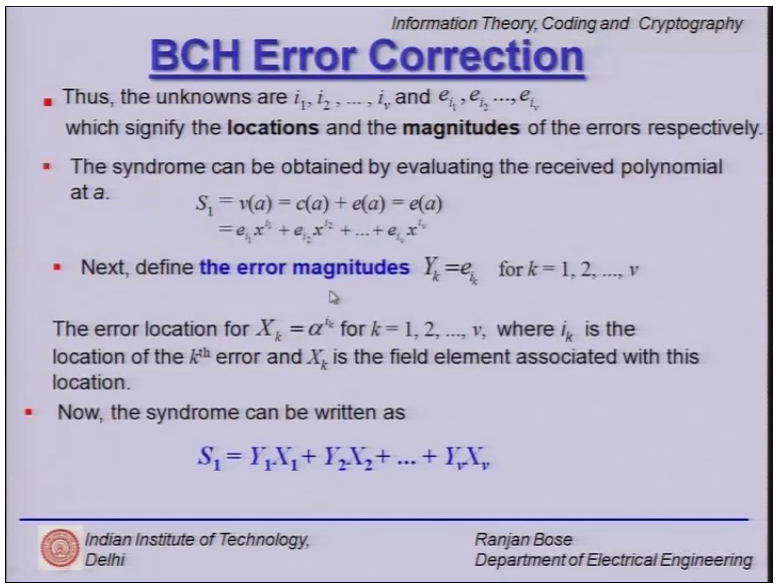
(Refer Slide Time: 44:22)



So, binary codes either e i k is 1 or 0, but in general it can be any value depending upon which Galois field we are working.

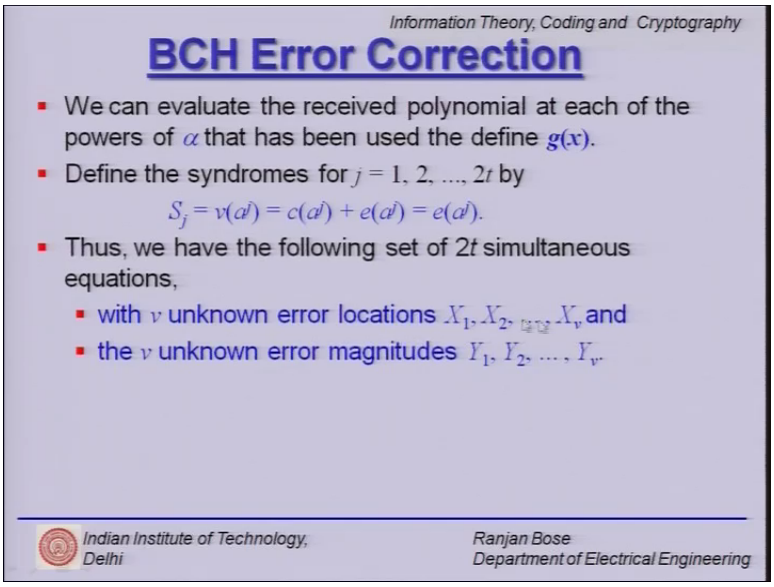
So, let us start with the basics we need to know for error corrections where the errors have occurred that is the location and what is the magnitude. For binary case it does not matter because it is either a 1 or 0 so if you found it to be 0 make it 1 so flip, but if it is done by any GF 4 whether if it is a 1 and whether it should become a 0 2 or 3 you have to figure out the magnitude of this error also.

(Refer Slide Time: 45:07)



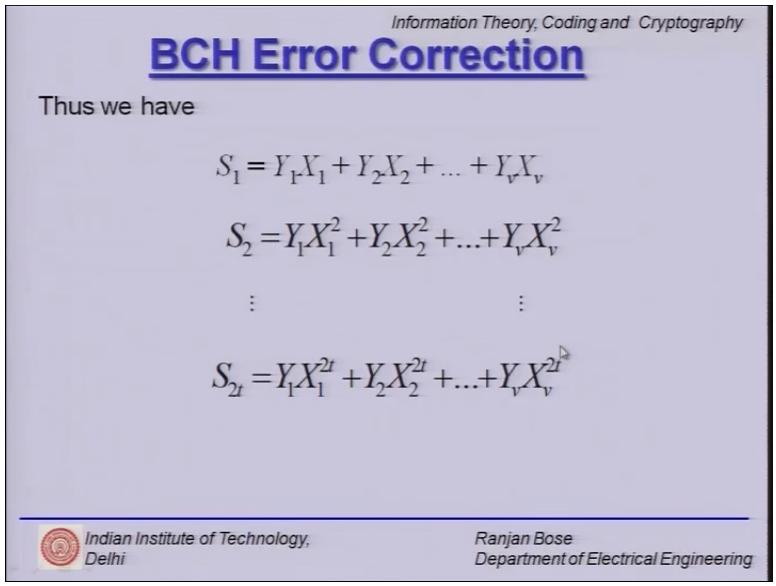
So, what do we do, we look at the syndrome the syndrome can be obtained by evaluating the received polynomial at alpha we did this theory in the last class and that is why this 2 t term came, while the g x is LCM of f 1 x plus into f 2 x into f 3 x up to f 2 t x. So the same philosophy we write the syndrome a polynomial as follows and we then define the error magnitudes by y k and the error locations by x k. So the syndrome can be written in the following way in terms of the error magnitudes and error locations.

