

**Electronic Design Automation**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture No #13**  
**Synthesis: Part VI**

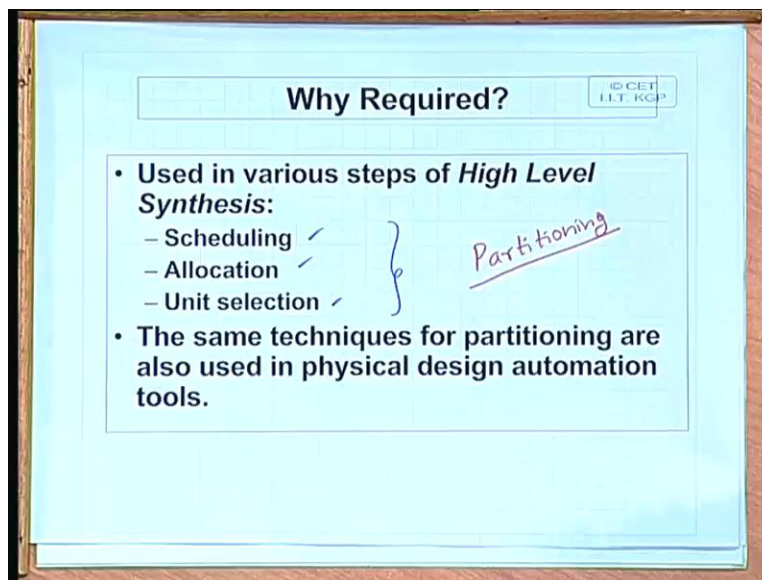
Last lecture we had looked at the way in which we can represent behavioral specification or a design namely through the control and data flow graphs the c d f g. And we had also mentioned that during high level synthesis we need to go through number of different steps like scheduling allocation or unit allocation or binding. And in each of these steps it is that intermediate data structure c d f g which is manipulated or modified in order to get a more and more desirable and more and more efficient version of the same. So we would be looking at the steps one by one. But before that one thing you should remember that if you have a big design big specification to handle then typically the c d f g itself would be very big. So instead of manipulating a very big data structure it is always advisable or better to break the data structure up into smaller parts and then manipulate or process each of them individually.

(Refer Slide Time: 02:19)



So there is the need to do something called partitioning on the data structure during the process of high level synthesis. Now in fact this process of partitioning is useful not only for high level synthesis. But as we would be seeing later that this partitioning is required even for other steps in the VLSI design circuits. You typically during the back end design phase or the physical design step.

(Refer Slide Time: 02:50)



So just as I have just mentioned circuit partitioning is required in various steps of high level synthesis. Scheduling allocation unit selection or binding these are the typical steps in high level synthesis process. And in each of these steps we may need to carry out partitioning. Partitioning of the problem we are tackling divide a bigger problem into smaller sub problem so that they become more manageable okay. And as I just mentioned that the techniques for partitioning that we would be talking about with respect to high level synthesis the same techniques can also be used for physical design automation tools and steps also. So, talking about partitioning to start with.

(Refer Slide Time: 03:52)

**Component Partitioning** © CET I.I.T. RGP

- Given a netlist, create a partition which satisfies some objective function.
  - Clusters almost of equal sizes.
  - Minimum interconnection strength between clusters.
- An example to illustrate the concept.

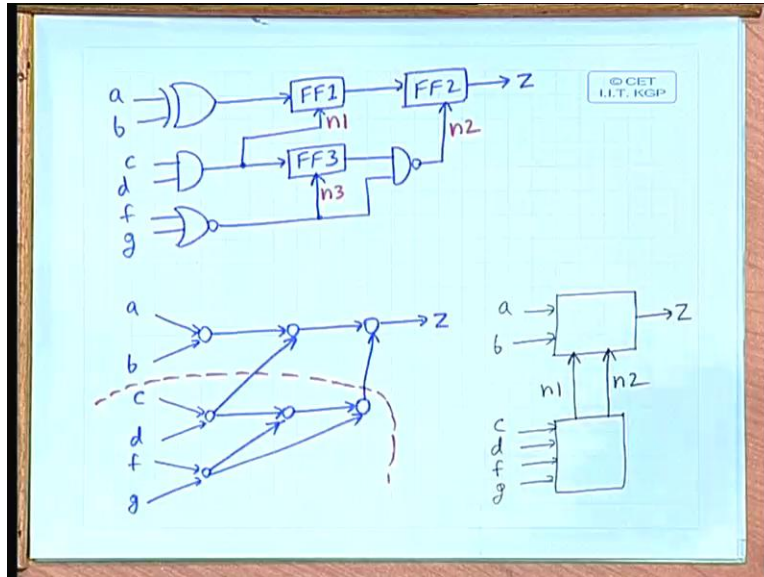
CDFG

Let us try to understand what is the basic problem what do we mean by partitioning. Partitioning means as an input we would be given some kind of a netlist okay although we are calling it a netlist but with respect to hardware description with respect to high level some high level synthesis this can as well be the c d f g. c d f g is also like a netlist comprising of the basic symbols in that flowchart kind of a notation okay. So that is some kind of a netlist some kind of nodes and their interconnection. This also we can call some sort of a netlist. And we have some kind of an objective function which we want to achieve or optimize. See when you talk about partitioning we basically talk about given a big structure of netlist, we want to divide it up into several parts.

And one of the primary objectives of this partitioning is that the different pieces which you call clusters they should be approximately of the same size. Not only that see across the partition boundaries the number of lines which cross that also should be minimum. So if we see that we have made a partition but there are number of lines crossing that will not be not regarded as a good partition. So a good partition is one where the number of lines crossing the partition boundary is also less minimized. So let us take a simple example to illustrate this. Yes [Student

Noise Time: 05:39] lines, I will take an example you will understand you will understand what do I what I mean by the lines crossing I will take an example.

(Refer Slide Time: 05:46)



So let us take an example of a circuit. This will be easier to apprehend. So let us take a netlist of gates. So there are 6 inputs and there are also some flip flops. This is flip flop 1. The output of flip flop 1 is feeding to another flip flop flip flop 2. The output of which is the final output. Similarly the output of this and gate is feeding a third flip flop; flip flop 3. This may be feeding a NAND gate this output. And these may be the clocks of this flip flops. So this is just a hypothetical design. Say this suppose this our original design which you want to partition into 2 parts. Now let us give some names to this clock signals atleast lets call this n 1, this n 2 and this one, n 3. Okay. So this is a netlist of gates but before we start the process of partitioning we, just it is advisable to represent it in a slightly more abstract form.

