Digital Switching

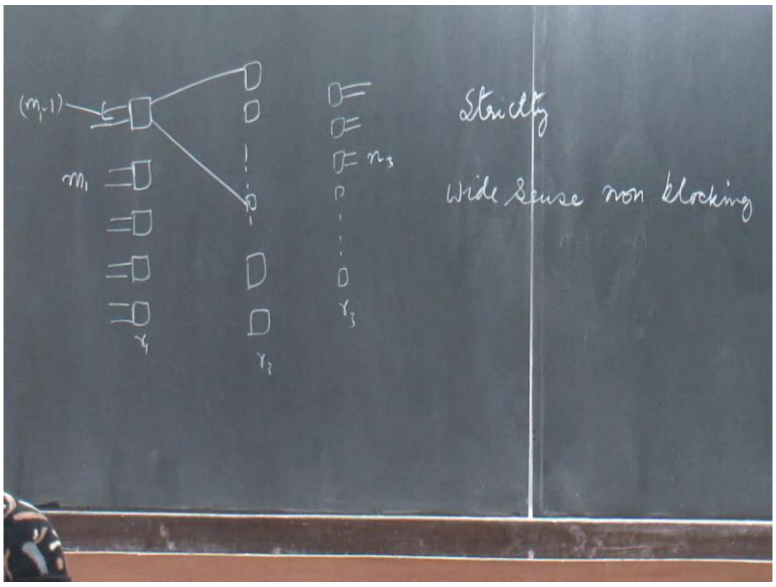
Prof. Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Lecture – 9

So, last lecture I think what we were doing was a clos network, and that particular condition was for unique cast condition, okay, not for multicasting see. So, in fact, after that I went back and then searched into the literature. I figured out it is still an unresolved issue unresolved problem pending, and still people have been investigating these things how to figure out the bounds. But I could actually sit down starting write and could actually figure out certain insights, and from there, I can actually build up a bound, but I think this can be further improved a lot. And there is also classification which also has to further come in.

So, my thing multicasting in multistage interconnection switching networks is still an open area open field which needs lot of investigation. And I think the one of the best reference will be that one of the gentlemen whose paper actually has been already uploaded from Taiwan who actually has written a book only on multicast multistage interconnection networks. So, whatever I actually have done yesterday night, so this is something new which is not there in my notes. So, this is a new addition, and for last eight nine years, I was actually teaching incorrectly this particular portion. And no students have actually ever told me that I am teaching it incorrect; they should have figured it out that it is incorrect. So, what they just struck to me while teaching the previous class; this happens once in a while, and I always get excited when this happens.

(Refer Slide Time: 01:59)



So, coming back to the same clos configuration and remember it is all argumentative; I am not doing any mathematics but can be done through Paull’s matrix, but Paull’s matrix is not a good approach, which I also figured out while reading the paper by John Turner, okay. There is also one paper which is also loaded in site from him, okay. There is an alternative approach for doing this. I will just give an idea and leave it; I think you people have to do your own investigations, because it is an open intend research now, okay. So, this is a situation and so far, what we know is you have r 3; you have r 1. I am just keeping the same notations which I have been using. I am not using the notations which are used in the paper; they use a different notation, okay.

So, you have to be careful while you are reading the paper, because I also got confused; chances are that you will also get confused, but I think if some practice you will be able to appreciate that. So, m 3 outputs, m 1 inputs, r 3 switches. So, clos theorem actually simply says that for a unique cast connection because if you can make only one connection. So, m minus m 1 minus 1; in worst case, these many are occupied. One is left out which has to be connected; this should have occupied all these lines. So, in fact, we called this thing as a unique cast is a special case of multicast where fan out is 1.

Now what is the fan out f factor? This I have not introduced earlier. So, fan out actually says that what will be the total maximum size which is permitted for a multicast group the output ports. So, because I am only permitting only one output port to be connected to input port, f is equal to 1. I think k I can take a case of f is equal to 2 also where only two output connections are permitted. Now this f factor will actually play a role and figuring out what will be hardware requirement. When f is equal to see all possible nodes can be connected.

Now you have to look at there is going to be somewhere an optimization. If I have to connect all the output ports to one port, any switch will work, okay; any switch will work. So, far this r 3 will become equal to m 1; that is good enough. So, you have reduced this number of elements, but if you start making it one, the extreme case on the other side is going to be m 1 plus n 3 minus 1; that is the one case when f is equal to 1. And if I take a fan out case of when f is equal to r 3 into n 3, then you require only r 2. Here r 2 has to be greater than or equal to this is here r 2 will be greater than or equal to how much? Any node you should be able to connect; r 2 should be equal to m 1 n 3, maximum of this under this condition, which is obvious.

I think intuitively you can figure it out; the problem happens in between. And I also do not know I am not going to do that. I am just going to give my estimate. At least if you have whatever I am saying equal to or higher than that, it will be strictly non-blocking, okay. Anything lower that I cannot say there is a possibility; you can actually have strictly non-blocking, but I do not know the proof, but that again it has to be worked out. Now I am also now going to add one more definition; earlier I have only told you strictly non-blocking. So, there is something called because this also term has been very extensively used n r. I also figure out that even from clos network, algorithms are available which will make it wide sense non-blocking, and it requires less hardware, okay.

There has been again lot of proofs on that. So, I have defined strictly non-blocking earlier, and I think it is a good idea that if we separate out strictly non-blocking for unique cast and strictly non-blocking for multicast; two definitions should be separate always, since two for wide sense non-blocking. So, wide sense non-blocking, there will be certain connections which have to be done between input and output, okay. If already some connections have been done and you want to do a new one, okay, you may not be able to do the connection. It is a blocking in that sense, but if it is a strict non-blocking, you can always make the connections without bothering about the earlier connections.

But take a case when it is possible to know all I o maps and circuit is you know all connections. So, input I 1 to some output something. So, I to o map is known to you already, and all connections are torn apart; none of them are connected. And you are able to invent an algorithm by which you can pick up connections in certain order, and they are set up as per certain pattern. If this algorithm you can invent and if that algorithm is used, you can always set up all possible I o map combinations; if that is possible, that technically means you are avoiding certain switch states, then switch will be always non-blocking.

