Electronic Design Automation

Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No #16
Backend Design: Part II

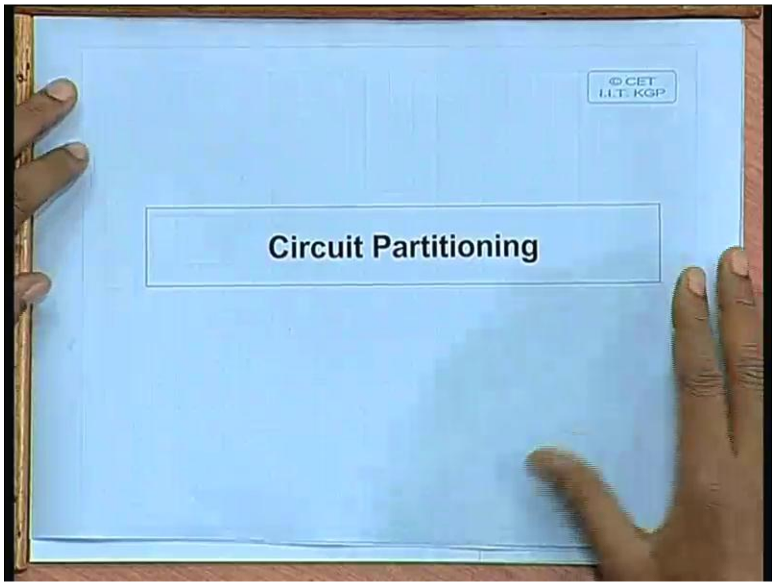
(Refer Slide Time: 01:00)

Lecture #16

Backend Design : Part II

In the last class we had started our discussion on backend design process in the entire cycle of VLSI design. Now today we would be starting by looking at one of the first steps that is involved in the backend design phase.

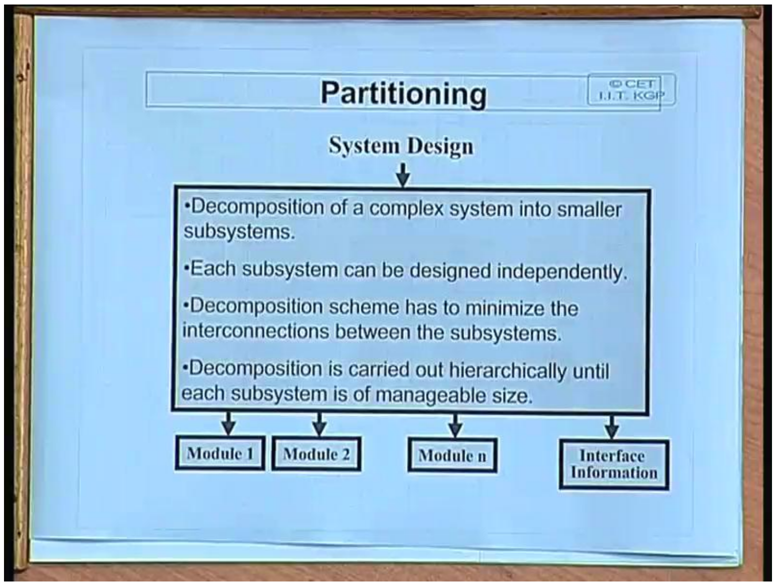
(Refer Slide Time: 01:28)


© CET I.I.T. KGP

Circuit Partitioning

And that is circuit or system partitioning. So basically partitioning is one of the first stages that we need to carry out when we go about doing a physical layout or physical design of the system we want to map into silicon or any other target concept. So first let us try to understand what are the different goals of partitioning? What does this partitioning problem involve?

(Refer Slide Time: 02:04)


© CET I.I.T. KGP

Partitioning

System Design

- •Decomposition of a complex system into smaller subsystems.
- •Each subsystem can be designed independently.
- •Decomposition scheme has to minimize the interconnections between the subsystems.
- •Decomposition is carried out hierarchically until each subsystem is of manageable size.

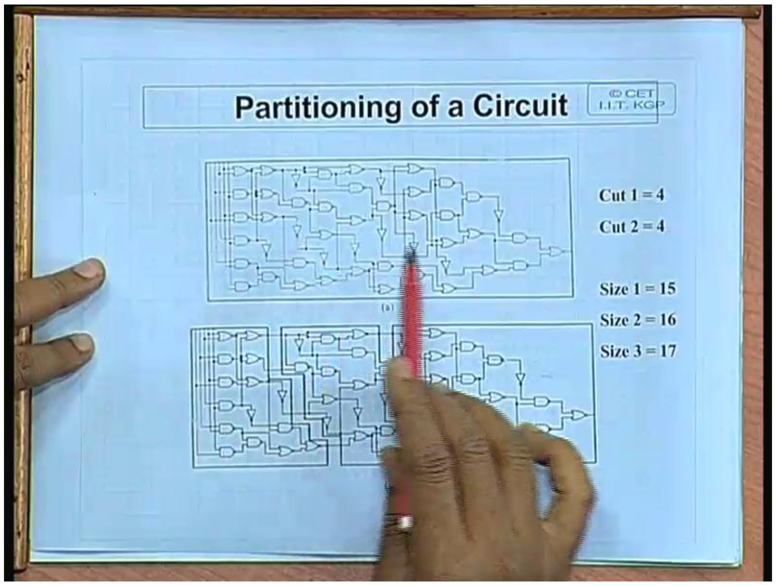
Broadly speaking partitioning can be carried out at any levels. So here whatever I would be discussing today, this is applicable to system level partitioning, circuit level partitioning, even net level partitioning at a very low level. So means where the netlist may comprise of transistors. So, we say that we have a system design this can be a design at any level as an input which is some kind of a net list. So our goal of partitioning is to decompose this particular system which well is supposedly a complex one, a large one into smaller sub systems.