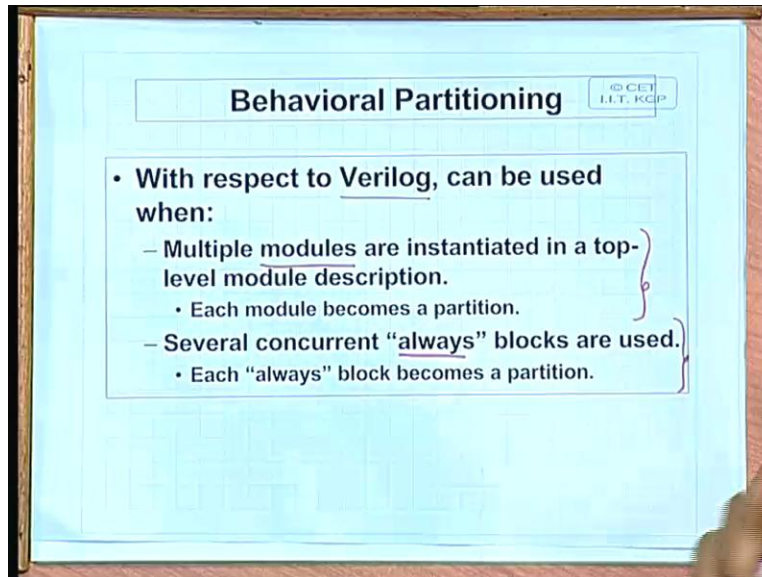
Like the same thing we can represent in the form of a graphs, where the components or the cells will denote the vertices and interconnection will denote edges. So we can have graph kind of a representation like this. The primary inputs are shown like this and these are the flip flops this output this is the flip flop number 2 this is z. And from here again you go to this NAND gate this

is that NAND gate from the NAND gate you come here from here you go to n 1 from here you go to this NAND gate as well as to n 3. So this is your netlist graph now this graph I have constructed from a given circuit this graph you can also construct from a given c d f g okay. So it does not even matter whether you are trying to partition a netlist or the c d f g the net thing is that you have a graph available with you okay. So this is a graph available with you and I want to divide it up into 2 parts.

Well there are algorithms to do that we will be talking about the algorithms. But let me show you a typical cut or a partition. See a typical good partition in this case will be to cut it like this. Why number one thing is that you just look at the sizes of the partitions. In 1 partition there are 3 nodes in other partition there are 4 nodes. So they are approximately equal and also you see number of interconnecting lines that are crossing the partition that is also 2 not much. So the idea behind the partitioning is that once you partition a problem the constituent sub problems will be handled independent independently. So possibly this particular design will be synthesized in terms of say modules like there will be 1 part of it where a b will be the inputs z will be the output. There will be another block the second part c d f g.

These are the 4 inputs and this will go to this n 1 and n 2, n 1 and n 2 will be going to the other block. So the idea of this partitioning is this you can synthesize this 2 independent partitions in terms of 2 independent modules with the number of interconnections between the modules minimized okay. So even in terms of layouts if you design and synthesize this modules independently they may be located a little far apart if the number of interconnecting lines is less the cost of interconnection will also be reduced. So that is why we also say that the number of lines crossing the partition should also be minimized okay. Before going into the actual algorithms which people use for partitioning.

(Refer Slide Time: 11:20)

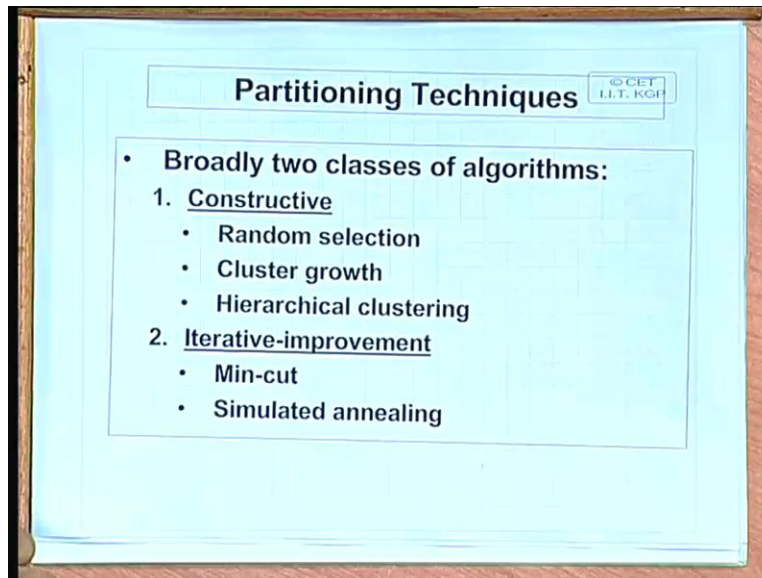


Let us talk about behavioral partitioning when you are using say a high level language say like Verilog or vhdl for description let us look at this. See in Verilog there are 2 ways you can implicitly define the partitions well here the user when he or she is giving the specification can implicitly define the design partitions say by defining modules. Say a Verilog port can be divided up into several modules and each modules becomes a natural partition. Now typically when the user defines the modules. So the design description or the design structure is captured there in itself so the user knows that these are the natural partitions and the number of interconnected lines will be less naturally.

So this is one way from a Verilog description you can capture the partitions and another way is that whenever even inside a module there are several always blocks. Now the always blocks according to the semantic they are supposed to be executing concurrently now since they are executing concurrently they cannot be synthesized independently. They can also be considered to be independent partitions which can be designed and synthesized together. So if there are several concurrent always blocks then each such block can also be regarded as a partition. But these are just some guidelines when you are using a language like Verilog for description okay but it is up

to the cad tool exactly how it will be defining the partitions and manipulating with it. In fact there are many algorithms which have been proposed for partitioning.