You can set up all possible permutation combinations of input to output only if you are following certain algorithm. For a strictly non-blocking, you do not need any algorithm; any kind of map which is there I o is free, you can always set up the connection. But here you require an algorithm that is the only condition and rearrangably non-blocking even if you cannot build up an algorithm of this kind; you might have to now decide on rearrangements.

Student: Sir, random routing is used?

Professor: That that that will not be used; that will not be used.

Student: That will not be used?

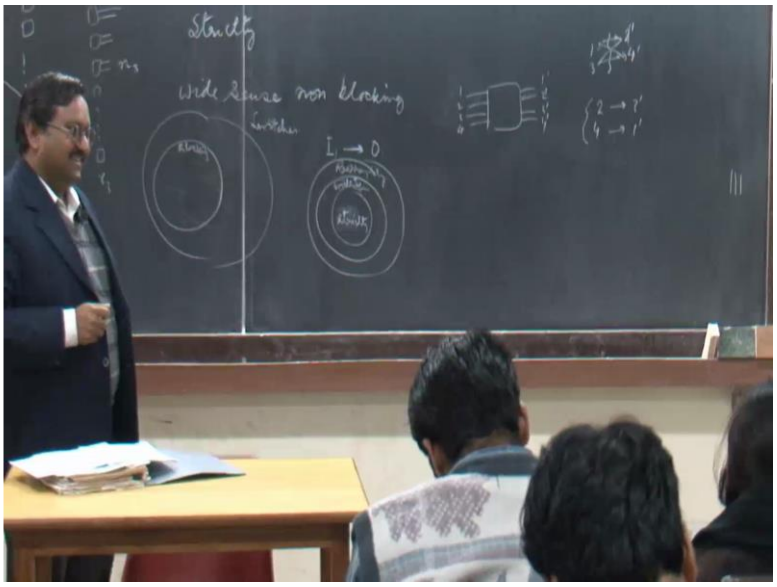
Professor: Random routing is for that; that was only for finding out estimating the blocking probability.

Here I am not. I am trying to find out the characteristic of an interconnection network, okay. So, that is wide sense non-blocking; this I think is the good way of defining if you have an algorithm. So, I do one-by-one every connection, and based on the connection, I will choose an algorithm to set up. And this algorithm will ensure it will be always non-blocking. You cannot just simply pick up anything which is free and set up a connection.

Student: Sir, what is the difference between the wide sense non-blocking and rearrangeably non-blocking?

Professor: Rearrangeably non-blocking is okay; I think let me elaborate on this.

(Refer Slide Time: 09:32)



You have this some switch I am giving an example; they are only four by four. Okay, how it is done inside; do not worry about it. So, there is one, two, three and four; I am giving you four scenarios. So, I have created some map, say, let me put primes two to three prime, four to one prime, okay; these are the two connections which are already setup. Now without bothering about without disturbing these if I can set up say any combination. So, now what is left is one and three and what are left on this side are two prime and four prime. Any possible combination I can take; they are two into two total four combinations which are possible, okay.

So, it can be one four if it is a unique casting; I can always set up these things without disturbing these that is the strictly non-blocking, okay. Whatever way this connection has been setup, this connection also can be setup in this switch in multiple different ways with the multiple possibilities; multiple switches threats are there. But I have built up an algorithm when I am setting up these paths; I follow that algorithm to avoid certain states of the switch. And now when this connection request comes in when I set up, because I am avoiding certain switch states; I will have to set up these that is a wide sense non-blocking; I have t follow an algorithm.

Student: That can be rearranged?

No, rearranging it; I am not doing any rearrangement. When they were set up, at that time I followed an algorithm which is permitting me to maintain this as strictly non-blocking.

Now if I dismantle all these connections and then I can always find out a combination that all maps can be done that is a rearrangeably non-blocking. You can follow in less hardware; wide sense non-blocking requires slightly more always. Wide sense non-blocking requires more amount of hardware than rearrangeably non-blocking, but whereas as we are setting up connections, you are following a certain algorithm that what we call a switch engineers thumb rule, rule of thumb base that try to push the connection towards the heavily loaded side as far as possible, okay. If it is not possible, then go to the least loaded site in the network, and you will minimize on the blocking probability.

Now that is a rule which you are following; if you do not follow the rule, you might end up in blocking. [FL] This actually wide sense non-blocking I will give an example, then you will understand later on. Okay, what you are saying he is saying if there is a blocking network; if you look in terms of switches, all switches by default set of all switches and if you look in terms, there will be a blocking switches. Within blocking, there will be this is what we call blocking in the sense; you cannot set up without disturbing or with disturbing, it all depends on that.

Now making every time, [FL] whether all blocking will get encompassed by this no; blocking is a separate set altogether.

Student: See blocking may be separate set, sir; this will be diffuses.

Professor: If there is a strictly non-blocking thing with wide sense will be a small part of it.

Student: Sir, smallest will be it

Professor: Okay.

Student: When there will be wide sense and then maximum will be rearrangeably.

Professor: Right.

No no not minimum; I am talking about all these thing. You need not do any rearrangement; you do not do any rearrangement or do a rearrangement is actually is a rearrangement; rearrangement of zero entities. So, this is also a rearrangeably non-blocking in that sense you do not avoid certain states that as good as avoiding certain states. So, this also becomes subset of wide sense. Only thing addition thing which is coming here is avoiding certain states; here you are not avoiding, and here you further now you are able to do the rearrangements.

Here you are not doing rearrangements, not doing rearrangements plus doing rearrangements is equal to doing rearrangements; it is like this. So, in that sense, I think this should be okay.

Student: Sir, which is followed when there is certain connections are already present. Is that they just depend upon what kind of connections already present?

Right, it has to figure out what is the current state and what new state has to be set up. And based on that, usually the new connection when you want to set up can lead to more than one few many states. So, you have to choose one of those states; which one you have to choose will be given by the algorithm. So, you have to avoid certain states so that you should not end up in blocking states. Now that is basically is the idea.

Student: Algorithm is similar?

No; it all depends on the switch design. So, once we will do it at some points for five stages, then you will appreciate that, okay. So that I will do later on once I finish up with the cantor network after that.

Student: Sir, what is the internal problem with output state problem.