(Refer Slide Time: 45:50)



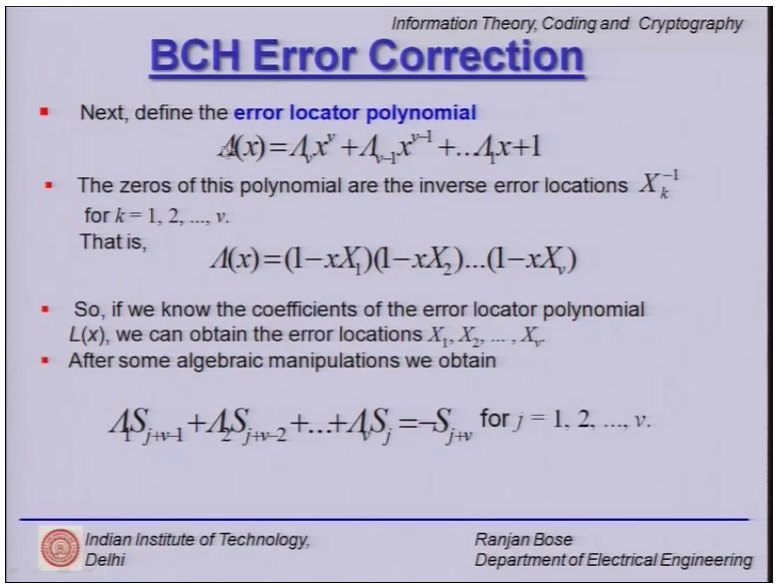
Now, we have 2 t syndromes so we have got S j equal to this evaluation at j is equal to 1 2 up to 2 t. And now we have got this 2 t sets of simultaneous equations and we know that nu less than equal to t errors have happened. So we have nu unknown error locations X 1 X 2 up to X nu and nu unknown error magnitudes Y 1 Y 2 to y nu and our aim is to solve this X and Y.

(Refer Slide Time: 46:32)



Thus we have a system of equations like this and now some of the interesting algebraic manipulation that juggleries can be used to solve them efficiently.

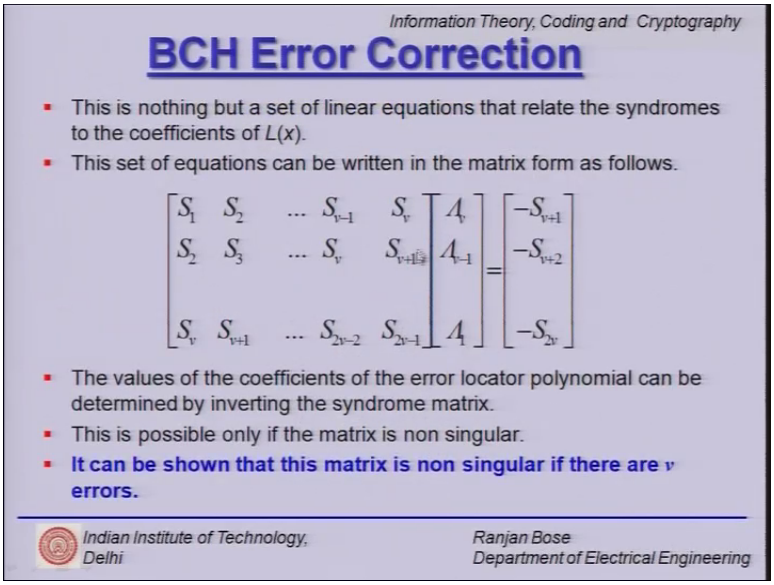
(Refer Slide Time: 46:45)



So, we define an error locator polynomial as follows the 0's of this polynomials are the inverse error locations, this is how you define an error locator polynomial ok. So if you see this lambda x is nothing, but 1 minus x capital X 1 the locations, and then we know the coefficients of the error locator polynomials can be obtained using the locations X 1,

X 2 and X 3. So you do some manipulation in terms of the syndrome polynomial and you can write it as follows.