This is the simplest way of specifying the goal of partitioning given a bigger system we have to break it up into smaller sub systems. Now the reason we do this partitioning is that, each such sub system can be processed independently. Well, here we are saying design it depends on the context under which we are doing partitioning. So each such sub system can be handled independently so that we have some kind of parallelism a with respect to the thread of activities so after this partitioning is over. So, one goal of partitioning is also to achieve some degree of parallelism beyond a certain point.

And obviously this partitioning cannot be done in any arbitrary way. There will be some desirable properties that need to be looked at. One is of course to minimize the number of interconnections that cross the sub systems. So we have to design say a bigger system might break it up in a two sub systems. It must be such that the number of lines crossing the partition is minimized. That will be called a good partition. And it depends on exactly what is the size of the target partitions we are looking at. So we may have to do this kind of partitioning hierarchically till each sub system that we get is of manageable size.

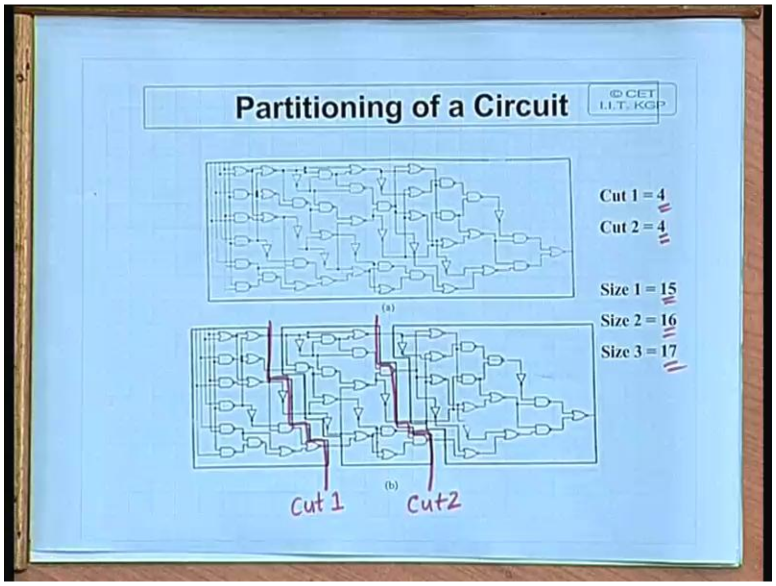
So after this process of partitioning is over what we will be getting? We will be getting a set of smaller modules and some information regarding the interfaces. Interfaces means how these modules are interconnected among themselves so some information regarding that. So after the partitioning process is over we have a set of modules and some information regarding how these modules are interconnected. Fine. So let us take a very simple example. This is a gate level netlist of course the gates are very small I am not sure whether we would be able to see it very clearly.

(Refer Slide Time: 05:21)



This diagram out here shows a gate level net list on top. This gate level netlist we are partitioning like this.

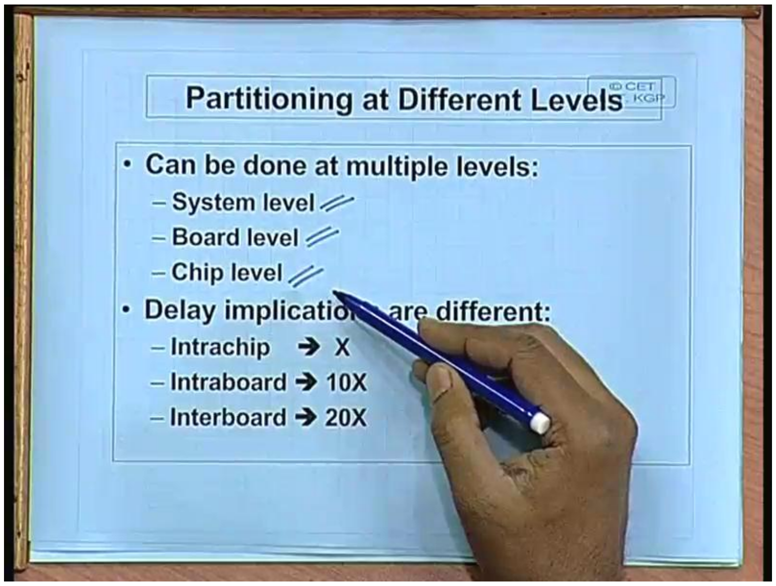
(Refer Slide Time: 05:42)



Say this is one partition this is another partition. Given a netlist like this, suppose we want to divide up into three parts this is one partition we are showing out here. For this partition the number of lines which are crossing the partition, the cut 1, this is cut 1 and this is cut 2, these are 4 each. So there are only four interconnecting lines which are crossing the partitions and moreover the sizes of the partitions are more or less equal. So this will be called a good partition. Right? So this is just an example just as partition as I told you earlier this may be carried out at any level; not necessary in the level of gates. So there the basic building blocks may be higher level blocks as well cells. So before we talk about ways of doing partitioning ways of estimating the cost of a partition say with respect to the size of cut.

There are a few other things you need to understand and consider the idea is that this process of partitioning that we are talking about, this can be done at several different levels like say you can do partitioning at a very high level which we call the level of the system. System level partitioning. Well we can do partitioning at the level of boards the printed circuit boards; board level partitioning. And we can even do partitioning at the level of chips. These are called chip level partitioning. So actually when we do partitioning at the different levels we need to look at the cost involved. Because whether we are doing partitioning at the system level, board level or chip level, the cost of partitioning will vary.