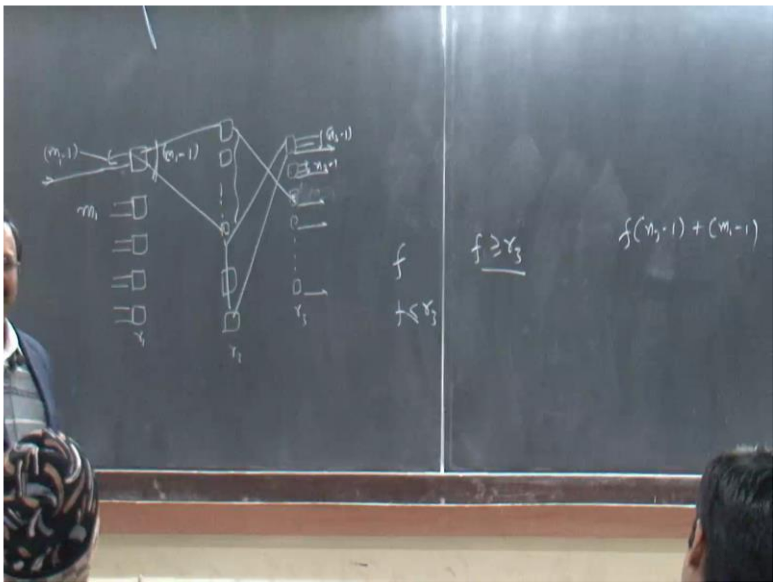
Professor: Hmm.

Student: Internal problem.

There is no blocking in any switch here, because these ports are more than this. Only blocking is because of interconnection; not switches are only strictly non-blocking, they are crossbars or if you want to implement them, we are assuming that you are recursively making them also by three stage network which is strictly non-blocking if you want to maintain them strictly non-blocking switch, okay. And if it is a rearrangeably non-blocking, each block should be further can be implemented by three stage rearrangeably non-blocking switch; you have to ensure that.

So, you can keep on doing it till you come to the fundamental element. So, it will not be three stages, but technically, it will be more number large number of stages, but it is a cascade recursive construction, okay. We will come to that later on, so that we can estimate the cross point complexity. So, this one is pretty simple what I did.

(Refer Slide Time: 16:45)

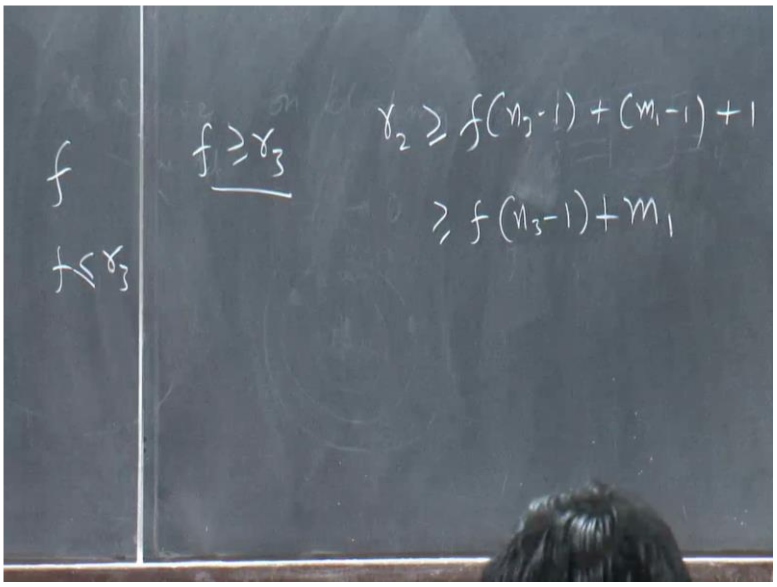


I am doing now for f; fan out f is general, switch is this. So, I will assume this is the one input which has to go to f outputs. If all f outputs are one single switch, life is very simple; I have to just simply route only one and then this will do the fan out. Fan out happens in third stage, okay. Fan out also can happen in first stage, because I can do a fan out here and then this can do the fan out; that is also possible. I am not taking that case as of now; I am looking at worst case scenario. So, there m minus 1 m 1 minus 1, these are occupied; only one is free which has to be routed to somewhere.

Now these are occupied switches. I require some free switch to connect to the f possibilities, and f fan outs are going to happen at this point, at this point, at this point. So, I am assuming that f is less than r 3; f is going to be less than r 3 less than or equal to r 3. The moment f is greater than or equal to r 3, I will simply take r 3 as the fan out; that is very important. So, this comes condition between minimum of the two kind of thing will come. Now this particular switch I can assume that n 3 minus 1 are occupied and they are connecting to the middle stage switches and not overlapping with these.

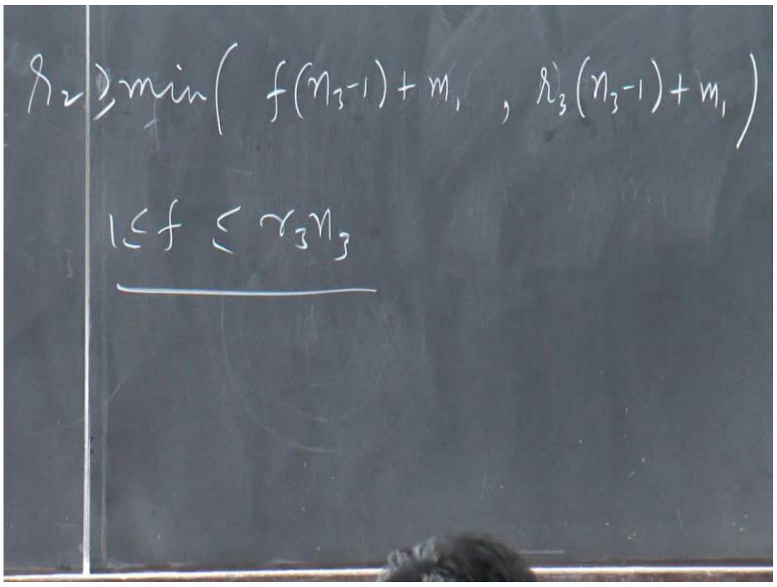
There is only one guy whom I have to do the fan out worst case scenario, okay. I can keep on doing this n 3 minus 1 are occupied n 3 minus 1. So, I take only f inputs only those switches where I need do the f fan outs f switches. So, how many these middle switches which will be occupied? F into n 3 minus 1 plus m 1 minus 1; I need one extra to whom I can just put this free connection. And since, only n 3 minus 1 are occupied, each one of them can be connected to this free one; these are all independent remember. I am taking this one; this one is free not occupied anyone of them. So, all the links from all of this fan out output switches are free to that. So, I can just route one here this can do the fan out to all f's. So, only one extra will be required.