(Refer Slide Time: 13:20)



Broadly they can be classified into 2 classes. One the first class is called constructive they are called constructive because you are trying to construct the partitions from the scratch they are slowly growing in size in that sense you are saying that they are constructive. Each partition starts with very small it slowly grows in contrast iterative improvement methods they start with some partition and it tries to improve upon it. So it is not that the clusters are growing in size here the clusters are already existing some initial partition you start with and from there you try to improve upon the partition. See you will see that some of the constructive partitioning algorithm are very simple and that means you can you can immediately understand they will not give very good result. But they are used as the initial partition for iterative improvement schemes okay. So let us look at this schemes one by one.

(Refer Slide Time: 14:32)

The slide is titled "Random Selection" and includes a copyright notice "© CET I.I.T. KGP". It contains the following text:

- 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.

To the right of the text is a small graph diagram with five nodes and several edges. Above the graph, the text "n=100" is written in red.

Starting with the random selection it is the simplest possible scheme. This method says that you randomly select nodes one at a time and place them into clusters of fixed size until the proper size is reached. Say you have 100 nodes in your graph and the designer says that I need 4 partitions. So you randomly pick up any 25 of this nodes place them into the first 1 randomly pick up another 25 place them in to second. Another 25 into the third another 25 into the fourth now since we are picking them randomly obviously the partitioning will not be a good one. And you will typically find there are large numbers of lines crossing the partitions okay. So it will not be a very good partition.

But it will be a partition where the sizes of the partition are equal that is the only thing you are you are ensuring so you repeat this until till all the nodes are being allocated. Obviously if it is very fast easy to implement obviously it produces poor results this goes loud saying as I mentioned this method is still used to generate the initial partitions for iterative placement algorithms. Because the iterative placement algorithms must have some partitions to start with, so you can just randomly generate a partition and from there you can start okay. This random selection goes about creating the partition absolutely blindly without looking at the structure of the graph. But the next one, the so called the cluster growth.



(Refer Slide Time: 16:23)