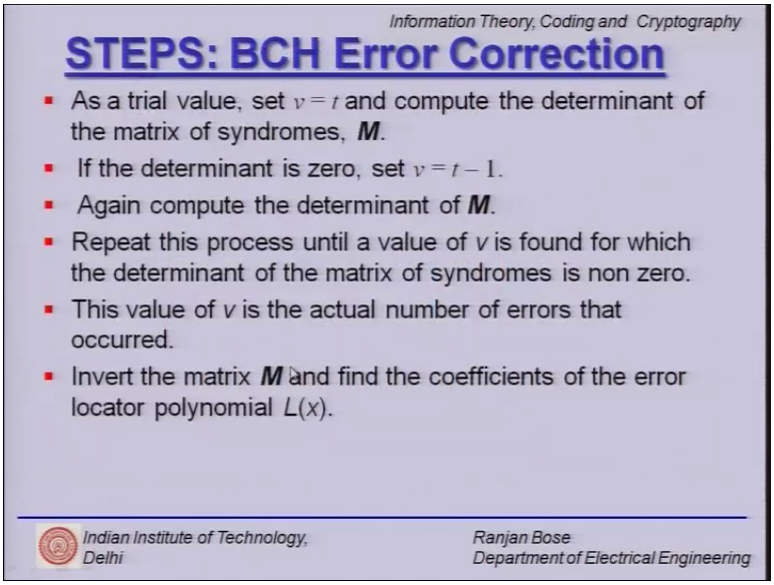
(Refer Slide Time: 47:22)



Finally it can be represented in the matrix form in terms of the syndromes the and the error locator polynomial coefficients.

So, it can be shown that this matrix is non singular if there are nu errors. So we should be able to solve this to correct for nu errors, but we do not know because we have designed it to correct t errors and if less than t errors occurred what will happen.

(Refer Slide Time: 47:53)

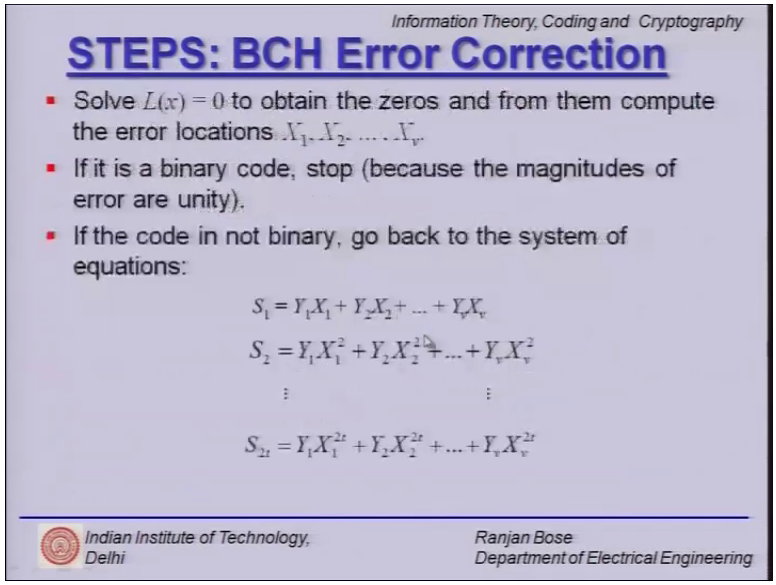


So, these are the steps that we do, we always start with the worst case suppose the maximum number of errors occur for which it was desired, so set nu is equal to t and compute the determinant of the matrix of syndromes M.

Now, if nu is equal to t and t errors have occurred then it determinant will be nonzero; if it is 0 it means that fewer than the number of errors for which you have designed have occurred. Suppose you have designed for 3 error correction and only 2 errors have occurred then the determinant will be 0, then of course you lower it to t minus 1 and so and so forth and till the determinant is nonzero, that will tell you how many errors have happened. Now you have to find out where the errors have happened and what is the magnitude of the error.