(Refer Slide Time: 19:41)



So, you require plus one. So, r 2 has to be greater than or equal to this for f multicast, okay. So, the condition will turn out to be this; that is what I am assuming in worst case scenario. Now you can actually look into the Paull's matrix; it will become far simpler, okay. This is also I think known as this is actually theorem also comes from the graph theory from where this can be easily derived; this is fine. So, next is possibility is that my fan outs are more than r 3 worst case scenario.

(Refer Slide Time: 20:36)

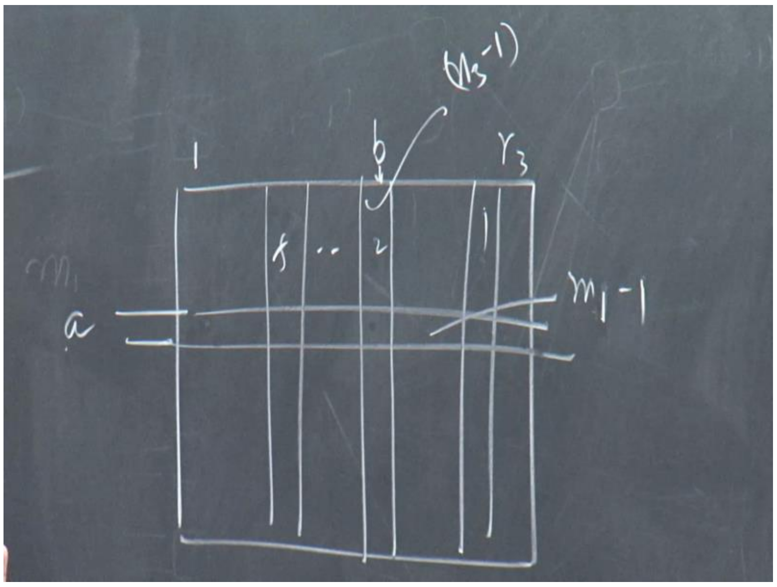

$$r_2 \geq \min\left(f(n_3-1)+m_1 \;,\; r_3(n_3-1)+m_1 \right)$$

$$1 \leq f \leq r_3 n_3$$

So, then in that case, I have to actually have min of f n 3 minus 1 plus m 1 r 3 n 3 minus 1 plus m 1. R 2 has to be greater than this much minimum of these two, which is obvious because when my fan out becomes more than r 3, okay; rest everything is same. So, this will be only smaller than this when f is larger than r 3. So, you do not require more than this that is upper bound I get for multicast. I think they are the better bounds which are possible. As I told you when fan out f is equal to r 3 into n 3, you require actually r 2 has to be very small. R 2 has to be equal to maximum of m 1 and n 3. When f is equal to 1, it is m 1 plus n 3 minus 1.

And when I am looking at some other f's, these are nothing but in these two ranges; f is equal to one and f is equal to the maximum value of r 3 n 3. You are in this two range; your f cannot be anywhere else f has minimum value one n maximum value this. So, all intermediate value will require larger hardware which is obvious and somewhere the optimum happens. This optimum will never be greater than this which is obvious; this optimum will never be greater than this. So, I have given you a bound on that. So, coming to Paull's matrix to actually derive the same thing; so, I can forget this and do the Paull's matrix. I think for some of you now by this time, you can figure out what is Paull's matrix. So, once you know the middle stage crossover stuff.

(Refer Slide Time: 22:47)



So, I m doing exactly same thing in Poisson matrix; what it means? You have a row; you have a column. You know how many columns are there r 3 columns, 1 to r 3, okay. You want to set up a connection. So, you have only n 3 minus 1 symbol. This guy is going to have one output free; remaining is all occupied. So, it has n 3 minus 1 symbol. So, worst case scenario and I want to do f fan out. So, there are f columns only to whom I can connect at worst case not more than that. We will of one two three and some f; in worst case scenario, each one of them will have n 3 minus 1 symbols.

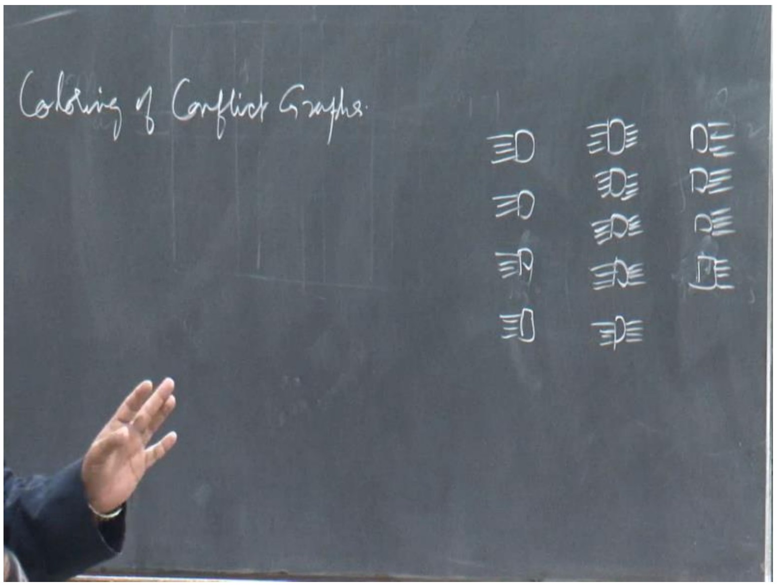
So, n 3 of minus 1 symbols here, n 3 minus 1 here, n 3 minus 1 here, n 3 minus 1 here; all symbols put together, they are all distinct which is a worst case scenario. You make some total of them and if your r 2 is more than that at least one higher than that, you can find out a free symbol which can be placed everywhere, because that symbol also has not been used here has not been used anywhere, and a symbol can be repeat in multicast scenario. Only one extra will be required, and that is how you get f into n 3 minus 1 plus m 1. Now there is an alternative approach which is known as coloring of conflict graphs.

Student: Row maximum m1 minus 1.