© CET  
I.I.T. KGP

### Cluster Growth

$m$  : 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 U {t};  
    V = V - {t};  
  }  
}
```

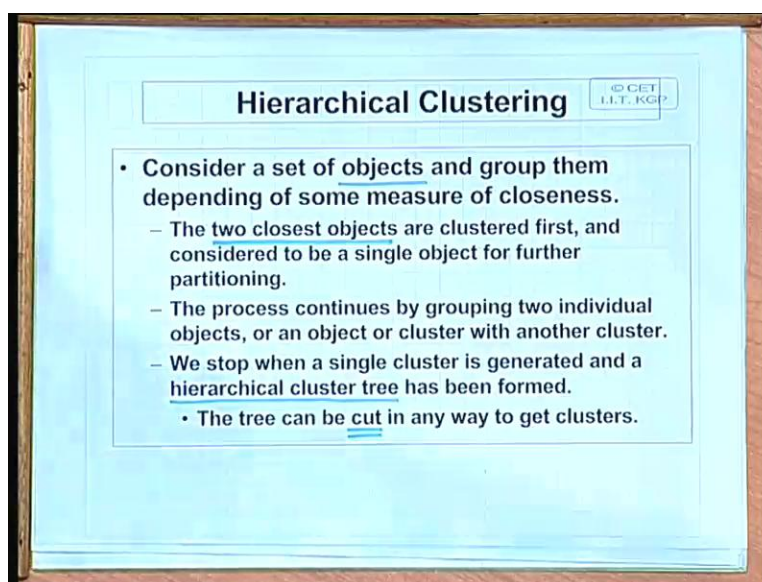
This has been explained in an algorithmic form this is one algorithm which does not try to go about creating the partitions blindly. But it has some intuitive justification what it is doing so here let us try to understand what the algorithm is. Suppose  $v$  denotes the set of nodes in the graph and  $m$  is the desired size of each cluster. So the number of cluster will be  $n$  equal to cardinality of  $v$  divide by  $m$  this will be the number of clusters. So in this outermost loop for  $I$  equal to 1 to  $N$  well we are creating 1 cluster at a time first say  $I$  equal to 1 we are going to create the first cluster this is for  $I$  equal to 1. Now what do we do we start with the vertex with the maximum degree the vertex which is most strongly connected to the other nodes we start with that. See we do this because the vertex which has the maximum degree that possibly will create the maximum problem in terms of the number of lines crossing.

Degree means node and how many lines are either falling into it or coming out of it inputs and outputs totally. So intuitively what we are trying to do is that we pick up the maximum degree node first and try to put in nodes 1 by 1 into it which are most strongly connected to it. You see this is our initial seed is the vertex in  $v$  with maximum degree  $v_i$  this is the partition we are calling  $v_i$ ,  $v_i$  initially contains only the seat only this one node. And the original vertex set we are removing seed from it because we have already placed it. And in an inner loop the size of each

cluster is  $m$  we have to put in  $j$  equal to 1 to  $m$  minus 1 we have to put in  $m$  minus one more nodes. So in this loop we select a vertex which is maximally connected to the nodes already present in  $v_i$ . So initially  $v_i$  contains only 1 so let us select 1 node which is connected to it by 2 arcs say that will be  $t$  you add  $t$  to this and remove  $t$  from the original set.

Next time when you select another node you select 1 which is maximally connected to the nodes which are already placed in this way we are trying to put in nodes which are most strongly connected to the nodes already placed there. So this are the intuitive justification this strongly connected ones will go into 1 cluster total number of lines you count you count the total number of lines. When I am selecting this so I am counting how many lines are going to this node how many lines are going to this node some total coming or going out number of lines that are getting connected. So this is obviously much better than random placement because it has some intuitive justification. But the problem is that it is a greedy algorithm. I am always trying to move towards the best possible alternative at any given step so globally it may not give the best possible solution but it will give a reasonably good solution. This again is used as the initial placement for alternative placement series fine. And another scheme which is used for this constructive placement is called hierarchical clustering.

(Refer Slide Time: 20:30)



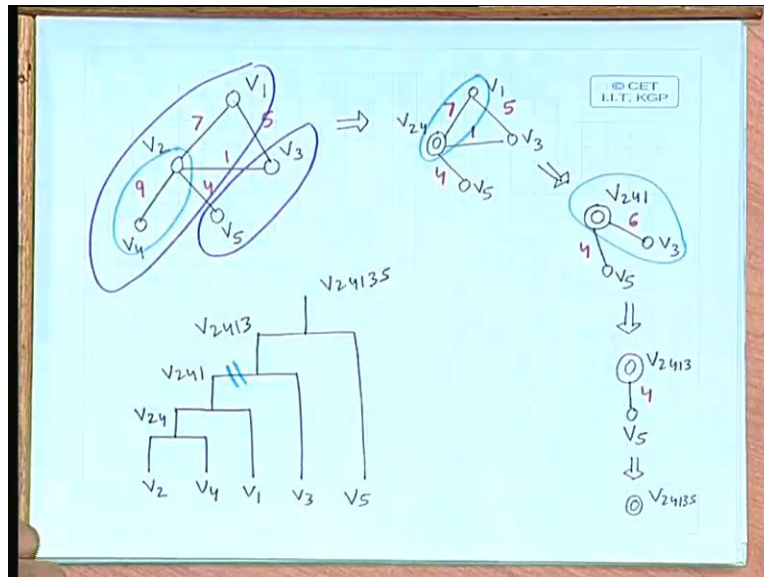
**Hierarchical Clustering** © CET  
I.I.T. KGP

- Consider a set of objects and group them depending of 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.

This method is also interesting see here what we say is that we consider a set of nodes or objects whatever we call and group them depending on some measure closeness. So well in the earlier method we were already doing that in the method of cluster growth we were already doing the same thing. We were grouping them using some measure of closeness but the way we are doing this is different. See in the earlier case we were constructing 1 cluster at a time but here we would be hierarchically constructing some kind of a cluster tree that we will see a little I will just take an example. The idea is that you identify the 2 closest objects or the nodes in the graph if the 2 nodes which are most strongly connected among themselves you put them into 1 cluster. And you consider the pair to be a single object or a single bigger cluster you can say for future processing and you repeat this process.

Now the next iteration that bigger cluster comprising of 2 nodes will be treated as 1 and again you see which are the 2 nodes, which are nodes which are strongly connected. And in this way you go on constructing in the form of hierarchy step by step you do this and we stop when everything has been taken together and we have a single big cluster. And we will see through an example that we have or means we are constructing a hierarchical cluster tree in the process now once you have the tree we can cut the tree in any way we want to create any number of clusters we want. We will illustrate this with an example that means what do we mean by the cluster tree and how we can cut that tree. Okay. Let us take a simple example.

