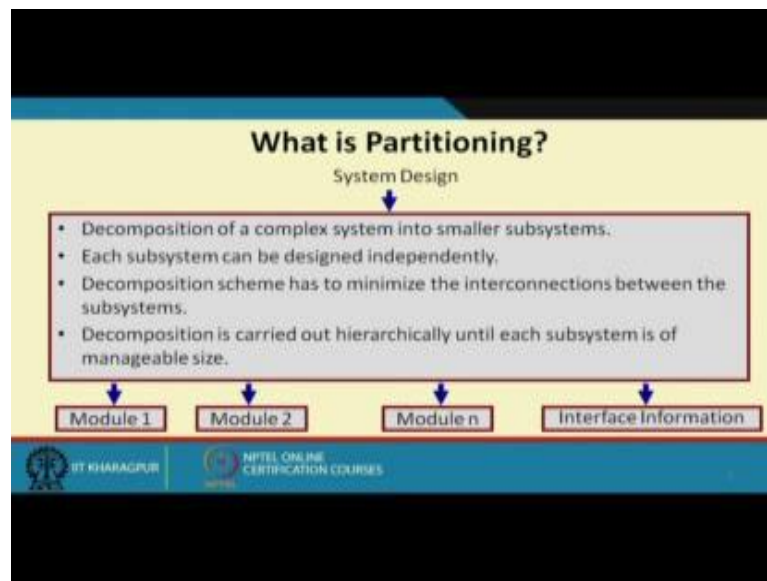


VLSI Physical Design
Prof Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 07
Partitioning

So, continuing with our discussion on VLSI physical design. In this second week we shall be looking at some of the, you can say initial steps of design automation at the backend design liable, namely we shall be looking at the problems of floor planning placement and of course, before that we need circuit partitioning. So we start our discussion with a lecture on partitioning which is usually the first step in this process. So let us first try to explain this scope of the partitioning problem and what does it involve.

(Refer Slide Time: 01:10)

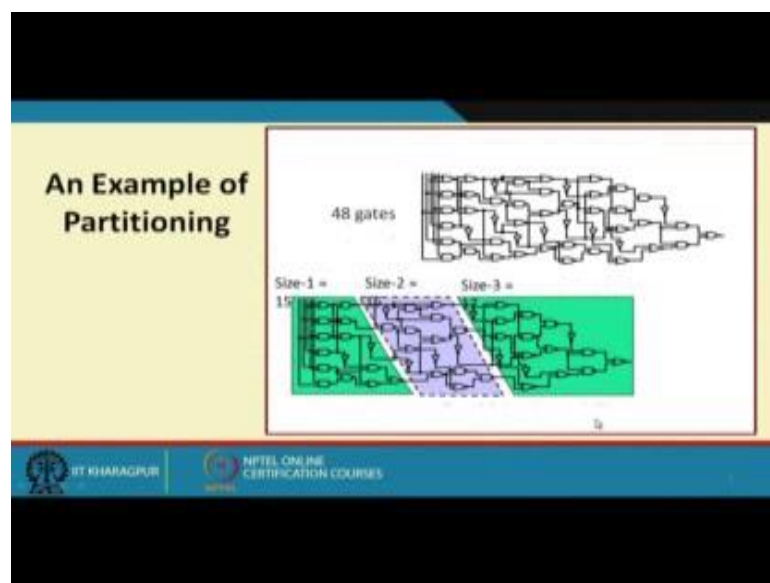


So, when we talk about partitioning, we take any kind of a system design netlist, it can be a netlist at the level of gates at the level of transistors or in fact, any kind of modules and blocks. So when we say we are doing partitioning, we are essentially decomposing the system into smaller and more manageable subsystems. Such that each of the subsystems can be handled designed and laid out independently, but there are a few criteria that needs to be satisfied in this process, so while during the decomposition we have to care that the number of connections between these partitions or subsystems must be minimized. And this decomposition we can carry out hierarchically making the blocks

smaller and smaller, until we reach a stage where each of the blocks can be handled and designed independently.

So starting with a large system we continually go on partitioning it in smaller and smaller pieces, until we reach a stage where all the pieces are of manageable size. So, as the output of partitioning, we get the individual module netlists let us say there are n such modules and of course, how they are interconnected this we call as the interface information.

(Refer Slide Time: 02:50)




Let us look at the same example we had a look earlier. So we illustrate the problem of partitioning with respect to this gate level netlist that we are seeing here. So we have a circuit comprising of 48 gates so in this example we are illustrating how to divide it up into 3 clusters or partitions.

Well, one of the obvious objective is that the partitions has to be roughly of the same size so in this case the sizes of the partitions are 15, 16 and 17. There is another requirement as I had mentioned the number of interconnection line as you can see, between the first and second partitions the number of interconnections are 4 and between second and third it is also 4. So the objective of minimizing the number of connections across partitions is also fairly satisfied. So this example shows roughly how a partition or the partitioning process should split a netlist into a smaller netlist.

So, there are 2 criteria, number one the partitions need to be approximately of the same size and number 2, the number of connections between the partitions has to be approximately equal.

(Refer Slide Time: 04:19)



Partitioning at Different Levels

- Can be done at multiple levels:
 - System level
 - Board level
 - Chip level
- Delay implications are different:
 - Intrachip : X
 - Intraboard : 10X
 - Interboard : 20X

BT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

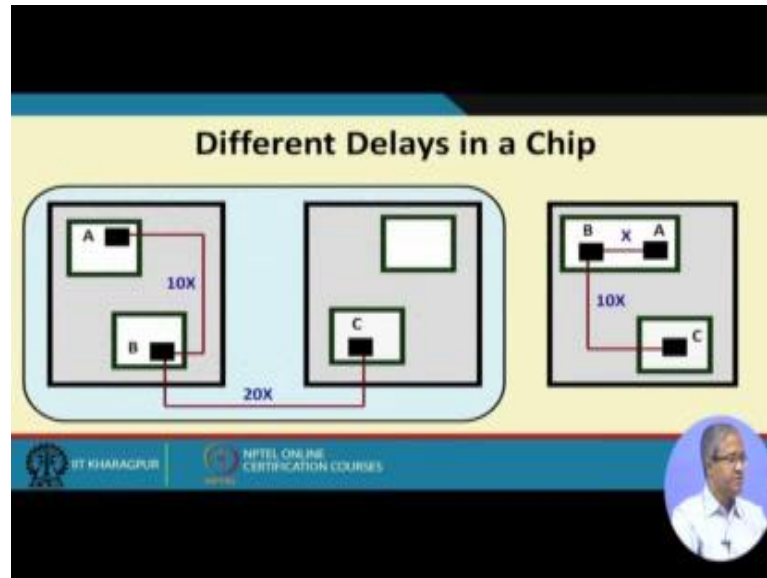
So, when we talk about partition this process can be carried out at different levels for example, when you design a system we can carry it out at the system level the whole system design we can partition into subsystems, but each of the subsystems can be possibly be mapped into a printed circuit board. Once we have a PCB or a board, then we can partition at the level of the board. So inside the board we can see that there can be a number of chips. And even inside a chip when you are designing a chip there can be a number of blocks or modules so a connection of them.