Professor: Row maximum, oh, m 1 right right; it is m 1, you are right, it is m 1.

Now there is an alternative approach I will just give the idea of the approach I think and I leave it to you to do the further investigation, okay. And maybe sometime later I will try to explain this stuff. So, again I will be doing it for first time. So, I will kick out something from the syllabus to do this.

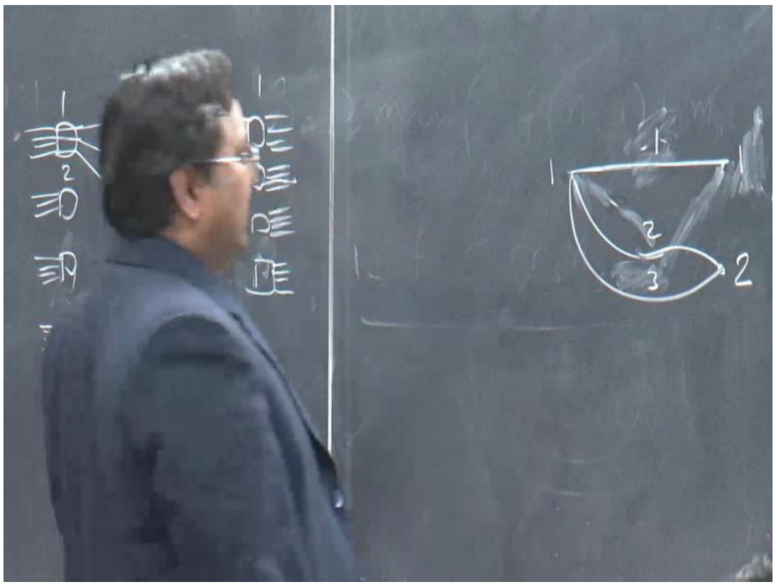
(Refer Slide Time: 25:03)



This is known as coloring of conflict graphs. So, idea is very simple. I am just giving the idea here and then we will move out to Slepian-Duguid theorem which is for rearrangeably non-blocking, okay. Now Slepian-Duguid theorem again is only for unique cast connections not for multicast, okay. So, similar I think treatment is required for multicast scenario even in that case. So, in this case, the way it is done I am just going to take a very simple example. So, these are two, okay, keep it three. So, there are four outputs in this case, and I think let me keep it four, because all switches then these will be four by four switches, okay.

So, what will be the minimum number of required switches in the middle stage for strictly non-blocking; I am not bothered about it as of now. I am looking at the conflict graphs, okay. I am looking at giving you the philosophy how it is going to work. So, Paull’s matrix also is nothing but a graph theoretic approach, okay, because you can see in Paull’s matrix, there are rows and columns and I am putting something. So, the middle stage switch is nothing but a link; that is what I am doing.

(Refer Slide Time: 27:05)

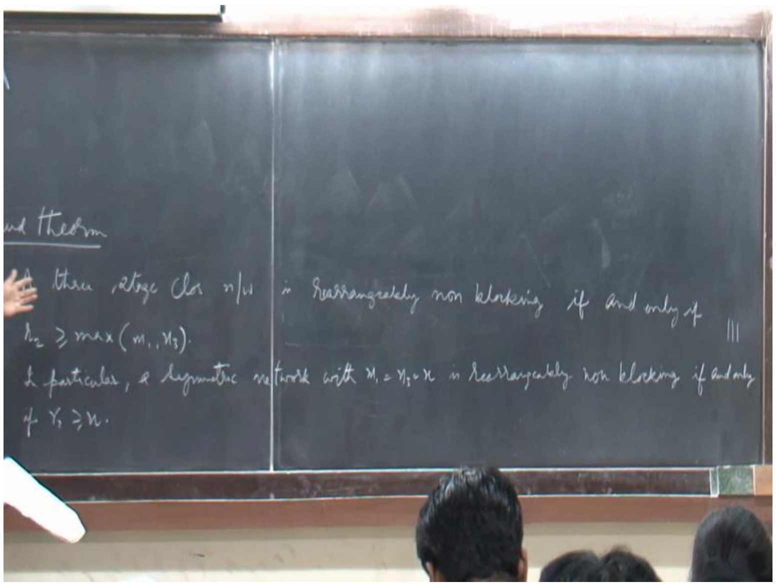


So, when I am strictly saying that one is connected to two, I am now looking at an edge; this edge which is defined is the middle stage switch. So, I can connect; I can actually say this edge as a there is a vertical graph inversion which actually can be done. So, I can see this edge as a node and this as the switches not basically the input ports. These are the switch numbers in the input stage, okay. So, for this case for example, if I set up the connection this way, only this much connection is there how to set up this graph. So, in this case, these are the edges which are there.

So, one-to-one, this particular switch I am passing a route through one. Similarly, when I am passing a route through two, there will be another edge actually here. Now since the nodes are common, I cannot use same color here; I cannot use same color. The number of edges which can emanate is actually equal to the number of input ports which are there and number of output ports which are there; that are the degree of the nodes. You have to ensure when two nodes are sharing something, they have to have. No, one to one there is nothing; yeah, right right. It has to be one to two, and this will be two, and this will be three.

No, there is I think is a problem here. I am making some mistakes somewhere in this case; okay, I think lets because multicast we will come back again anyway. So, that time I will take care of this; we can skip in this particular part. So, when doing multicast, I will come back to this. So, let us move to our regular stuff the Slepian-Duguid theorem the next one.

(Refer Slide Time: 30:09)



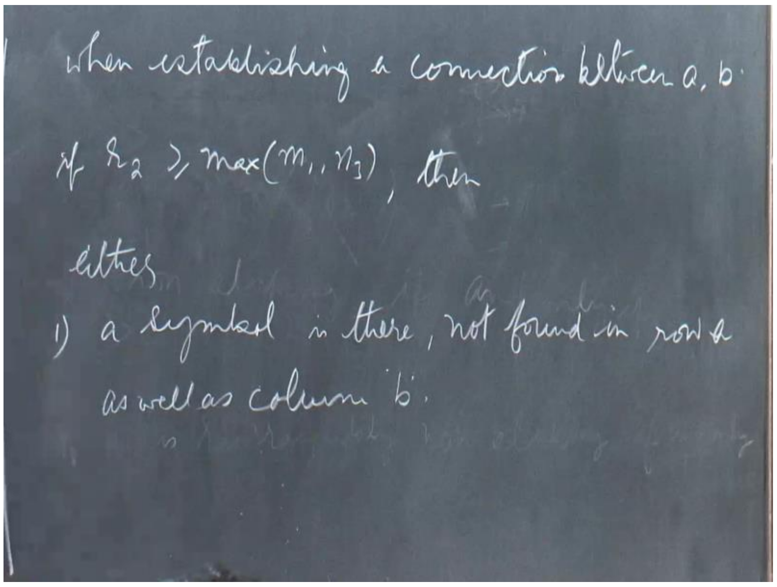
ud theorm

A three stage Clos n/w is rearrangeably non blocking if and only if $r_2 \geq \max(m_1, n_3)$.