(Refer Slide Time: 16:23)



Suppose I have a graph like this,  $v_1$ ,  $v_2$ ,  $v_3$ ,  $v_4$  and  $v_5$  these are the nodes between 2 nodes there can be several interconnecting lines. Because the nodes may not be as simple as gates it can be bigger modules also so let's also give some weights to the edges which will indicate the number of edges crossing 7, 5, 1, 4 and 9 say. So once you do this now you try to find out which is the pair of nodes which are most strongly connected. In this graph we can obviously see  $v_2$  and  $v_4$  their weight of edge is 9. Okay, so in the first step what you do you transform this graph into 1 which looks like this  $v_1$ ,  $v_3$  this node will now become a macro node. I am showing it by double circle this will be connected to  $v_5$ , this node I am calling  $v_{24}$  this is a combination of 2 and 4. So this 2 and 4 we have combined okay into 1 this numbers denote the number of wires connecting nodes  $v_1$  and  $v_2$  number of wires connecting  $v_1$  and  $v_2$ .

So now again you come you just put down the weights of this edges say for example when you put down the edge from  $v_1$  to  $v_{24}$  it will be the sum of the weight  $v_1$  to  $v_2$  and  $v_1$  to  $v_4$ . But  $v_1$  to  $v_4$  there were no edges so it will still remain 7 okay this will still remain 7 this will be 5 this will again remain 1 because there is no edge from  $v_3$  to  $v_4$ . This will also remain 1 this will also remain 4 okay so in the next step you again look at the smaller graph 7 is the next largest weight so next step you get a graph like this. You combine  $v_1$  and  $v_{24}$  so you will be getting 1

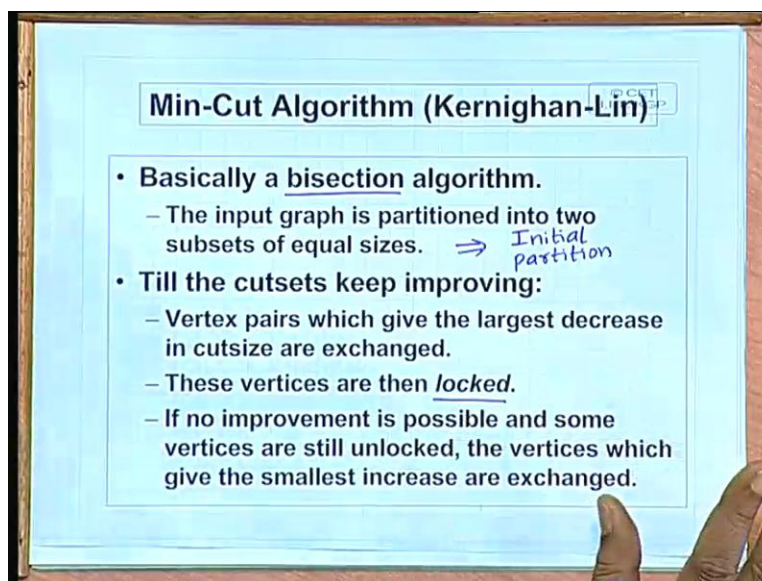
node which is  $v_2 v_4 v_1 v_2 v_4$  you are adding 1 to it. So from  $v_2 v_4 v_1$  you have 1 node which connects to  $v_3$  another node which connects to  $v_5$  okay. So here we have combined  $v_1$  and  $v_2 v_4$  now here when you try to compute the weight  $v_3 v_3$  at a weight 5 with  $v_1$  and 1 with  $v_2 v_4$  so it will be 6. But  $v_5$  only 1 edges was there it will remain 4 so from here the next step will be you will be combining  $v_3$  and  $v_2 v_4 v_1$ .

So there will be a node  $v_2 v_4 v_1 v_3$  and a node  $v_5$  which will be joint together and the weight will be 4. And in the last step even these 2 will be combined together and you will be getting a single node  $v_2 v_4 v_1 v_3 v_5$ . So this is how you are combining the nodes and you are ultimately leading up into a single big node. But if you represent the way you are combining with a tree okay I am drawing the tree. Now you see how the trees can initially we had combined  $v_2$  and  $v_4$ . So this I am representing like this  $v_2$  and  $v_4$  were combined to get a node  $v_2 v_4$  next step  $v_2 v_4$  and  $v_1$  were combined so  $v_2 v_4$  and  $v_1$  were combined to get a node  $v_2 v_4 v_1$ . Next step  $v_2 v_4 v_1$  and  $v_3$  were combined to get a node  $v_2 v_4 v_1 v_3$ ,  $v_2 v_4 v_1 v_3$  and  $v_5$  was combined so finally  $v_2 v_4 v_1 v_3 v_5$ . See you look at this tree now this is the hierarchical clustering tree. This tree was constructed taking into account this strongly connected nodes in that order you had clustered.

So suppose our objective is to divide this graph into 2 parts then possibly we will try to cut it say where say at this level let  $v_3$  and  $v_5$  remain in 1 cluster if you want  $v_4$ ,  $v_2$  in another cluster. So if we cut the tree see the edges in the tree indicate that the cluster on the left and the cluster on the right are not that strongly connected because strongly connected ones will be grouped together on the left side. So as you go up the tree the weights of the edges of this tree will become less and less and less so if you cut here for an example then 1 2 and 4 will come into 1 partition. Means you will get 1 partition like this and 3 and 5 will be in another partition. So number of lines crossing will be 3. Yes. [Student Noise Time: 29:09] Oh, yeah. Just wait sorry 5 and 4, 9 and 10. Yeah right this does not insure globally optima this is again a greedy approach you can say every time we are getting the best and trying to construct. So this again I am repeating this kind of constructive algorithm these are used mainly for generating initial good partitions.

But through iterative improvement scheme we can improve upon these solutions. So now we shall look at the iterative improvement schemes. In fact we will see that in many of the cad algorithms this iterative improvement schemes work much better than any you can say deterministic algorithm. So would be looking at few of those methods I will be talking about so I am repeating these constructive methods are typically used to generate an initial good partition okay. But after the initial good partition is created you typically go for some of the you can say iterative improvement scheme. Now one of the simpler iterative improvement scheme yes [Student Noise Time: 30:25] in this one. So your question is that here we have always 2 partition it not it is not necessary too. Suppose we have a very big tree okay we have a very big tree but the number of nodes may be 100. So we can choose to divide even to 4 also I can cut it here I can cut it here and cut it here. So since this is a tree whenever I cut 1 node it will divide the tree up into 2 sub trees okay fine.