So, you can say the partitioning can be done at the system level at the board level or even inside the chip at the chip level. And the point to note is that when you have a large system usually the total circuit is divided across a number of printed circuit boards whereby we get the system.

The delays are important well if we are connecting 2 points inside a chip suppose the delay is X, but when you are going across chips between 2 chips within in a single board the delay can be roughly 10 times, but when we go across boards, it can be as large as 20 times or even more. So you can see that the delay can be of the order of the magnitude higher as we go for you can say intra chip routing to intra boards and across boards. So it

is important to ensure that the critical nets that we have they have to be put within the low delay regions as far as possible; that means, inside the chip it is preferable, if it not possible only then you can go across chips.

(Refer Slide Time: 06:21)

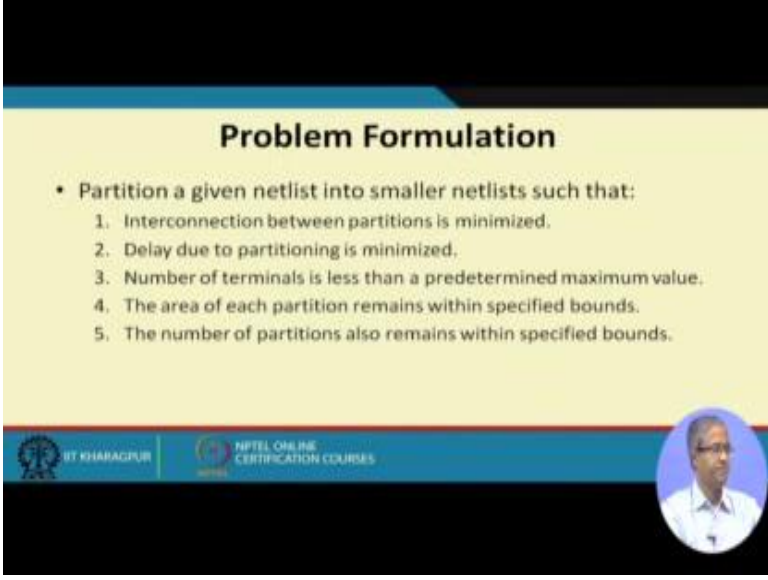


So, a simple illustration here, so, on the left we see board which consists of, this is a system, let us say system which consists of 2 boards. And each of this boards consist of some chips. So I am showing some connections so a connection between a block a inside this chip and A block B inside this chip this will incur a delay of 10X. Similarly, connection between this B here and C on the next board it can be 20X. So from A to C the total delay can be 10 plus 20, but suppose in an alternate mapping if we put A and B within the same partition, then the connection between A and B the delay can be only X. And also the C if you can put it on the same board then the delay between B and C can be B and C can be only 10. So instead of 30X here the delay between A and C becomes 11X. So this shows you so how the critical nets or the higher delay parts can be put means inside a chip or within a board, so as to minimize the delay as much as possible.

Now, one thing we should also remember, well we normally talk about the critical paths. The paths which take maximum time for signal to propagate; the critical paths typically determine the maximum frequency of operation of the circuit. Now the way we have said we are trying to break a critical path or refine a critical path by putting them into higher speed sections of the circuit. So that way our critical path can become of smaller sizes.

But one thing you should also remember, well in doing this some of the other paths which were earlier inside the chip may go across the chip and their delay might increase in the process, some of the paths which were not critical earlier might become critical after this change or modification.

(Refer Slide Time: 08:44)



The slide is titled "Problem Formulation" and lists the following requirements for partitioning a netlist:

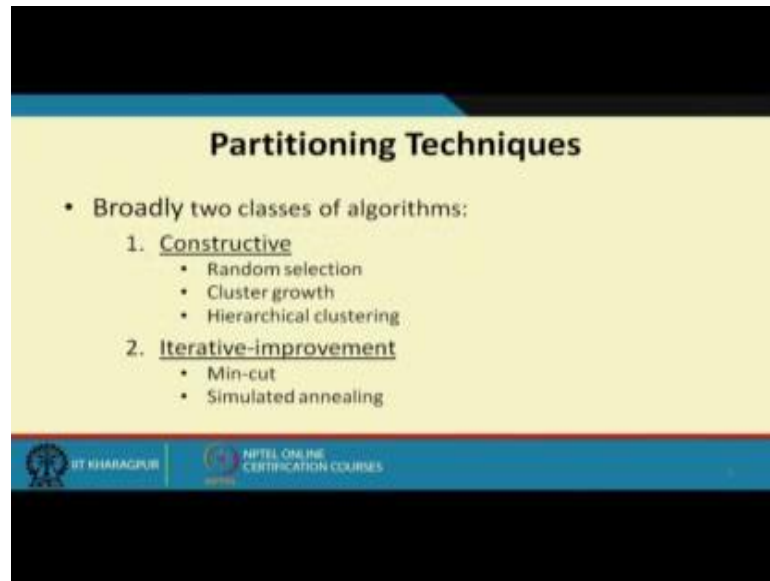
- Partition a given netlist into smaller netlists such that:
 1. Interconnection between partitions is minimized.
 2. Delay due to partitioning is minimized.
 3. Number of terminals is less than a predetermined maximum value.
 4. The area of each partition remains within specified bounds.
 5. The number of partitions also remains within specified bounds.

The slide also features logos for IIT Kharagpur and NPTEL Online Certification Courses, along with a circular portrait of a man in the bottom right corner.

So, the partitioning problem, if we want to formulate, so we can say it like this, so we are given a netlist, we are wanting to partition this netlist into a set of smaller netlists with a number of these requirements to be satisfied.