In particular, a symmetric network with $m_1 = n_3 = n$ is rearrangeably non blocking if and only if $r_2 \geq n$.

So, I think because of the same conditions, it is only true for unique cast connections. So, in this case, the statement is the three stages clos network is a rearrangeably non-blocking if and only if r 2 is greater than maximum of m 1 and n 3; that is the condition. And in particular, a symmetric network with m 1 is equal to n 3 is equal to n is rearrangeably non-blocking if and only if r 2 is greater than or equal to n, which actually is nothing but from here we just put m 1 is equal to n 3 is equal to n, and this is what will be true written in different form; that is the only thing. Now we have to prove it actually; this will also tell us how you do the rearrangements. So, let us start with the proof.

(Refer Slide Time: 32:33)



when establishing a connection between a, b:

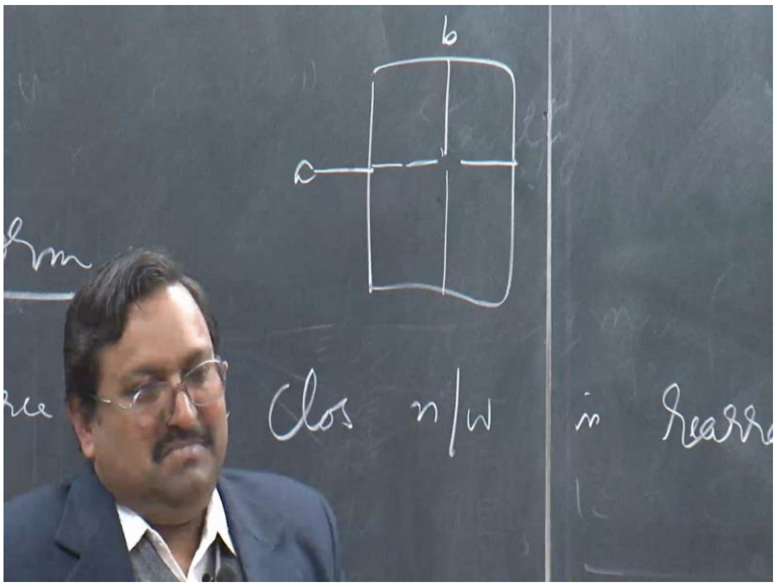
if $r_2 \geq \max(m_1, n_3)$, then

either

1) a symbol is there, not found in row a as well as column b.

So, first condition we have to prove is when establishing a connection between switch a input stage switch a and output switch b, okay; if r 2 is greater than or equal to maximum of m 1 n 3 either, then there are two conditions which are possible. The first one is a symbol is there not found in row a as well as column b.

(Refer Slide Time: 33:47)

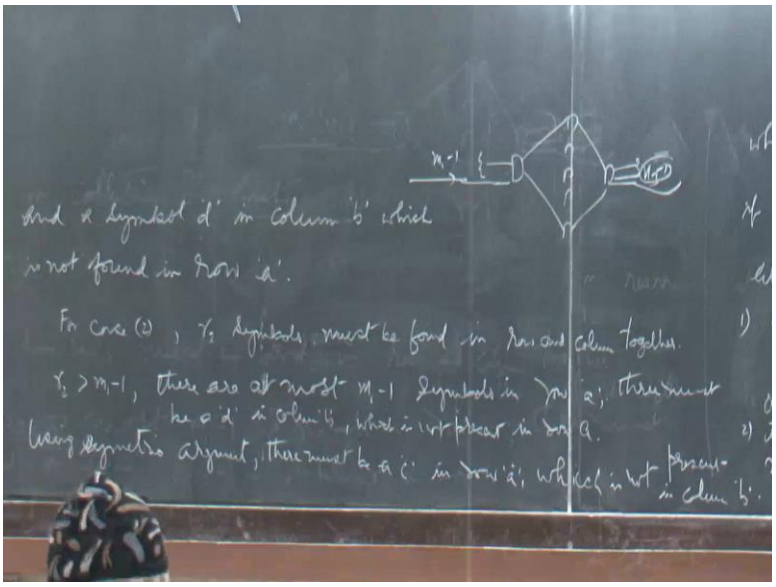


What I actually imply is these are rows and columns in the Paull’s matrix. I want to set up a connection between then, and I have to prove if this condition is satisfied, either I will get when I look into this, look at all the elements here, look at all the elements here;

utmost there will be r 2 elements. I will be able to find out an element which can be put here; that is one possibility, but this may not be true. So, this is not true, then what will happen. So, either is this or, or is a next condition; or says there exists.

Now this is important, because once I able to prove this, I can do the rearrangements; this gives that idea or and we have to prove this always happen. This is always true; there exists a symbol c in row a, which is not found in column b and I think remaining step I can write here.

(Refer Slide Time: 35:15)



and a symbol 'd' in column 'b' which is not found in row 'a'.

For case (2), r_2 symbols must be found in row and column together.

$r_2 > m_1 - 1$, there are at most $m_1 - 1$ symbols in row 'a'; there must be a 'd' in column 'b', which is not present in row a.

Using symmetric argument, there must be a 'c' in row 'a', which is not present in column 'b'.

Further, actually it continues from here, okay, on that side and a symbol d in column b which is not found in row a. What actually it means is either you will able to find out the element which is not here as well as not here, you can put that; if that is not true, it actually means all elements you take here, all elements you take here, you make a union of all these elements; all elements are consumed. There is no free element; that is what. If the free element would have been there, after combining these that would have put here; that is the case one.