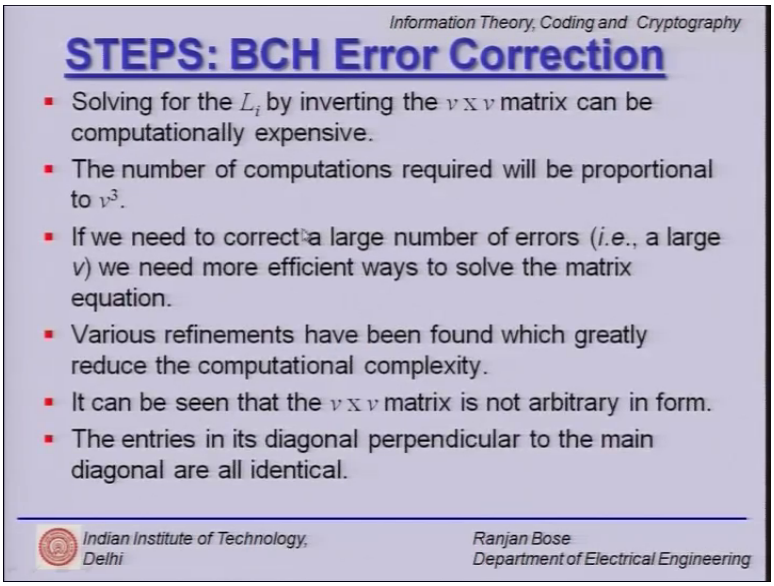
So, now it is nothing, but solving this matrix as in terms of the set of equations in terms of the matrix.

(Refer Slide Time: 48:50)



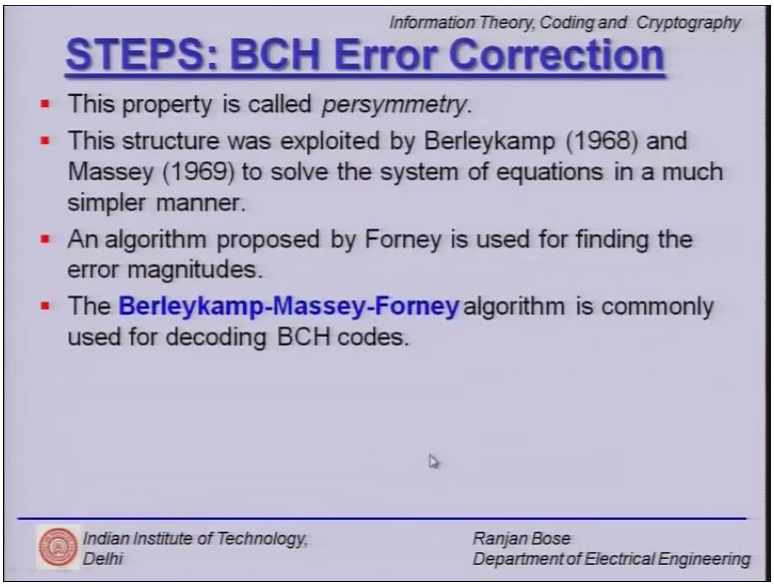
So, we solve the location polynomial and if it is binary code, once you have found the locations of the error you stop because it is either 1 or a 0, if it is non binary we go back to the system of equations and find out the Y 1, Y 2 up to Y nu which are the magnitudes.

(Refer Slide Time: 49:12)



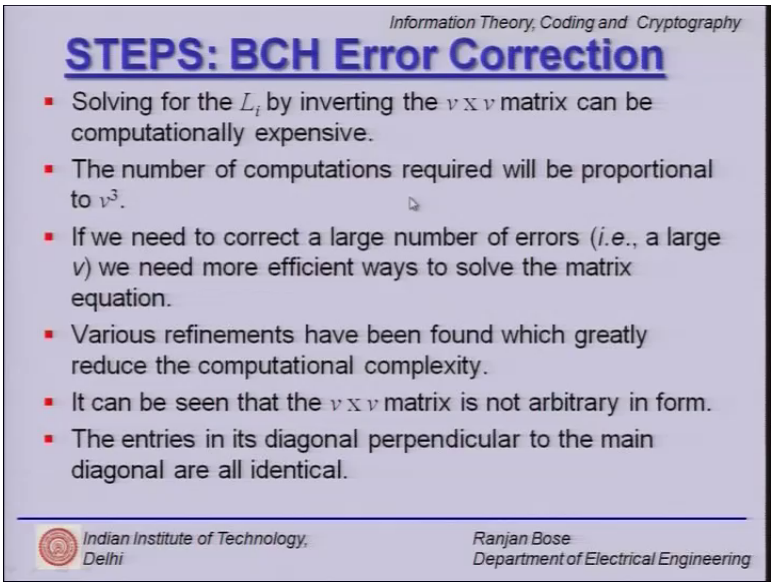
Thus we have been able to efficiently decode BCH codes using those nu cross nu matrix.