The number of connections between the partitions they have to be minimized, delay due to partitioning this I have just now mentioned the critical path delays this has to be minimized, at least the signal nets which are critical which determine the clock frequency. And each chip or board usually has a limit to the number of interconnecting terminals that you can have. So the number of terminals has to be within that maximum value limit. And of course, each of the partition should fit a chip or a board so it this can have some maximum upper bound in terms of the size or area. And also the total number of partitions that you are allowed to have that can also be specified, that you can have this many number of chips in which you can partition a whole design or this many number of boards not more than that, so these are the restrictions.

(Refer Slide Time: 10:04)



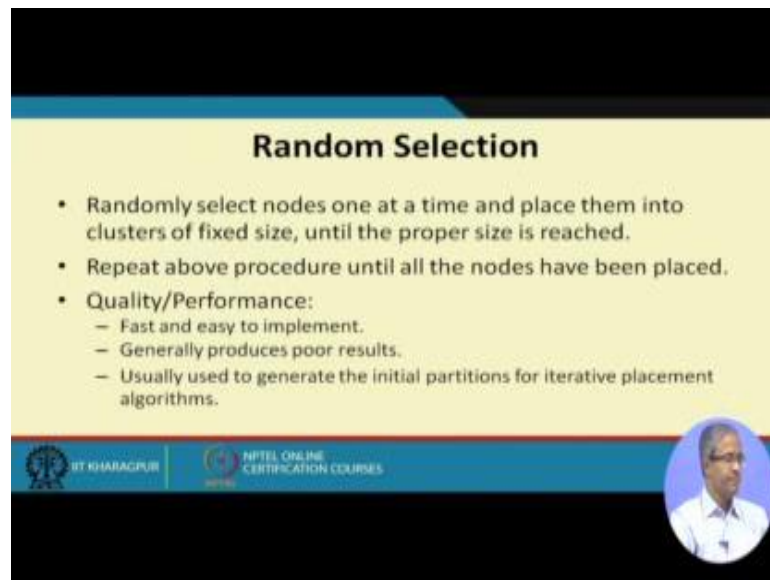
Partitioning Techniques

- Broadly two classes of algorithms:
 1. Constructive
 - Random selection
 - Cluster growth
 - Hierarchical clustering
 2. Iterative-improvement
 - Min-cut
 - Simulated annealing

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

so talking about the partitioning techniques, broadly the techniques can be classified as either constructive or something called iterative improvement. Constructive placement means we are starting with nothing we are starting with an empty partition, and we slowly add blocks or modules to create the partitions bigger and bigger in that way we allow the partitions to grow, but in other hand there is a second class of algorithms these are called iterative improvement. Here the idea is that we start with an initial partition we have several partitions already existing to start with, and as part of these algorithms we try to improve the quality of the partitions by making changes incrementally and iteratively on this given set of blocks and the partitioning.

(Refer Slide Time: 11:15)



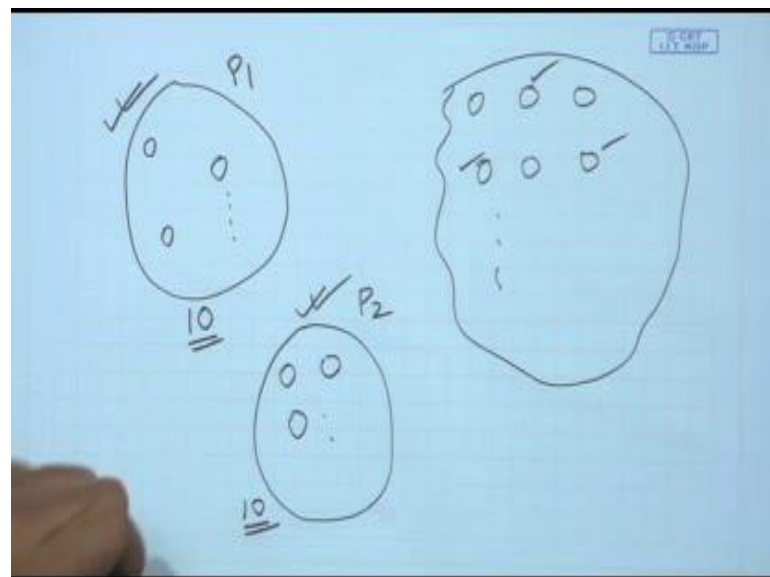
Random Selection

- Randomly select nodes one at a time and place them into clusters of fixed size, until the proper size is reached.
- Repeat above procedure until all the nodes have been placed.
- Quality/Performance:
 - Fast and easy to implement.
 - Generally produces poor results.
 - Usually used to generate the initial partitions for iterative placement algorithms.

Logos for IIT Kharagpur and NPTEL Online Certification Courses are visible at the bottom left. A circular portrait of a man is at the bottom right.

So, let us look at some of these methods one by one. Well random selection is very simple. This says that you have a set of nodes. So you randomly select the nodes one at a time and you go on placing them into clusters of fixed size, until the proper size is reached. What does this mean?

(Refer Slide Time: 11:38)



Let us say suppose I have a requirement that inside a cluster I can have up to 10 blocks. And suppose I have a set of blocks to place, there are many such blocks. So what I do from this set I randomly pick one, I place. I randomly pick another, I place. I randomly

pick another, place. In this way I continue till this limit of 10 is reached. So once this 10 is reached I can say that my partition P 1 is done. Now I move to my next partition P 2. So in a similar way I again pick the blocks randomly I place them here, until my limit of 10 is again reached. So my P 2 is done. Then I move to P 3, P 4 and so on. So this method you can see is very simple and pretty obvious. And quite naturally the way we were doing it. You were doing it entirely randomly. We were not looking at the property of the blocks the way they are connected and so on. So usually the quality of the partitions that are generated in this process is not so good.

So we look at another method which is better in that respect, and we call it cluster growth.