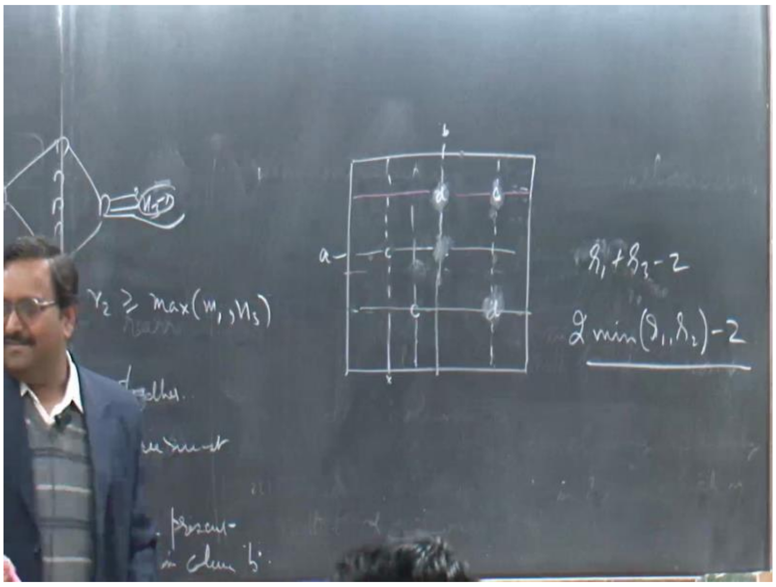
Case two can only happen if you take all the elements here and all the elements here; put all of them together in one basket, all elements are consumed. There is no free element which can be put here. And if that happens, you will always find out an element which is missing here but which is present in the column. So, that is what I am saying. There exists a symbol c in the row a; there is existing some symbol c here and this c is not present in column b. And also correspondingly, there will be another symbol d which will be existing which is there somewhere in the column, but which is not present in the row. That actually means you can always find out a pair; you can always find out a pair.

Now proof for this is going to be very simple. Once if I am able to prove it, I can then as a consequence prove that I can make the rearrangement; a new connection can always be set up. So, this actually means. So, now let us go to the proof. So, for case two, r 2 symbols must be found I think; this should be must, must be found in row and column together if first is not true. And since, what is happening? You are having a switch; how many of the input ports are busy m 1 minus 1. So, there is one which is available one port and how many symbols are there in total? R 2 symbols, okay.

And this actually means because of this condition that only m 1 minus 1; this actually means since r 2 is greater than m 1 minus 1, it has to be r 2 is either equal to m 1 when m 1 is equal to n 1, okay. So, it has to be always greater than m 1 minus 1; otherwise, the switch itself will become blocking, okay. So, for rearrangeably non-blocking, this has to be true. So, once this is true, there are utmost m 1 minus 1 symbols in row a; only m 1 minus 1 symbols are there, and r 2 is greater than this. So, here has to be some symbol which is not present in the row, but when you are combining row and column, you are consuming all r 2 symbols.

So, that symbol should be present in the column, okay. So, my first statement; sorry, this first statement this is about symbol d which exists in column b and not found in row is true. Look at from the output side they are n 3. So, n 3 minus 1 must be occupied. So, one is free which you want to connect same rationally that r 2 is using same principle; I call it using symmetric argument. So, this actually, here you can write there must be a d in column b which is not present in row a, and using symmetric argument, there must be a c in row a, which is not present in column b. So, one of the two conditions will always be true; we have proven it, a very simple argument gives it, okay. So, once this is there, I can now figure out what we call rearrangement procedure.

(Refer Slide Time: 42:09)



If case one is true, you can find out an element neither in row and nor in column, place that here, and you can set up the connection. If there is some element of that kind exists, I can set up the connection; if that is not true, I can always find out a c and a d pair. Second condition will always be true, okay; one of the two always has to happen. So, first case is straightforward; second case we will find out a pair, look into this row. Can a d repeat? Can a c be here actually now? No, I can always find out a c; if in worst case if c is not there, for example, in this column there is no c, very nice; I can always replace this c here and d here. C was not there in the column. So, I can always use that. I can do the rearrangement, but if I do the c that two c's cannot be there in a row.

So, what we will do is we will not do the replacement first; we will find out d, we will search for c here, okay. We will search for c somewhere if c is there. If c is not available, my search stops. If I can find out a c, I can find out search in the whole column, whether there is d available there somewhere or not. Now you look into this row. Now one of the important thing when you are going to search for this column, you cannot find a c here; c cannot repeat in a column the valid Paull's matrix condition, c cannot be here. So, you will never come to this column back again.

When you have searched, you come here d you will be searching a new column. This column will be excluded; this column will be excluded. When you search for a d, you cannot search for this row, because d was not present here, okay, and d is already here.

So, d cannot be there. You cannot come back to any other older searched row; that is not possible actually, okay. So, now once you come to d and you will again search for c; this column is avoided; this column is avoided. You will find out somewhere here anywhere c. You will search now in this column for a d, okay.

You will find out a d; d cannot be here. It is a unique cast connection. I am not looking at a multicast scenario; I am looking at a unicast scenario, okay. So, it means d can rightly it will not be coming here; I will be doing search somewhere here. And I will keep on doing search, and search will stop; that is guaranteed. Every time I am visiting a row or columns that is excluded from the search and maximum rows and columns are you forget this and this is r 1 plus r 3 minus 2. These are the new columns which will be there in rows and columns; worst case, only those many searches will be required. In fact, it will be even lesser than this; I will tell you how that will be done, okay.

So, once this is there, and at some point, you will stop; you cannot find. [FL] You have found d; there is no c here. So, you stop at this point. Now what has to be done? Replace this by c; this by d; this by c; this by d; this by c, and put the d here. You can set up a connection. This is the rearrangement; this is the rearrangement which has been done. Now I think on graph it is fine, but let me show you out on a switch; that is the best way of doing it, okay. So, are you able to appreciate this, no not clear, okay. So, conditions are okay, the text thing which I have done that when r 2 is greater than or equal to maximum of m 1 and n 3, maximum of these two.