(Refer Slide Time: 07:45)

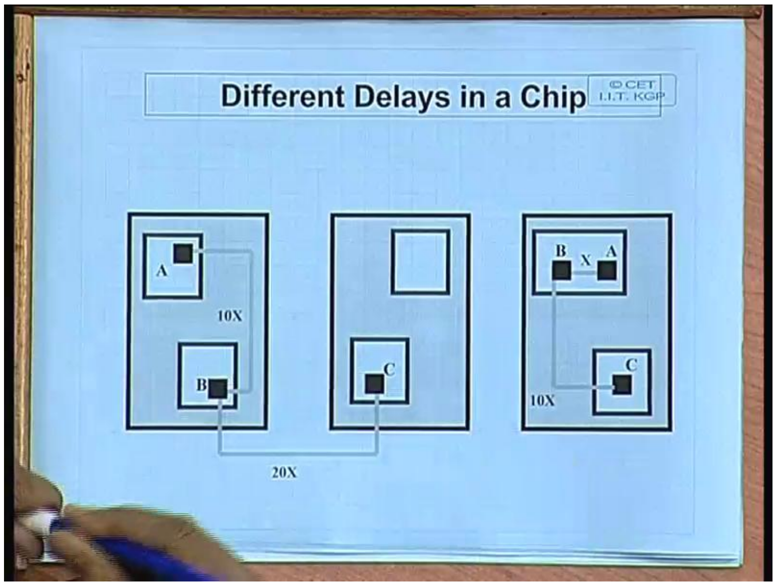

Partitioning at Different Levels

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

This we will be having very rough idea. Suppose we have a chip this is a chip and suppose we have a netlist which is supposed to be mapped within the chip itself. And if we do a partitioning at that level, so we are partitioning this circuit into two sub circuits which will be finally mapped inside the same chip. So the cost of intra chip partitioning let us say some parameter x. But if the partitioning is such that a part of a circuit goes into one chip and the part of the other circuit this some other part goes into another chip and they are interconnected across the chip boundaries. This is the intra board that means within a printed circuit board between chips. Then typically the delays are of the order of 10 times of the basic delay. Similarly when we talk about two different printed circuit boards sitting on the same slot in a system and they are interconnecting among their; they are communicating among themselves.

There the delay can be even more because the signals will have to go through the back plane back into another PCB. So it really depends just not the partitioning. But exactly at which level you are doing partition the cost of partitioning will depend on that. Delay in the circuit and there will be capacitive loads and other loads when you are driving off chip. Off chip delays will be much more due to that. Because the capacitive and resistive load will be much more when you are driving some signal outside the chip just as compared to inside the chip. Yes, [Students noise not audible (Time: 09:33)]. Well intra chip I am showing a diagram. See intra chip means inside the same chip.

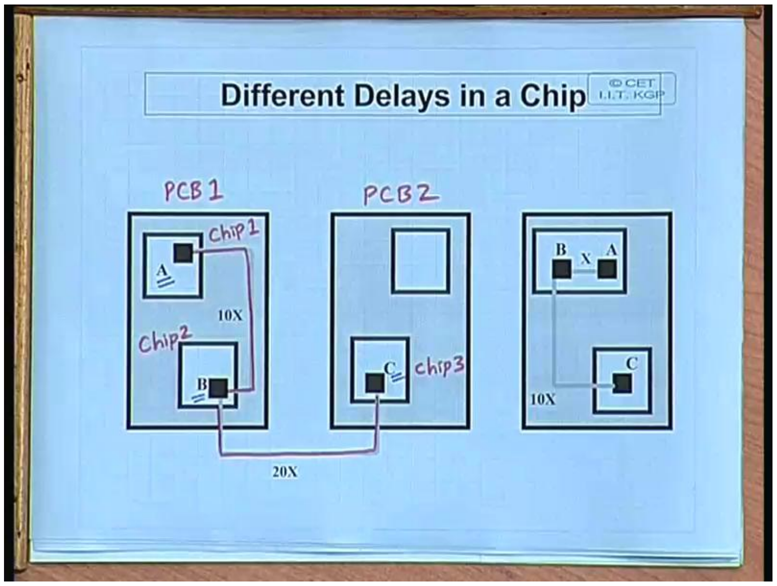
(Refer Slide Time: 09:40)



Different Delays in a Chip

Suppose we had some kind of a design like this where we had three sub systems a, b and c. Well we had done some partitioning with this a, b, c, being the three partitions.

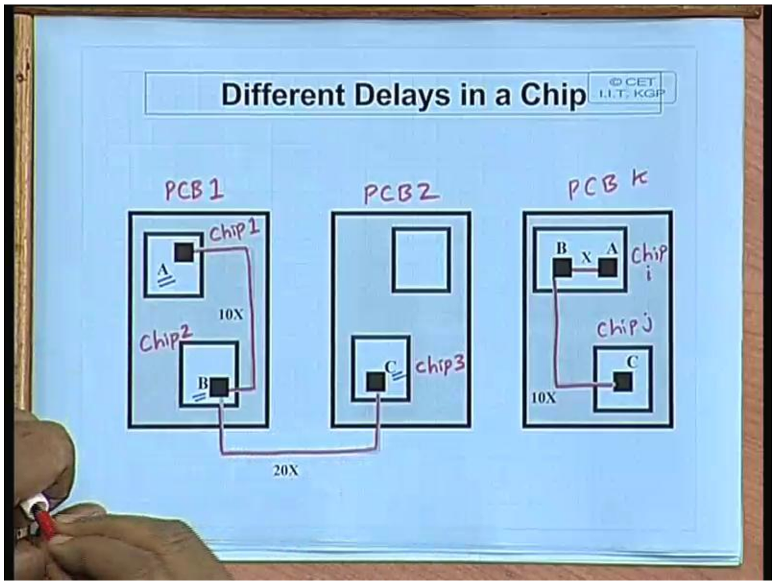
(Refer Slide Time: 09:57)