(Refer Slide Time: 13:02)

Cluster Growth

- Starting with a single node, add other nodes to form the partitions based on connectivity.
- Number of clusters n can be an input parameter.

m : size of each cluster, *V* : set of nodes

```

n = |V| / m;
for (i=1; i<=n; i++)
  seed = vertex in V with maximum degree;
  Vi = {seed};
  V = V - {seed};
  for (j=1; j<=m; j++)
    t = vertex in V maximally connected to Vi;
    Vi = Vi ∪ {t};
    V = V - {t};
  endfor
endfor

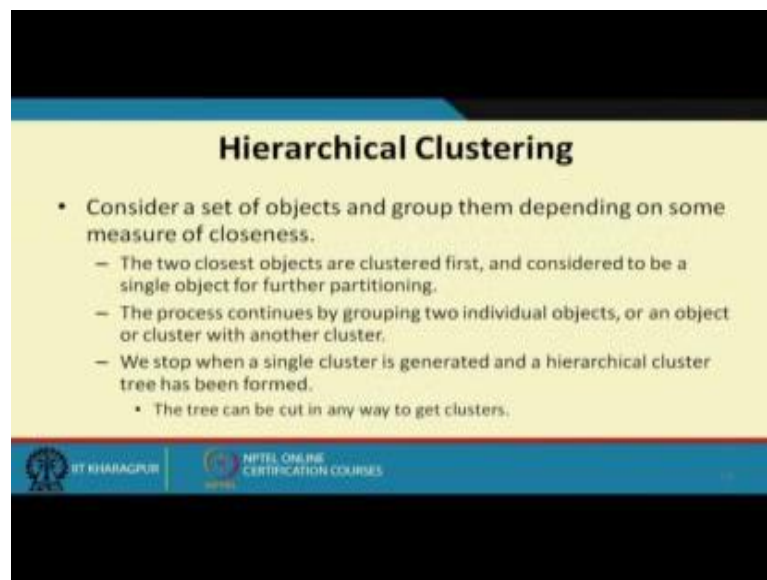
```

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, in the method of cluster growth what we do, we start with a single node and add other nodes to form partitions, but not randomly based on connectivity. And the number of clusters you want to divide that can also be in input parameter. So let us look into the outline of this algorithm which has been shown here. Here let us say that we have the set of nodes let us call it capital V . And m denotes the size of each cluster. So the number of partitions will be the size of V divide by m . So what you do for each partition for i equal to 1 to n , we repeat. So we initialize a variable $seed$ as the vertex in this set of what is V which has the maximum degree. Degree means it is connected to maximum other blocks, the block which is maximally connected to other blocks.

You select that vertex and let that vertex be my initial seed for that partition. I call it V_i . V_i denotes the i th partition. So once I select this one, I remove this seed from the original V I take it out. Then since the size of each cluster is m so I have to add this m minus 1 in fact, j less than m , this m minus 1 remaining vertices to V_i . So what I do at every step I check and find out a vertex t , which is maximally connected to the vertices which are already there in V_i . So what I do? We take the union of t with V_i , and we take out this t from V repeatedly and once you complete this process this said V will be empty, and we get our just desired cluster. So this is one very simple method depending on the connectivity we try to create the clusters and we grow the clusters in size iteratively one by one.

(Refer Slide Time: 15:35)



Hierarchical Clustering

- Consider a set of objects and group them depending on some measure of closeness.
 - The two closest objects are clustered first, and considered to be a single object for further partitioning.
 - The process continues by grouping two individual objects, or an object or cluster with another cluster.
 - We stop when a single cluster is generated and a hierarchical cluster tree has been formed.
 - The tree can be cut in any way to get clusters.

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

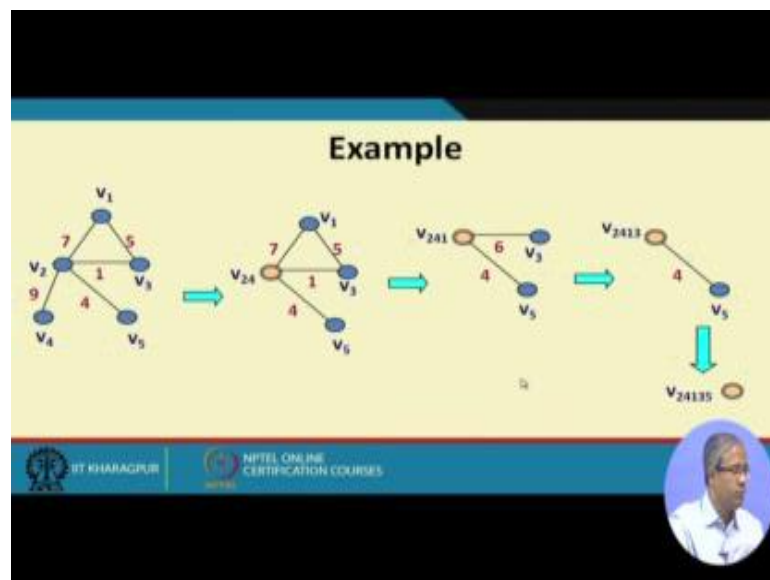
Now let us move into some methods which are little more practical, in the sense that we have lot more flexibility. Here this classes of methods are called hierarchical clustering. So what they do they do something like this? You consider a set of modules or objects and group them depending on connectivity means closeness. Like suppose if there are 2 blocks between which you said that there are 10 connections, they will always want those 2 blocks to remain together closer together. So this is the measure of closeness.

The blocks which are more heavily connected they should be kept closer together this is the basic idea. So what you do we carry our cluster in a hierarchical way. The 2 closest objects in terms of the connectivity are clustered first, and once we do this this sphere of

objects are merged and considered as a single object subsequently. And you repeat this process just one by one you try to select 2 vertices which are closest in my remaining netlist and you merge them together and you keep the information in which order you are merging the vertices, because you will be using this later to do the actual partitioning.

So, you repeat this process, and you stop when subsequently a single cluster is generated and something called a hierarchical cluster tree has been formed, a cluster tree which indicates this sequence of object pairs which have been merged to generate the single cluster. So I am showing an example to illustrate. Then once you do this you can cut the tree to form 2 or more clusters. Well let us see how it is done.