(Refer Slide Time: 31:05)



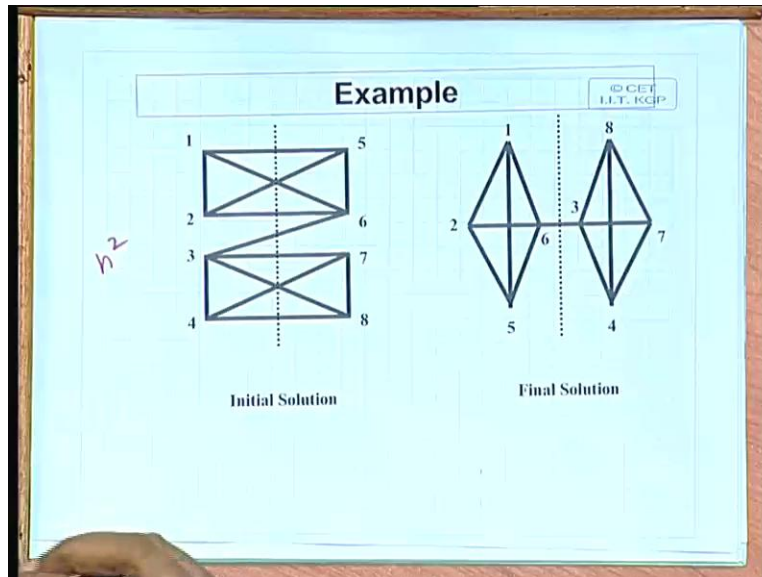
So we will look at the first iterative improvement scheme. This is a very well known and classical scheme this method is called Kernighan lin algorithm. Now one characteristic of Kernighan lin algorithm is that this is a bisection algorithm it divides a graph into 2 equal parts 2 equal parts this is the only thing it can do. Say input graph is partitioned into 2 subsets of equal

size but 1 thing you will have to supply with a with an initial partition. And this algorithm will be improving upon that okay initial partition can be generated randomly also or using any of the other methods I discussed. The algorithm is conceptually very simple well cutsets what is a cutsets in a graph while any line which divides the graph into 2 parts is called a cut set and the number of edges that is crossing a cutsets is called the cost of that cut okay.

So our objective is to is to get a cut of minimum cost minimum of edges crossing so the algorithm goes like this till the cut sets keeps improving vertex pairs which give the largest decrease in cut size are exchanged you start with 2 sets. Pick one from this, one from this exchange them you pick those 2 nodes whose exchange will give you the maximum gain okay. And once you have exchanged them you lock this vertices so that they are not exchanged again in the future but this is not always following this greedy path. Sometimes you will see that all vertices are not locked yet but still by exchanging you are not getting any improvement then what you do if no improvement is possible and some vertices are still unlocked the vertices which gives the smallest increase in cost are exchanged.

So you are deliberately going towards the worst solution with the expectation that after that you may be you will get 2 vertices to be exchanged which will give you a better say gain. So you are trying to come out of the local minimum you trying to get into better solution so this is the crux of the Kernighan lin algorithm. [Student Noise Time: 33:52] Kernighan lin algorithm was designed for 2 equal partitions there are some other variations of this where you can move 1 node from a cluster to another instead of exchanging it that version is also there some other algorithm is there. Yes there is an algorithm called Fiduccia-Mattheyses algorithm that does exactly this you select 1 node from 1 cluster move it to the other cluster so that you can have clusters of unequal sizes also. Now so let us try to illustrate this with an example a simple example.

(Refer Slide Time: 34:32)



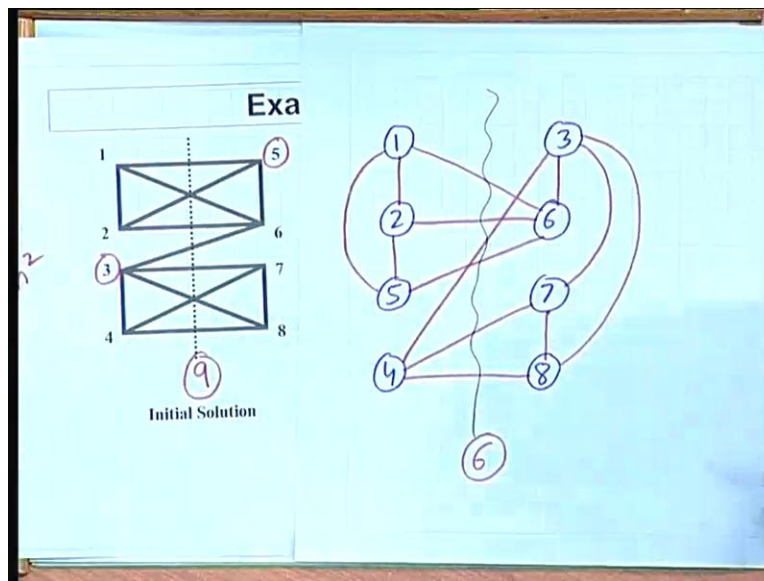
Let us take a graph like this. There are 8 nodes these are the edges and as you can see we have started with a very bad partition. 1, 2, 3, 4, 5, 6, 7, 8, 9. 9 edges are crossing the partition the cut well I have shown the final solution side by side final solution you can have a single edge crossing the cut. But let us see the steps how we have arrived at this well first from this initial graph you will have to check every pair of nodes 1 5, 1 6, 1 7, 1 8, 2 5, 2 6, 2 7, 2 8, 3 5, 3 6, 3 7, 3 8, 4 5, 4 6, 4 7, 4 8, that which pair if I exchange them gives me the maximum benefit. So this step itself has a complexity of  $n^2$  if  $n$  is the number of nodes because we have  $n$  by 2 nodes in on partition  $n$  by 2 in another partition. So the time taken to do this will be  $n$  by 2 whole square so it is of the order of  $n^2$  so I am not showing you the detail that means what is the cost gain of the pair.

I am showing you the best 1 which can be found you can see that if you exchange 3 and 5 you will get the maximum benefit okay this you can do by trial. Benefit means you exchange them as a trail and see that what are the number of edges now look now which cross the partition. Now if we exchange 3 and 5 for instance the graph will look like this I am showing it side by side if we exchange 3 and 5. So now the first partition will contain the vertices 1, 2, 5 and 4. 1, 2, 5 and 4 and the second cluster will contain 3, 6, 7 and 8, 3, 6, 7 and 8. So now let us draw the edges. Say



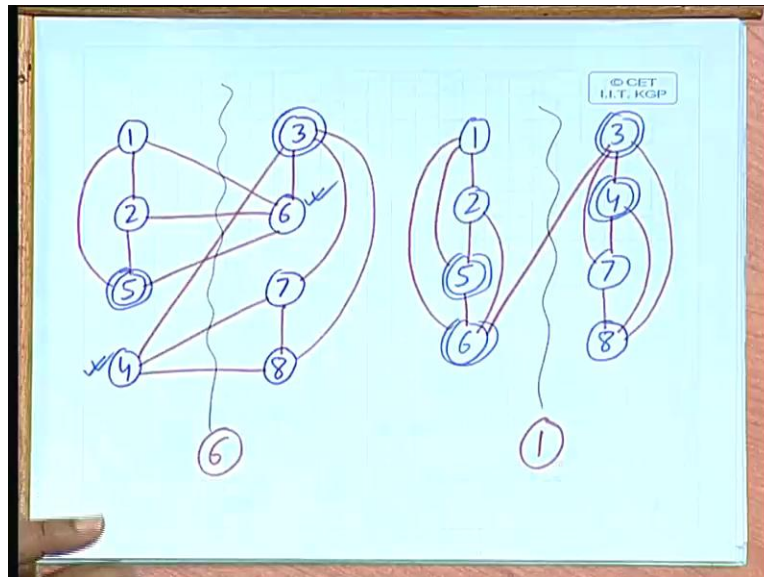
from 1 there is an edge to 5 6 and 2 5 is in the same cluster now 6 and 2 from 2 1 already I have added 5 and 6 5 is here now 6 is here. From 3 6 7 4 6 7 is now here and 4 from 4 3 is already included 7 and 8 7 and 8 from 5 these I have included 6. 6 3 is included from 7 to 8 no 3 8 I have not included and 3 to 8 there is 1 7 to 8 is 1 that's all.

(Refer Slide Time: 37:57)