Different Delays in a Chip

Now these are the interconnecting lines. Suppose in the initial solution, this diagram shows the initial solution. These are the two printed circuit boards. This is PCB1, this is PCB2 and inside PCB's these are the different chips. This is chip1, this is chip2. Now in this PCB, this is some other chip, chip3. So in this solution this, a, b and c, this three sub circuits get mapped into chip1, chip2 and chip3. So if it is a very big system I will have to partition it into smaller sub systems. Some of them can fit into a chip; some of the chips can fit into a board and if it is even very big it has to span across boards. So suppose a scenario like this means occurs.

(Refer Slide Time: 10:55)

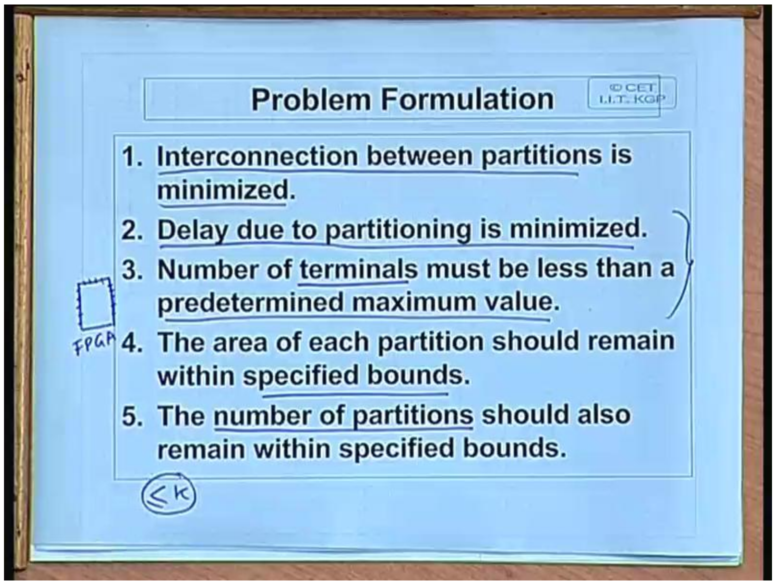


Now as far as the rough guideline about the delay we just showed, so in this case if a has to communicate with b there are some interconnecting lines the cost will be of the order of 10x. Because they are across chips. This is intra board, inter chip. Similarly if it is across boards the delay is of the order of 20x. But suppose as an alternative solution and suppose another way we had partitioned is that both these sub circuits a and b got mapped within the same chip say chip i. And this c got mapped into another chip, chip j. But both these were put with in the same PCB, PCBk. Then as far as our calculation goes this a and b interconnection between them will be having a cost of x only.

Because they would be lying inside the same chip while b to c will have a cost of 10x. See this kind of analysis we have to do particularly for the critical nets. The nets which will be ultimately contributing to the total delay of the circuit. So we will have to try, try and optimize and minimize the delay of those nets by optimally doing the partitioning. Say means we have a big design [Students noise not audible (Time: 12:18)]. Yes, delay of course and also feasibility. Suppose we had say, suppose I am giving an example suppose we have a big netlist, we want to map it finally into FPGA's. Now we know the maximum size of an FPGA.

What is the maximum size of a circuit a particular FPGA chip can fit? So we can say that well this is the total size of the circuit I will require 3 FPGA chips to map it. So now the question is that how I will partition the three circuits so that they can be mapped into 3 different FPGA chips with minimum performance degradation. So till when I am doing partitioning, the issue of performance should also be kept into my mind. So if I map into two different chips I am and if there are some critical nets which are crossing the chip they will be contributing significantly to the total delay. So I would prefer that the critical nets all lie within the same chip as far as possible. Okay? Fine. So with respect to partitioning we can we can now frame the problem slightly formally like this.

(Refer Slide Time: 13:34)



Problem Formulation

© CET I.I.T. KGP

1. Interconnection between partitions is minimized.
2. Delay due to partitioning is minimized.
3. Number of terminals must be less than a predetermined maximum value.
4. The area of each partition should remain within specified bounds.
5. The number of partitions should also remain within specified bounds.

FPGA

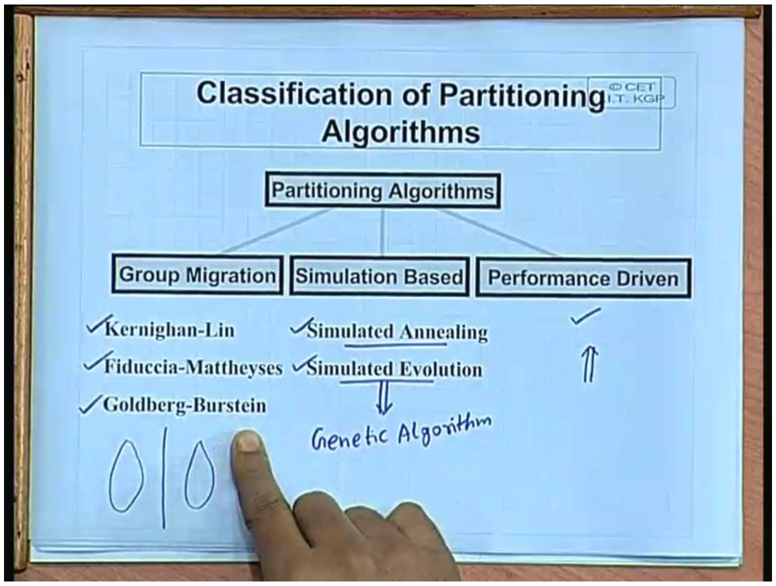
$\leq k$

That we have two partition a big system into sub systems such that, the first thing is that interconnection between the partitions is minimized obviously. Now this minimized this can also involve the costs which I have just mentioned whether the partitions are getting mapped into the same chip or the same board or whatever. So this is captured in the second point, the delay is minimized. Now if you are looking at the same level of hierarchy, delay does not make any sense. But if there are several level of hierarchies like chip, board and system then this delay makes sense. And if your target, suppose again I am giving that example.