(Refer Slide Time: 17:44)

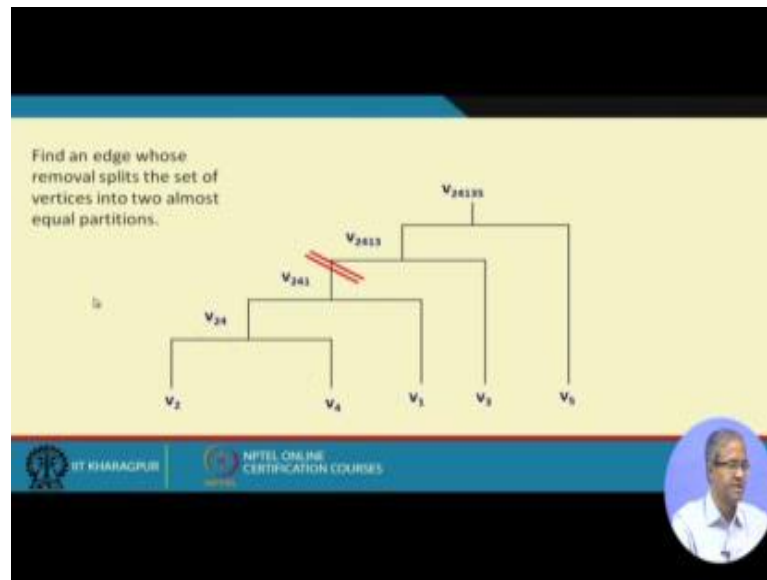


Let us take an example like this. Where this 5 vertices indicate some small netlists or some basic elements there can be gate there can be small set of gates. And this numbers they indicate the number of connections or the closeness. This 9 indicates there are 9 connections between V 2 and V 4. This 1 indicates there is one connection between V 2 and V 3 and so on. Now once you do this, you see that which pair of vertices are the closest. V 2 and V 4 are the closest you merge these 2 first.

So the first step what we do merge this sphere and generate a composite vertex called V 24. We repeat this process in the remaining graph you see which one is the closest this 7, V 1 and V 24 merge these 2 so we get V 241. So in the remaining graph, well once you see one thing once you merge V 1 and V 24, you get V 241. So the weight of the h

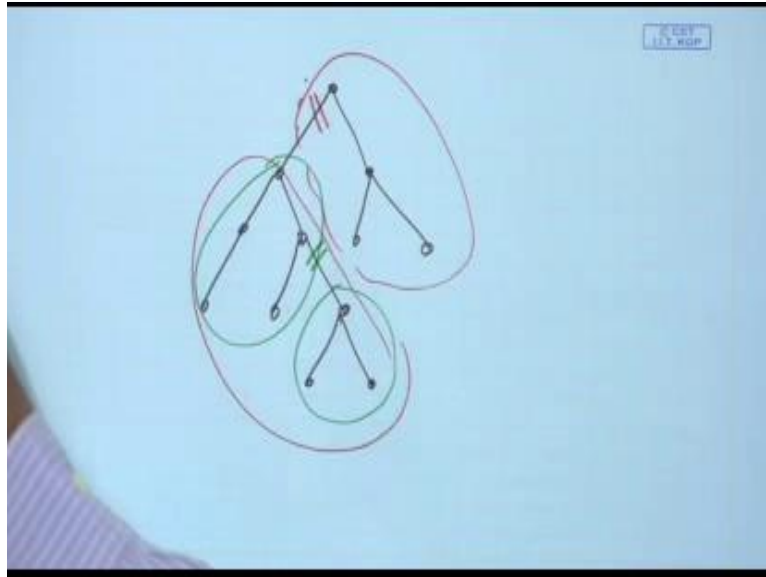
between V_{24} and V_3 will be, this 1 plus 5 6. Because we have merged these 2 so the number of connections between these 2 is now 6. So next one is 6 highest do this, then remaining 4 so you finally, do this.

(Refer Slide Time: 19:35)



So, this sequence of vertices you are merging 2 4 then 1 then 6 3 then 3 then 5. So you remember this sequence and you generate something called you can say clustering tree. So initially you merge V_2 and V_4 to get a node V_{24} . Merge V_{24} and V_1 get this in this way. So once we have this tree, then you can take a decision you can cut this tree any edge you cut, it will divide up into 2 parts, because you know a tree is a kind of a graph where there is a unique path between 2 vertices. If you cut any edge it will divide that it up into 2 parts.

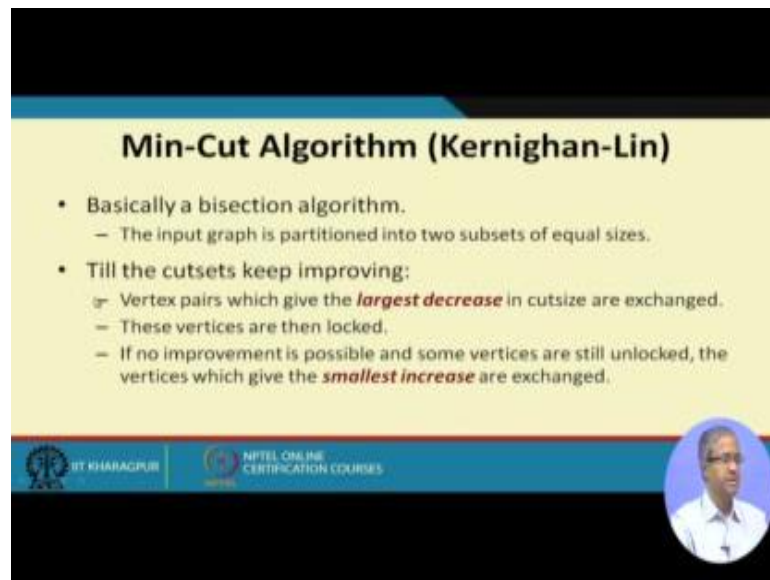
(Refer Slide Time: 20:15)



Now, suppose if you have a tree like this. Let us say you have a tree like this. It is just example I am giving. Suppose if I have a tree like this. So once you have a tree like this suppose I want to divide it up into 3 parts. So what you can do I can make one cut here, this you can see will divide the tree into 2 parts, so one is this part and the other will be this part, next what I can do I can make another cut let us say, here this will break this up into one cluster like this another cluster like this. So every time you cut an edge in this tree you will get one more partition or cluster generated.

so here also in this example let us say we want to divide into 2 parts we make a cut here so you get one cluster comprising of V 2 V 4 and V 1, another cluster comprising of V 3 and V 5. Now this clusters will be such that the once which are more heavily connected you are trying to keep them together right. So this is the basic idea.