So now if you estimate the cost of the cut line was like this now the cost of cuts initially it was 9 the cost was 9. But now it is 1 2 3 4 5 6 the cut size is 6 so it has gone down from 9 to 6.

(Refer Slide Time: 38:20)

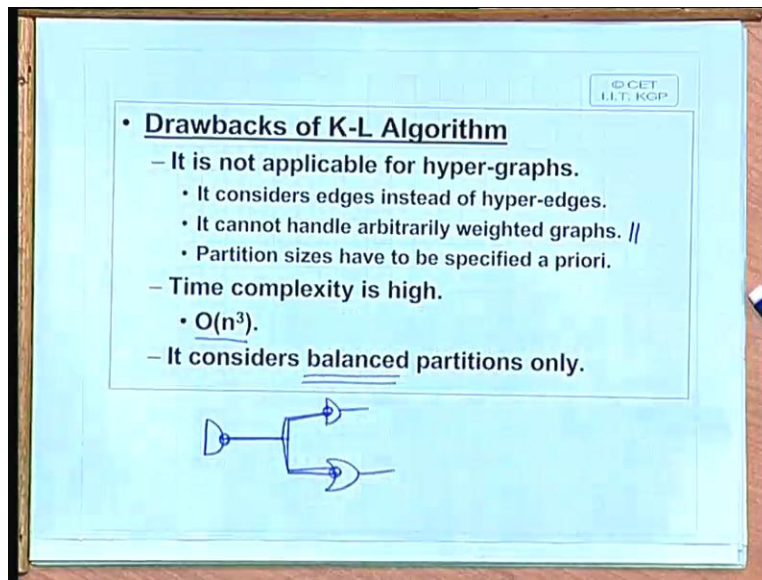


Now you repeat the same step from here onwards say in this graph you again check that which is the pair of nodes if we exchange them will give you the maximum benefit. Well here again since you have exchanged 5 and 3 these are locked these are not eligible to be exchanged any more this 2 have been locked so you check the other nodes which pair will give you the maximum benefit. Well here in fact 6 and 4 will give the maximum benefit you will be exchanging 4 and 6. So let us see the modified 1 one 2 5 6 and this side 3, 4, 7 and 8. So again you try to draw the 1 okay 1 2 two 5 1 5 1 6 six is also. Now here from 2 two 6 six is now here from 5 five 6 six is also here from 4 3, 4 7, 4 8, 4 7, 4 8, 3 6, 3 7, 3 8, 3 7, 3 8, 7 8 I think this is all. So now you see you will get a cut whose cost is 1 in fact in this algorithm you do not get any further improvement see if you recall we had said that if all vertices are not locked for example here 3 and 5 were locked.

But now we have locked 4 and 6 also so 1 2 7 8 are not locked so we can exchange with some increase in cost but we remember always that what is the best solution we have seen. So far this was the best solution we have seen so far we will see that even if we exchange 1 7 2 8 and something you will not get anything better than 1 so this will be the final solution which will be reported. So this same solution if you draw in a different way you get this. Now when you do the partition it is not necessary for the nodes to be interconnected know. My main objective is

number of the lines crossing the partition should be less that's all but these nodes may be disjoint also no partition okay. This method is simple it gives good results for many graphs. But it suffers from some drawbacks also.

(Refer Slide Time: 41:27)



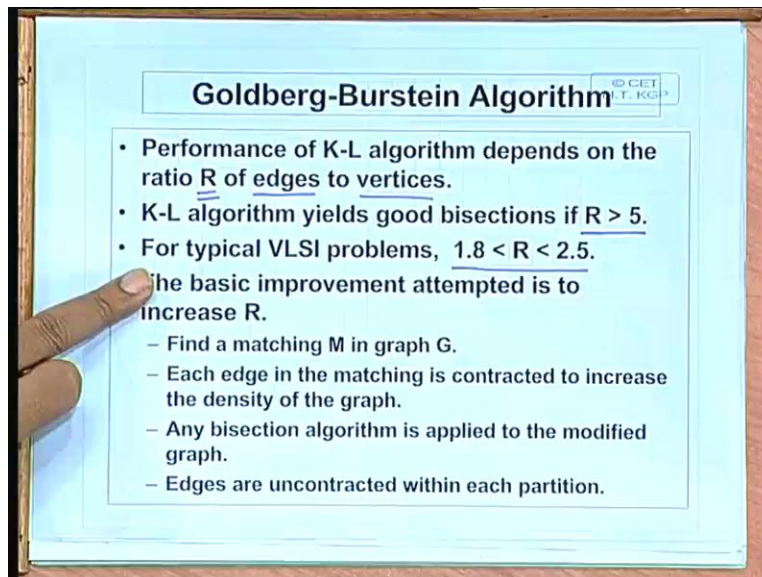
See it cannot be directly applicable to so called hyper graphs. Well what is a hyper graph see in a typical netlist the output of a gate can go to the input of more than 1 gate. So you have an interconnection line which joins 3 nodes this we can call a hyper edge an edge connecting more than 2 nodes. Now in a netlist such hyper edges are quite common ofcourse you can break a hyper edge into a number of 2 terminal edges but that is a way of simplifying a problem. But an algorithm which can directly consider hyper edge may possibly be more efficient okay. So Kernighan lin algorithm cannot take care of hyper edges it takes only edges connecting 2 nodes at a time. And it cannot also handle arbitrarily weighted graph the way it select the nodes it becomes difficult of course could with some modifications you can do this.

This is not impossible to do you can do this and another thing is partition sizes have to be specified beforehand. The time complexity is high because in each step as a result complexity was  $n^2$  and the number of steps is  $n$  if it goes around  $n^3$  and the partition sizes are

always equal. You cannot have unequal partitions right. [Student Noise Time: 43:10]. Now here if you want to apply Kernighan algorithm to generate more than 2 partitions it will have to be a number which is power of 2 whether 2 partitions or 4 partitions or 8 partitions like that. [Student Noise Time: 43:30] Yes yes when you are exchanging 2 nodes as I said we will have to check an node of this partition with all the nodes of the other partition. Its and then we have to find out which is the best. Yes, yes, yes. [Student Noise Time: 43:45] Yes. You are saying that in this example in the second step we have exchanged 4 and 6 now what is the question. First step we have we have exchanged 3 and 5 [Student Noise Time: 44:05] lock to the everyone else you are not exchanging them any more yes. [Student Noise Time: 44:18] weighted graph it can handle with some modification the way.

The way you compute the cost if you also incorporate the weight in the calculation you can handle as I said this is not a very strong restriction you can do it. See it is not handling hyper edges like that it is it is handling see the graph which is give as the input to the kl algorithm their all edges are connecting 2 nodes only. There is no edge which is connecting 3 nodes like you have in an netlist graph suppose the output of a gate goes to 3 different gates you break this up into 3 different edges. But it would have been hid away much better if we have treated the collection of those 3 edges as a single edge a single edge connecting 4 vertices [Student Noise Time: 45:16] better optimization yes. But as you will see that the that most of algorithms do break a hyper edge into 2 terminal edge because of the ease of manipulation. [Student Noise Time: 45:34] Yes, yes and Kernighan lin algorithm works pretty well. But there is an interesting you can say improvement to the kl algorithm which was suggested let me talk about this.