If your target is to map a particular partition into say an FPGA chip. This is an FPGA. So a particular FPGA chip will be having a maximum limit to the number of terminals or pins. So the number of terminals of your partition should be less than or equal to that limit. So means, if an FPGA chip has 100 pins, you cannot have a partition with 200 pins and you say that you will have to map into that chip. So the number of terminals must be less than that predetermined maximum value. And this maximum value will depend on your target. If it is FPGA you have a hard limit restriction. If it is an ASIC design it is more flexible.

The area of each partition should remain within a specific bounds. Again for FPGA this is obvious. You have the maximum size of the chip and even for ASIC's there are also some rules of thumbs. You will not want that to one particular module or a cell should become beyond a certain size because of design constraints and limitations other limitations. So normally area bounds are also specified. And the number of partitions may also be one restriction. That means I want to partition it into less than or equal to k partition say. But of course this k you cannot just specify any arbitrary value of k. They will also have to satisfy the other constraints; the size and pin restrictions. Fine.

(Refer Slide Time: 16:09)



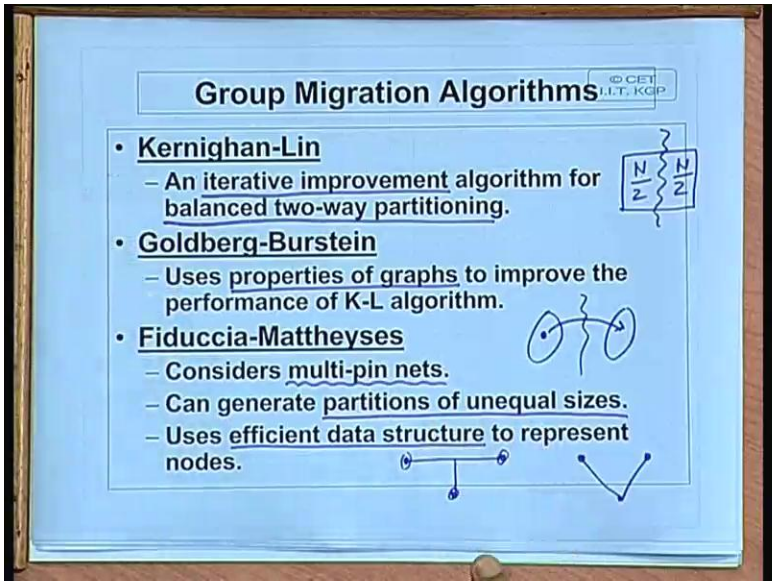
Classification of Partitioning Algorithms

Now broadly speaking the algorithms that have been proposed for partitioning, they can be broadly classified into three categories. Group migration, simulation based and performance driven. Group migration means you already have some groups or partitions defined. Migration means some nodes are moving across the partitions. So some of this algorithms we have already seen Kernighan-Lin, Fiduccia-Mattheyes and Goldberg-Burstein. These are the three methods which fall under this category. Given a netlist you start with an initial partition. Say two, then you try to improve upon this partition. These are all iterative improvement algorithms. There is an iterative loop in that iterative loop you try to go on improve upon the partition. There are some other classes of algorithms which are yes [Students noise not audible (Time: 17:06)] 10 or 12 we have to recursively do it. Because none of these algorithms can very naturally do a k-way partitioning in general.

These are all two way partitioning algorithms. [Students noise not audible (Time: 17:22)] Yes, yes yes. Some of them will be exchanging some of them would be moving a node from one to other. This Fiduccia-Mattheyes algorithm moves a node from one partition to other. Kernighan-Lin, it exchanges two nodes. Well the other category of algorithm, these are simulation based. In fact we will see that the simulation based algorithm works very well in with respect to physical design cad processors or algorithms. See in these kind of algorithm this search space is pretty high, pretty large. And you cannot optimistically try and obtain the best solution. So under these circumstances simulated annealing or simulated evaluation, well a particular form of simulated evaluation is also called genetic algorithm. So genetic algorithm is a kind of simulated evaluation evolution. So these work much better in terms of the quality of the solutions you can get with respect to the group migration algorithms.

And of course there is a third category which is performance driven. Well, there you can specify that my total delay has to remain with in this or some means you will be specifying some performance related constraints that have to be satisfied. Of course these methods are much more difficult to satisfy. Means these constraints are much more difficult to satisfy. They are normally based on certain heuristics which are applied starting with some of the solutions obtained from these. Starting with a good solution you try to make some modification so that the performance restrictions the performance you can say the constraints they are satisfied. But as I told you they are very difficult to implement in practice. Now some of the group migration algorithm we have already discussed. But still let us have a quick look at them.