(Refer Slide Time: 21:24)



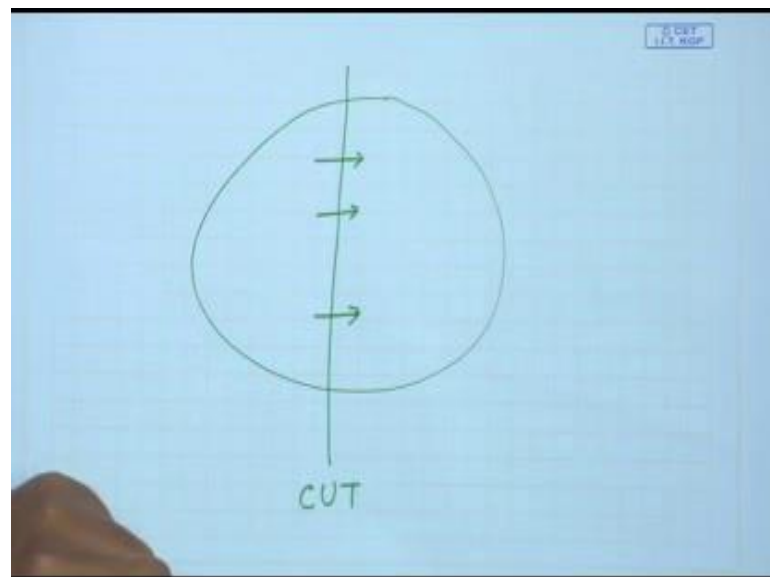
Min-Cut Algorithm (Kernighan-Lin)

- Basically a bisection algorithm.
 - The input graph is partitioned into two subsets of equal sizes.
- Till the cutsets keep improving:
 - Vertex pairs which give the **largest decrease** in cutsize are exchanged.
 - These vertices are then locked.
 - If no improvement is possible and some vertices are still unlocked, the vertices which give the **smallest increase** are exchanged.

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, next, let us go to a very important class of algorithms this is called min cut algorithm, that I am trying to keep this condition in mind that I am trying to do the partitioning in such a way that the number of lines that are going across the partition is minimized, which means.

(Refer Slide Time: 21:50)

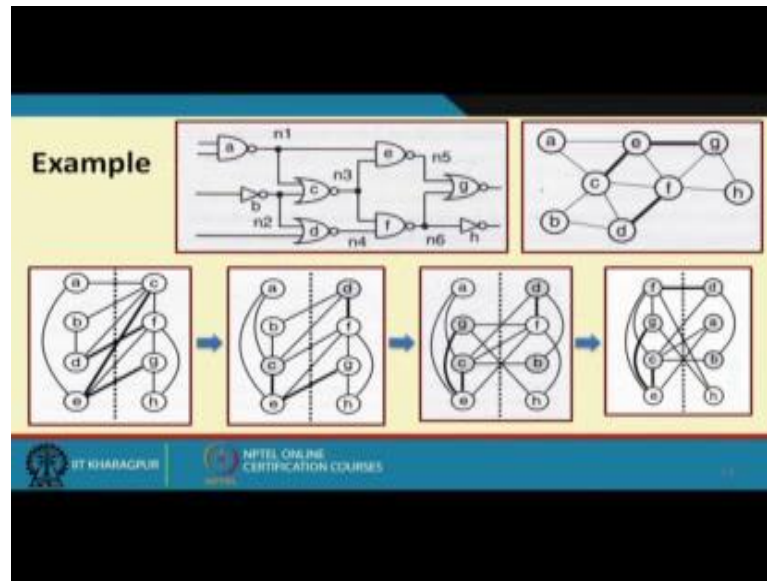


Suppose I have a netlist like this. I am doing a partition like this. I have to see how many signal lines are crossing. This is defined as the cut. So I have to minimize this size of the cut that will be called a good partition.

So this Kernighan Lin algorithm that I am showing here, this is basically doing something like this. And this is a bisection algorithm in the sense that the initial netlist is partitioned into 2 subsets, which will be of equal sizes. The method is very simple here we start with initial partition. Starting with initial partition we repeat iteratively the process till the cut sets keep improving. So what I do we find out the pair of vertices on from each of the partitions, whose exchange will result in a largest decrease in cut size. Like you have 2 sets available with you, 2 sets of vertices which is your initial partition you choose one vertex from this set one vertex from that set, you try to exchange them and see how much improvement you get. You repeat this process for every pair of vertices, and see at every step which pair gives you the best benefit or gain you choose that pair of vertices to exchange.

So, once you find that pair of vertices, you lock this vertex. Lock means those vertices will not participate in any further exchanges in the future, but if you see that no improvements are possible by exchanging pairs of vertices, then you choose a pair of vertex which gives the smallest increase in cost. So here also allowing some increase in cost with the expectation that if you do this may be later on you will get a better solution. So this is a very standard method of trying to avoid something called local minima. There can be a solution space or there can be multiple minimum points. You try to avoid from falling into the local minima, because there can be another minimum which is even better. So sometimes you accept what solution with the expectation that you get a better solution in the future, but here as you go on you remember at every step what is the best solution you have seen so far.

(Refer Slide Time: 24:42)

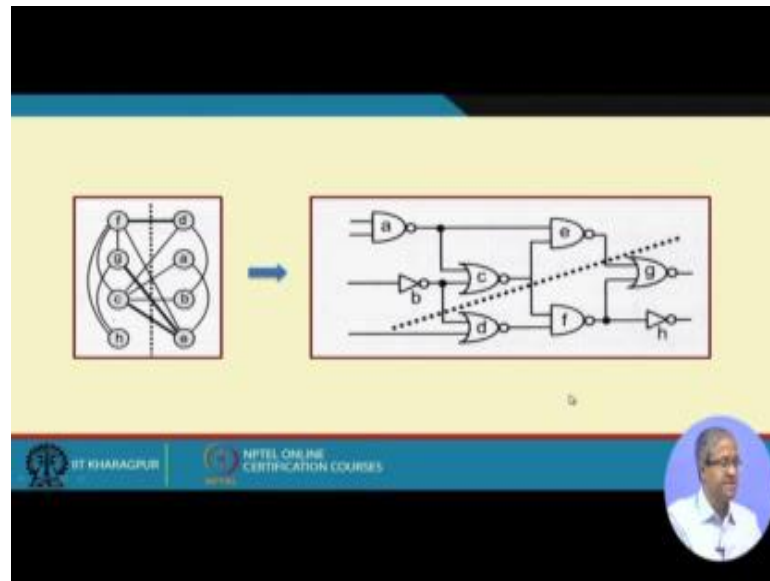


So, let us take an example illustration. So I have a circuit like this which I want to partition into 2 parts. So we construct a graph out of these gates, well here I take make an assumption I assume that the thin edges have a weight of 1 and the thick edges have a weight of 0.5. So let us say it is an assumption that I want that this weight to be less. Let us start with an initial partition like this. So for initial partition if you just count the number of vertices which are cut it is 1 2 3 4 thin edges and 3 thick edges. So the cost will be 4 plus 1.5, 5.5. You pair wise check you try to exchange a and c a f a g a h then b c b b f b g b h and so on you will find that exchanging c and d will give you the maximum benefit. So this step is shown here. If you exchange c and d, c is brought here and d is brought there, and this 2 vertices are shown shaded.