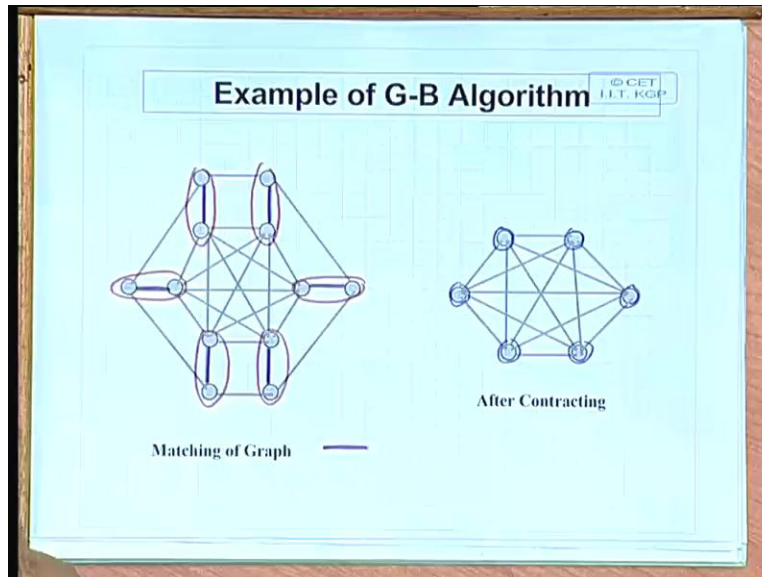
(Refer Slide Time: 45:49)



This improvement is called Goldberg Burstein algorithm. Well this is not a new algorithm this is a way to make the kl algorithm run better. See statistically it has been found that the Kernighan lin algorithm the way it works the performance of it it depends on the density of the graph that how dense are the edges  $r$  is defined as the ratio of the edges to the vertices in the graph number of edges divide by the number of vertices. It has been found that Kernighan lin algorithm gives very good results if  $r$  is greater than 5. But if the graph is a sparse graph not many edges then the quality of the result is not that good.

And it is also been found that for a typical VLSI problem the value of  $r$  is much less than 5. No the idea behind Goldberg Burstein is that some how if we can reduce the value of  $r$  and then apply Kernighan lin algorithm then it will work better than what it was doing earlier. So the idea we will illustrate with an example this is basically a graph theoretic transformation given a graph you try to find a matching and try to contract the edges in the matching by doing that you will be reducing the value of  $r$  in the resulting graph well I am just trying to explain this example it will be easier for you to understand.

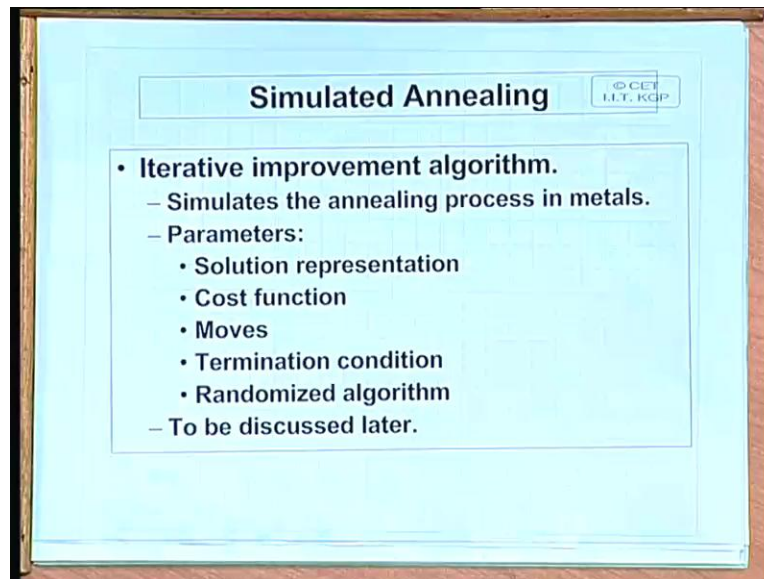
(Refer Slide Time: 47:33)



I have taken I am taking a simple example suppose this is a given graph these are the edges in this graph you try to find out a matching. Matching is a set of edges which connects only 2 pair of nodes and the matching set which you take there is no overlap between them that means you cannot define this to be a matching and also this to be another matching edge. The vertices should all be disjoint 2 edges in a matching must be having disjoint set of  $n$  vertices so for this graph this is 1 possible matching shown in pink. So what the Goldberg Burstein algorithm says is that you first identify a matching in a graph bigger the graph more number of edges like this you can identify and then you contract this edges by merging this 2 into 1 wherever possible.

So you now get a graph which looks like this so actually these vertices are contracted vertices contracted vertices where in each of them represent actually 2 vertices. So essentially each of these represent a pair and what you have gained in this is that in this new graph the value of  $r$  is less as compared to this. Matching edges is that matching is a set of edges whose end vertices are all disjoint so this is just an improvement of  $kl$  just after you do the contraction we apply  $kl$  algorithm for partitioning. Then after partitioning you again expand these macro vertices. Okay.

(Refer Slide Time: 49:40)



But the most widely used and most popularly used you can say iterating improvement algorithm is something called simulated annealing. Well simulated annealing the details of this algorithm we would be discussing a little later when we talk about the physical design automation steps like placement routing etcetera. Now you will see that the simulated annealing algorithm and you can say its sister there is another algorithm called genetic algorithm these 2 algorithms are very widely used in VLSI design where the search space is very huge. And you can understand by the different steps involved in this algorithm that this is not a deterministic algorithm at all it tries to simulate the annealing process in metals.

Normally when you try to form the metals you raise it to high temperature then slowly cool it down that is a cooling sequence just following a cooling curve you have to cool only then it will crystallize in a proper way. Now this algorithm involves a number of things you have to have a way to represent a solution you have to have a way to evaluate the solutions which is more costly than the other. And you define some moves where you can modify your solution to get a new solution there are some termination conditions so the idea is like this you start with an initial solution you apply a move. And if the move leads to the best solution you accept it if the move

leads to a worst solution you accept it with a probability and value of the probability becomes less and less and less as the number of iterations goes on.

So initially you can accept a worst solution with a higher probability but as the number of iteration goes on that probability values goes lower and lower this is basic idea behind the method. And this is found that for problems which are big where the search space is huge no deterministic algorithm can handle it in a reasonable way. These methods give excellent solutions in fact many algorithms in VLSI cad are implemented using simulated annealing or genetic algorithm. So the all these things we will be discussing later when you go into the VLSI back end design algorithms we will be talking about these things. Thank you.