(Refer Slide Time: 19:37)



Group Migration Algorithms

© CET I.I.T. KGP

- Kernighan-Lin
  - An iterative improvement algorithm for balanced two-way partitioning.
- Goldberg-Burstein
  - Uses properties of graphs to improve the performance of K-L algorithm.
- Fiduccia-Mattheyses
  - Considers multi-pin nets.
  - Can generate partitions of unequal sizes.
  - Uses efficient data structure to represent nodes.

$\frac{N}{2}$
$\frac{N}{2}$

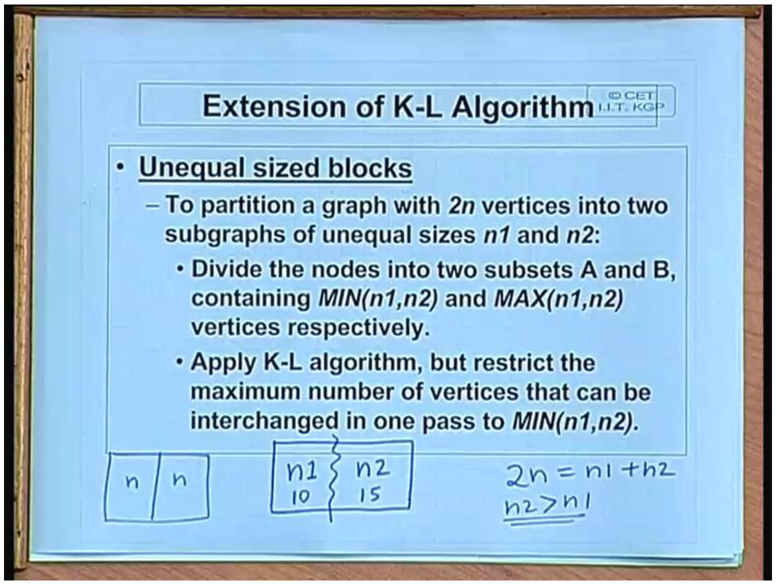
This Kernighan-Lin algorithm we had discussed. Well with regards to our discussion with respect to high level synthesis. This we had said that this is a balanced two way partitioning algorithm; the basic version. Balanced two way partitioning means given a system I want to divide it up into two equal halves. If the total size is n, it will be a n by two and n by two. And it is an iterative improvement algorithm. In a loop we try to identify the pair of nodes whose exchange will give me the best benefit and accordingly I try and exchange such pairs of nodes. Right? So Kernighan-Lin basically starts with a balanced initial partition. And as part of the iteration step it will go on selecting two nodes.

One from this partition one from the other partition and exchanging them and evaluate the cost. The pair which gives the best benefit will be selected. And in this way you go on. Fine. And we had mentioned Goldberg Burstein is not a separate algorithm in its own. Right? It uses some properties of graphs. It modifies the graph into another and after that we apply Kernighan-Lin algorithm. It tries to improve the density of the graph. It is seen that Kernighan-Lin algorithm works well for graphs with high densities. Goldberg Burstein algorithm tries to do that and then apply Kernighan-Lin. Well, this Fiduccia-Mattheyes we are not going into detail, this algorithm is slightly different in the way it works. Well here also you start with two partitions. This can be randomly generated initially.

But now the moves are not like exchanging. One particular node can move from one partition to another. So this can cause or this can lead to partitions of unequal sizes. So in Fiduccia-Mattheyes algorithm we can have partitions of unequal sizes. Moreover we can also keep information about multi pin nets. Multi pin nets means there may be three different pins which have to be interconnected. Normally in the traditional graph we can represent this by either two edges or three edges, whatever it be point to point edges. But we can have so called hyper edges in the generalized graph. So this Fiduccia-Mattheyes algorithm can also take care of this kind of hyper edges. And all these nodes which are connected by a hyper edge it will try to bring into the same partition. Okay?

Means one interesting part of Fiduccia-Mattheyes algorithm is that, it uses a very special and efficient data structures to represent the nodes so that search for the node whose movement or migration will give me the best benefit that gets reduced. We basically do not have to search much. It maintains some kind of a priority list. The node on the top of the list is the one whose movement will give me the best benefit. So it tries to maintain a list like that. Fine. So Kernighan-Lin and its extension Goldberg-Burstein, they give balanced partition; Fiduccia-Mattheyes may give non-balanced partitions unequal. Now we can extend this basic algorithm say for example Kernighan-Lin algorithm is very simple in concept. Now we can extend this Kernighan-Lin algorithm to handle a few other situations.

(Refer Slide Time: 23:37)



Extension of K-L Algorithm

©CET I.I.T. KGP

- **Unequal sized blocks**
  - To partition a graph with *2n* vertices into two subgraphs of unequal sizes *n1* and *n2*:
    - Divide the nodes into two subsets A and B, containing *MIN(n1,n2)* and *MAX(n1,n2)* vertices respectively.
    - Apply K-L algorithm, but restrict the maximum number of vertices that can be interchanged in one pass to *MIN(n1,n2)*.