Now you see the cost has dropped down 1 2 3 4 thin edges and one thick edges. The cost is 4.5. In the next step the vertices you are exchanging are g and b. Bring g here b because here you again check the pair whose exchange will either give you the maximum benefit or if you cannot get a benefit the minimum increase in cost. So it is g b. So if you do this you will see that you get 1 2 3 4 5. 1 2 3 4 5 6 so the cost will be 6.

So, in a similar way you proceed. So in the next step you exchange this f, and you have this you exchange this f and a bring f here a here, so the cost which is again increasing. Then in the last step you exchange the remaining 2 you get this final one. Now in this process we will see that, this cost is the minimum one you have seen so far.

(Refer Slide Time: 27:04)



So, you have written this a b c e in one partition. So you declare this as your final partition a b c e in one the rest in the other. So this is basically what is Kernighan Lin by partitioning algorithm. The drawback of this algorithm is that, this is not applicable to hyper graph directly. Hyper graph means there are more than 2 nodes which are connected together. That is called a hyper edge there is an edge connecting 3 vertices.

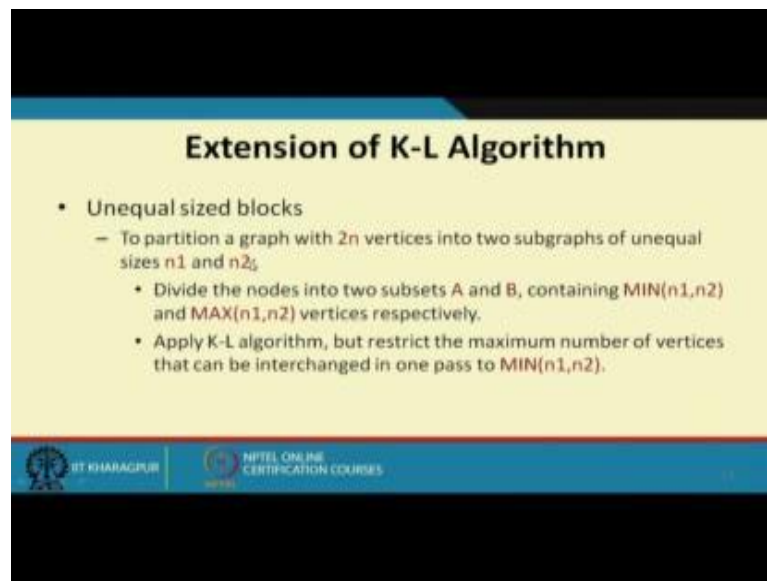
(Refer Slide Time: 27:35)

- Drawbacks of K-L Algorithm
 - It is not applicable for hyper-graphs.
 - It considers edges instead of hyper-edges.
 - It cannot handle arbitrarily weighted graphs.
 - Partition sizes have to be specified a priori.
 - Time complexity is high :: $O(n^3)$.
 - It considers balanced partitions only.

So, Kernighan Lin algorithm does not consider this hyper graph directly. This is one drawback. And it cannot handle arbitrarily weighted graph, although the example that we

have seen has weight. So it can handle, but the calculation will be slightly more complex. And the partition sizes must be known before end. The time complexity is high in terms of the number of nodes there will be maximum n by 2 iterations, in every iteration there will be order n square complexity of selecting the pair, so where all it will be order in cube. It considers partitions of equal sizes balanced.

(Refer Slide Time: 28:35)



Extension of K-L Algorithm

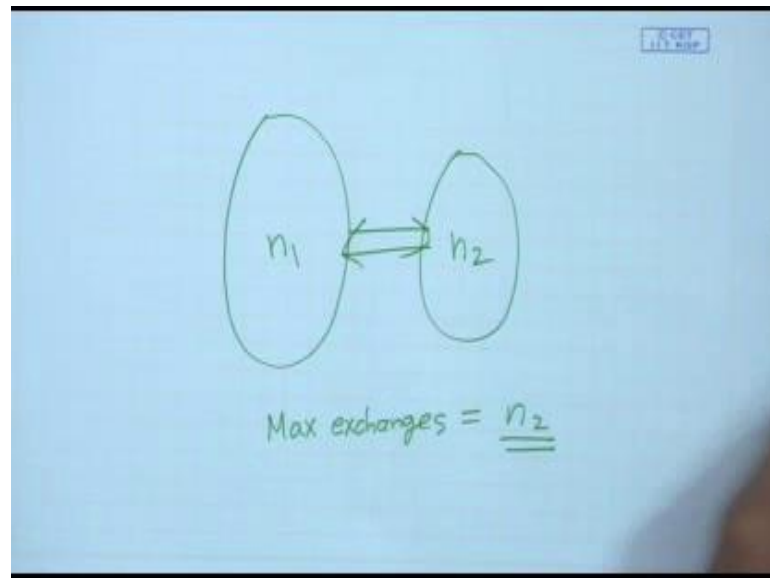
- Unequal sized blocks
 - To partition a graph with $2n$ vertices into two subgraphs of unequal sizes n_1 and n_2 ,
 - Divide the nodes into two subsets A and B, containing $\text{MIN}(n_1, n_2)$ and $\text{MAX}(n_1, n_2)$ vertices respectively.
 - Apply K-L algorithm, but restrict the maximum number of vertices that can be interchanged in one pass to $\text{MIN}(n_1, n_2)$.

BT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, this Kernighan Lin algorithm can be extended in several ways. Firstly, you can consider unequal block sizes. Suppose I have a graph with $2n$ vertices, but I want to partition it into 2 sub graphs, so not equal n and n , but n_1 and n_2 , or n_1 and n_2 are not equal.

So it is something like this. I want to divide it into one part which is bigger.