(Refer Slide Time: 49:24)



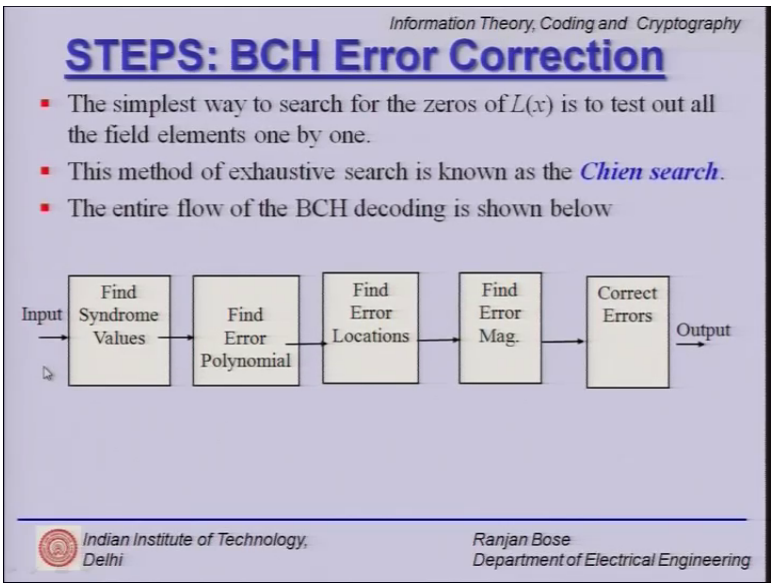
So, the people have done a lot of work on this and you know the entries in the diagonal perpendicular to the main diagonal are all identical.

(Refer Slide Time: 49:30)



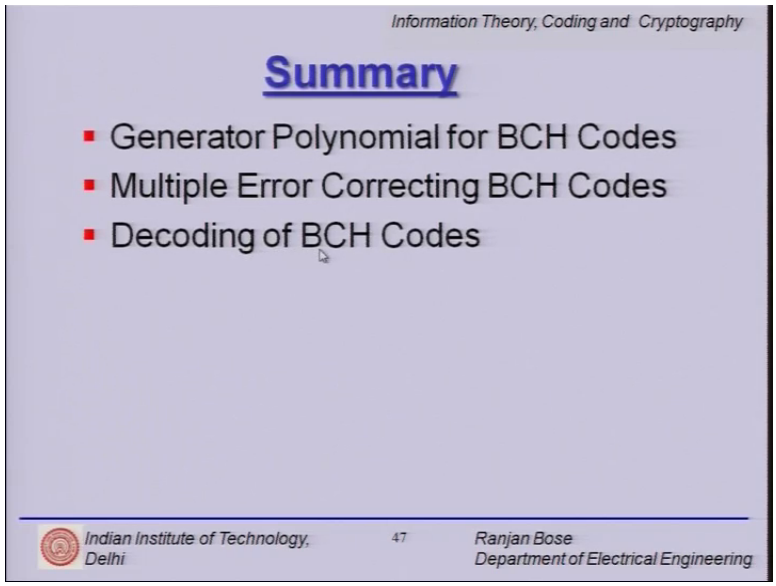
If you look at this thing you see s 2 s 2 then s 3 s 3 s 3 and so and so forth and diagonal s nu s nu s. So this is a very peculiar structure and this is called the persymmetry and this structure was exploited by Berleykamp-Massey to solve the system of equations in a much simpler manner because of the symmetry. And another algorithm proposed by a Forney is used to find out the magnitude of the errors. Consequently this efficient technique is called the Berleykamp-Massey-Forney algorithm and it is commonly used in the industry for solving the error locations and magnitude of BCH codes.

(Refer Slide Time: 50:19)



So, finally, if you have to summarise this is the flowchart; you have the inputs, you find out the syndrome values, find the error polynomials, then find the error locations if it is binary stop if it is non binary find the error magnitudes, then correct the errors and you get the output. So this is the flowchart of the BCH error correction capability.

(Refer Slide Time: 50:45)



So, now we come to the end of today's talk let us summarise what we have done. We have looked at the generator polynomial for BCH codes and then we look at how BCH codes can be used to correct multiple number of errors and finally, we looked at some examples and figured out the Berleykamp-Massey-Forney algorithm for decoding a BCH codes efficiently, with that we come to the end of this module.