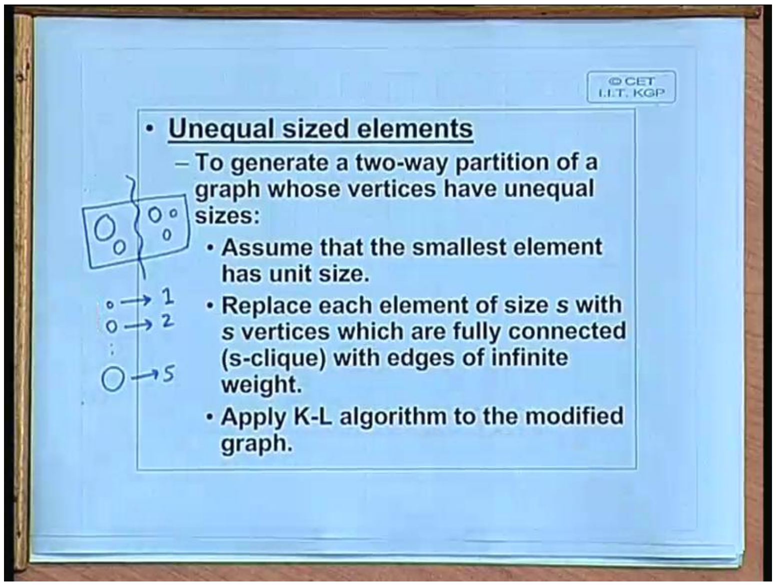
n
n
n1
10
n2
15
2n = n1 + n2
n2 > n1

These are the extensions of the basic Kernighan-Lin. The first extension we look at is that we well; we want to have unequal size blocks. So now what we do? We start with an initial partition. Suppose we have an netlist, we start with an initial partition which are unequal in size. With n1 nodes in the first partition, n2 nodes in the second partition. So we want two partitions of size n1 and n2. So we start with a partition of that size only. So suppose the initial graph had 2n or 2n vertices, we divide up it into n1 and n2. So 2n will be equal to n1 plus n2. Now depending on which is less which is more of course there may be one of the partition will be containing min n1, n2 nodes. Other will be containing max n1, n2 nodes. So if n2 is greater than n1, the first partition will contain n1, second one will contain n2.

Now the way we apply Kernighan-Lin algorithm is the same as we did earlier. But at each iteration we will be exchanging only n one pair of nodes assuming n1 is less. [Students noise not audible (Time: 25:05)] Yeah, because in the earlier case in the traditional algorithm, if there are 2n algorithms, we are dividing around them into n and n and we are iterating till all n pairs of nodes were exchanged. But here we will be stopping with the lower value. So apply kl algorithm, but restrict the maximum number of vertices that can be interchanged in one pass to minimum of these two. If n1 is say 10, if n2 is 15, then you exchange 10 pairs of nodes. [Students noise not audible (Time: 17:22)] That has to be. Yeah The 10 pairs of nodes which gives you the best benefit among these 15. Fine.

So this is one extension you can have to kl algorithm which were handled unequal size blocks. But at the very beginning you will have to apriori mention the sizes. Not that depending on your netlist you will try to optimal optimally find out what is n1? What is n2? Not that. The user will be specifying initially the values of n1and n2. Dynamically you cannot find n1 and n2. Well in many partitioning cases what facts what happens is that, well you start with a graph and we assume that all nodes of the graph are equal. Equal in terms of complexity or cost. But it may so happen that some nodes may indicate a sub circuit or a module which is bigger than that corresponding to another node. So this Kernighan-Lin algorithm can also be extended to that scenario.

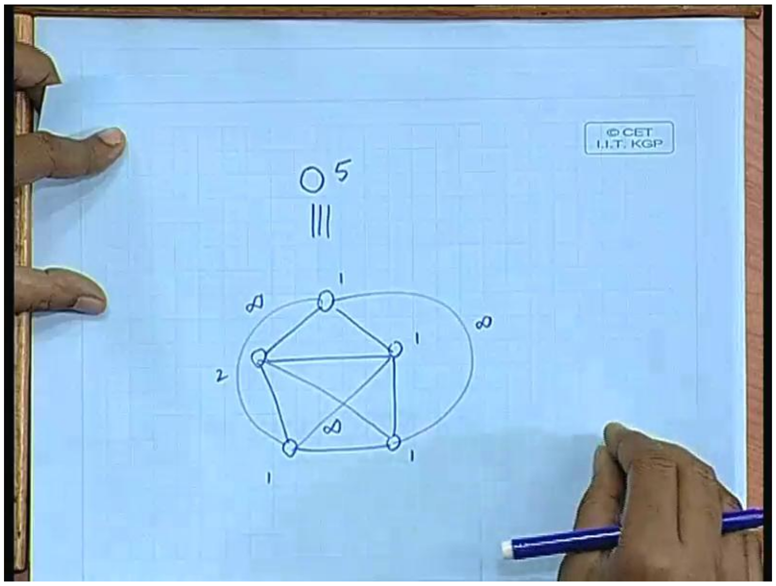
(Refer Slide Time: 26:49)


© CET
I.I.T. KGP
1
2
5

- **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.

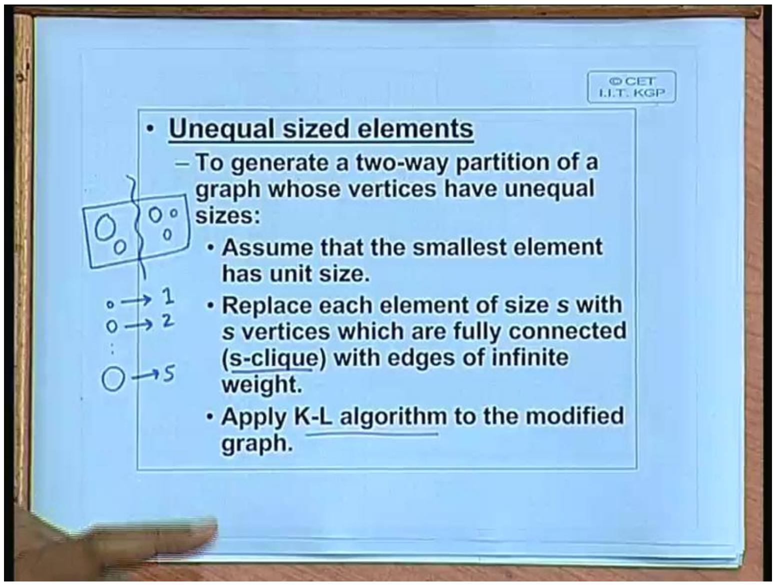
Where the sizes of the elements are unequal. So here what we are what we are looking for is that we still want to partition this circuit into two halves. But now my nodes are not of all same sizes. Some are big, some are small. Right? Now let us see how we can represent this in the graph and then you can apply the conventional Kernighan-Lin algorithm. See looking at the sizes of the nodes it can be the smallest, it can be slightly bigger, it can be bigger. So here what we do is that we assume that the smallest node it has a unit size. Now with respect to the size of the smallest node we can assign sizes to the other nodes. Say this is 2; this is say 5 and so on. So assume that the smallest element has unit size. Replace each element of size s, with s vertices which are fully connected. Like if there is a node of size 5, say.