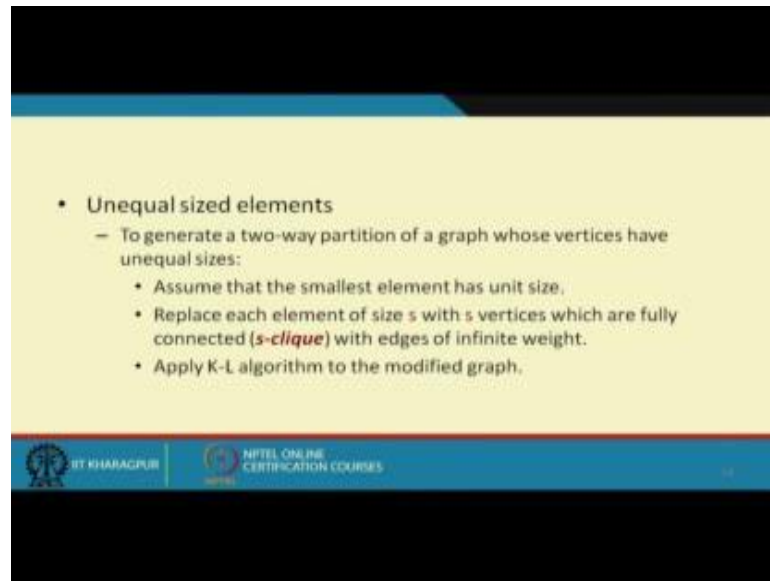
(Refer Slide Time: 29:04)



Let us say there are n_1 vertices here, another partition which will be smaller there will be n_2 vertices here. So here we proceed in a similar way, but if you can see if we had n_1 and n_2 here, for at every step the maximum number of exchanges that we can have, maximum exchanges can only be n_2 here, because n_2 is smaller; after n_2 so all of these n_2 nodes will be locked. So this will be the minimum of this n_1 and n_2 . Since n_2 is smaller it will be n_2 .

So here you see so what we are doing we are dividing the node into 2 subsets containing minimum of n_1 and n_2 maximum of n_1 n_2 vertices one smaller another larger. And this is what I am saying at every step we are limiting the number of vertex exchanges to the minimum of n_1 and n_2 . Just this one change if you make then you will be able to handle this unequal block sizes.

(Refer Slide Time: 30:14)



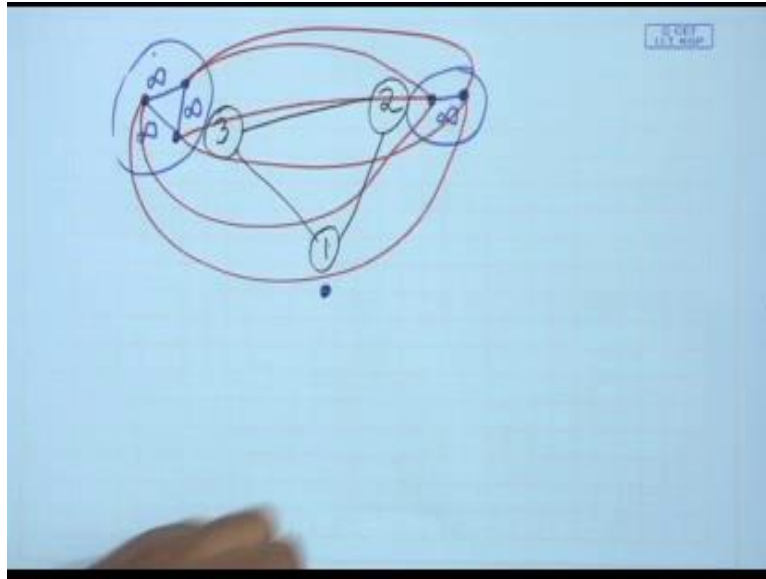
• Unequal sized elements

- To generate a two-way partition of a graph whose vertices have unequal sizes:
 - Assume that the smallest element has unit size.
 - Replace each element of size s with s vertices which are fully connected (s -clique) with edges of infinite weight.
 - Apply K-L algorithm to the modified graph.

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another extension you can have is to handle unequal sized elements. Like so far we have assumed that in the graph so all the vertices are similar they are connected you are swapping exchanging vertices. So their cost will be the same, so that whichever pair you are exchanging. But in general some vertex can indicate not a single gate, but a may be a collection of 3 gates or 4 gates. So the sizes of each of the vertices can be different in general. So in this second variation you are considering unequal sized elements. Here the assumption is like this we assume that the smallest element has unit size. You replace each element of a size s with s vertices which are fully connected. This is called a s -clique with edges of infinite weight. So what I mean I am just explaining.

(Refer Slide Time: 31:15)



Suppose I have a graph like this. Let us say let us take a very small example. A 3 vertices which are connected, a graph like this. Now these nodes are not equal the weight of this is 3 the weight of this is 2 the weight of this is 1 let us. Say so what does this mean this 1 means this is a unit a edge it consists of a 1 gate let us say. This 2 means you replace this by 2 vertices. 3 means you replace it by 3 vertices which are connected among themselves. And this weight s of these edges you would take as very large. Why you are taking very large? Because they will always remain together, they will always remain together. So by doing this you create modified graph.

Here let us say if we just replace this by 3 and this by 2 then this will be connected to this this will also be connected to this. This will be connected to this; this will also be connected to this. This will be connected to this; this will also be connected to this. Like this you make all such connections. You make all such connections you create a new graph and you run this scale or Kernighan Lin algorithm again on that. So the infinite edge pair of vertices they will always remain together which means those clusters which are representing higher voltage they will always remain as single clusters. So, this is one change you can make.

(Refer Slide Time: 32:53)

Performance Driven Partitioning

- On-board delay is much larger than on-chip delay.
 - On-chip delay is of the order of nanoseconds.
 - On-board delay can be in the order of milliseconds.
- If a critical path is cut many times by the partition, the delay in the path may be too large to meet the goals of high-performance systems.
- Goal of partitioning in high-performance systems:
 1. Reduce the cut-size.
 2. Minimize the delay in critical paths.
 3. Timing constraints have to be satisfied.

BIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And of course, this is another important thing which will be discussing later that we can also carry out with this partitioning with an eye towards performance. Because I have already seen that on board delays are much large than on chip delays, typically within a chip delays can be nanoseconds or fraction of nanoseconds, but on board across chips delay can be as large as milliseconds, due to capacitive and resistive effects.

So, as I said earlier if a critical path gets cut many times, the delay can be an unacceptably high. So for high performance systems your partitioning goals can be different. Reducing cut size is of course; yes you have to minimize the delay in the critical paths thereby satisfying the timing constraints.

So with this we come to the end of this lecture. So, we continuing with our discussion on floor planning in the next lecture.

Thank you.