So, first condition is when this condition is satisfied, I should be able to get an element which is not there in row a and column b; if that is there, I will just put that element and can set up the connection. If that is not true, I can find out two element pairs c and d, for example, here. So, if c is in the row, it will not be in the column, and correspondingly, d should be in the column, and it will not be in the row. Now why that happens is because r 2 symbols must be there taking rows and column together, and row cannot contain more than m 1 minus 1 symbol. And my condition is it is maximum of m 1 and n 3. So, there is one additional symbol which is not present there which is d.

Now if you look at the column, similarly, column will have n 3 minus 1 symbol, but together I am already consuming all r 2 symbols and my r 2 is again greater than maximum of m 1 and n 3. So, there has to be a symbol c which is not in the column, but it is in the row. So, I can find out c and d pair. So, I have found this c and d pair, okay. So, usually what I can do is I can just use, replace this c by d, because c is not used here. So, I can use that c for this connection.

So, whatever connection being dumped by this d can be now made through a c, and I can put a d here to set up the connection between a and b; that is the rearrangement, but the problem is this row itself might be having somewhere another c. So, what will happen to that? That c was already used for that connection, okay. So, I will replace this by d because d was already used here. So, it certainly has become d has become free; d has been freedom by assigning its connection to c, okay. So, I can do this, but then there can be a d here. So, I am now starting swapping the connection; I am doing rearrangement.

I can keep on doing it till I stop and I have proven that every time when I am going, I am trying to find out a corresponding element. So, if I am here c in the row I am searching for a d; if there is a d, I am in the column searching for a c I am always going to a new row or new column. So, in worst case, I will require these many searches, how we get this, okay. Let me do it again. So, at least till this point you agree c and d pair, okay. So, what we have to do is I have to just use this, put a c here and put d here; that would have made the rearrangement, but I cannot do that all the time. I can only use c here if it has already not been used in this corresponding row.

C can only be placed; d can only be replaced by c if c has not been used there. So, I will search for the c if it has been used. I have to take care of that c in first because c cannot happen twice; it has to happen only once. If the c would not have been there in this row, life would have been very simple; c here and d here, but that is not happening. So, if the d is here. So, I will search for c; I will first of all have to handle this c. So, what I can do is I can now search for the column, find out if there is another d. So, what I will do is I will replace this c by a d; this by c, and I can then put a d here and this by a c, because then in this case, this row should not have another c.

So, best is you actually now search for all these cross elements; d you will search for a c, you will never visit this particular column. This is excluded; c cannot happen twice in a column, okay. I am now visiting this particular column; when I am now searching for a row, I cannot visit this row, because d was not present here. I cannot visit this row, because d is already there. I will be visiting some other row, say, here, okay. I will search for a c; I cannot come here; I cannot here. This is already having a c. So, I might be getting somewhere here, for example, a c. I will now search; I will keep on doing it how many times?

Every time, I am ending up in either a new row or new column. I cannot revisit the older rows or older columns. In fact, somebody who is smart enough will figure out; this is incorrect by this time.

Student: It will be divided by 2, sir.

No, actually, okay, you will think what will be the correct value; this is not the correct value. You think; I will come back to this. This value is not correct, okay; this value is not correct. No no no no no. Okay, I will write, you think how this comes, okay. In fact, this is also incorrect; there is still better value, but anyway at least you should be able to get from here to here. I am only doing searching in the bunch; I am actually searching a chain remember. I am doing a chain search, okay, and chain will finish when either I will compute all rows or all columns. So, whichever is the minimum, I will deal only those many.

So, that is why two into one row one column, one row one column; that is how I am doing the search. If numbers of rows are less, I will finish first. So, I am taking minimum of those two and multiplied by two. This will actually become equal to this when r one is equal to r three, symmetric case, but actually this should be modified to this, but this also is an incorrect result.

Student: When are you multiplying with?

No, I got this. So, only one row has been done. When I am coming here, I am searching a column not the row. So, two elements have been done in one row, then I am doing next row; two elements similarly done in this row. So, when you look at the number of rows, every time two has been searched two elements. If the numbers of rows are less, I will finish up the rows all the rows, then I will stop. This one is not included, okay, and there is a final one which also will not be included actually; somewhere you will stop. So, it will become minus 2 because of that.

So, idea here is one row one column; sorry, number of rows number of columns minus 2, one is this and one is this. But I think more exact value will be this, okay, but let me come back to how you will do the rearrangements. Search the chain; chain will stop at some point. Replace all c’s by d and all d’s by c in the chain; your d will get freedom and put that here to setup the connection, okay. Now regarding how you will do number of searches which are required; this is known as Paull's theorem. So, everything becomes a theorem here and it is an argument which is the proof, or you can write it formally; it does not matter. So, I will use two colors here to actually identify.

Now what we will do is we will do slightly something smarter. So, there is a d here. What I will do is I will start with d, and find out if there is a c. I get find out a c. Now I will not search here; I will search here. Find out a d; I got a d. Alternately, I am extending the chains; once this, then this, then this, then this. So, next I will find out a d. Now I cannot visit this particular column row; it has to be something else. So, it is here, okay, then this will find out a c. Can I visit this column? No, then I will search for a c; I cannot visit this column. I cannot visit this; I will be visiting somewhere here, okay. So, now I am trying to cover up; when I will stop? If r 1 is less r 1 or r 3; so, if r one is less, I will stop at r 3 minus 1 search, okay.

So, what I will require is minimum of r 1 r 3 minus 1; this is the exact value, and that is what is simply the Paull’s theorem. Paull's theorem says that if that condition is satisfied, you will require only these many utmost these many rearrangements to set up a connection. So, you have come from here to here and here to here, but this comes from a double chain search alternative search. So, one of the chain will finish first. In worst case scenario, this will be doing doing; only these many changes will be required rearrangements. This is known as Paull's theorem in the text, Slepian-Duguid theorem; that is q part of Slepian-Duguid theorem.

Student: Double chain is Paull’s theorem.

Double this change is Paull’s theorem; that is his invention. So, I think this finishes with the Slepian-Duguid theorem and Paull’s theorem, and the next lecture we will now go to recursive construction of rearrangeably non-blocking switches clos network as well as strictly non-blocking clos network both. And we will try to estimate what is known as cross point complexity of recursively constructed switches which will be better than o n base power 3 by 2. So, that is the best which we got through over a clos thing if you remember earlier. So, you will get something better than this actually by recursive construction. So, that is the part which I will be covering in the next lecture.