(Refer Slide Time: 28:02)



You replace this node by a 5 node graph. While each of whose cost are unity small vertices and this graph is fully connected. So you expand this single node into 5 nodes with this edges. Not only that you also set the cost of this arcs to infinity; all these arcs. So that in any reasonable partitioning algorithm they will all fall into the same partition.

(Refer Slide Time: 28:48)

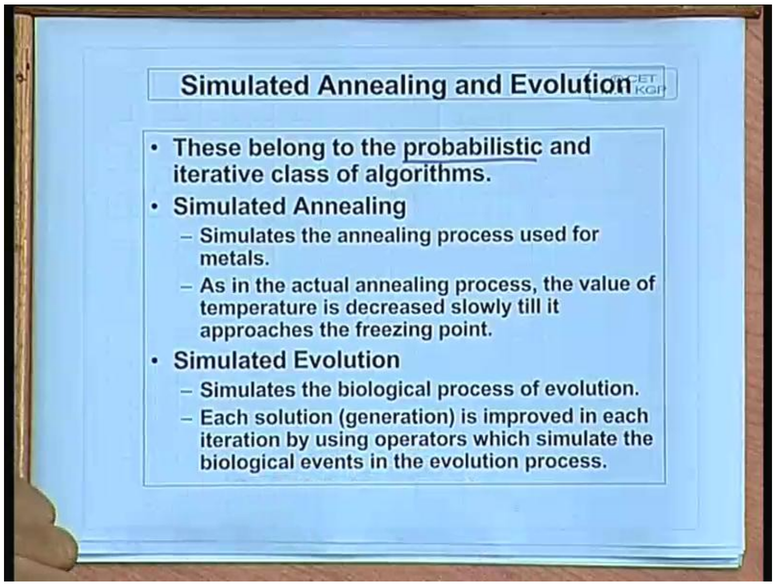


- **Unequal sized elements**
  - To generate a two-way partition of a graph whose vertices have unequal sizes:
    - Assume that the smallest element has unit size.
    - Replace each element of size *s* with *s* vertices which are fully connected (s-clique) with edges of infinite weight.
    - Apply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

© CET
I.I.T. KGP
→ 1
→ 2
→ 5

So replace. Yeah. [Students noise not audible (Time: 28:50)] No. It is non-integer multiple. So the way we are doing it well if it is non integer multiple even you said that base to means half or 0.5 or something like that. Just in terms of that you can do. So you expand it into a fully connected you just call that s-clique and with edges of infinite weight and then we apply kl algorithm. So these are some indirect ways in which apply the Kernighan-Lin algorithm to get a solution for these other cases where unequal size partitions or unequal size elements are there. [Students noise not audible (Time: 29:30)] It can be a combination of these two also. Because before applying the Kernighan-Lin algorithm we will have to start with a partition a partition can be equal it can be unequal. Yes, true. So now let us look at the algorithms which are more widely used because of the quality of solutions they give.

(Refer Slide Time: 29:59)



Simulated Annealing and Evolution

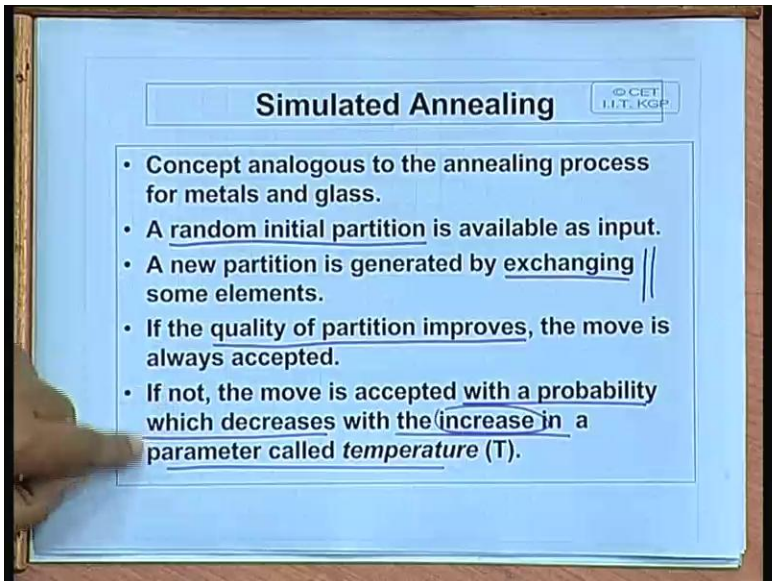
- These belong to the probabilistic and iterative class of algorithms.
- Simulated Annealing
  - Simulates the annealing process used for metals.
  - As in the actual annealing process, the value of temperature is decreased slowly till it approaches the freezing point.
- Simulated Evolution
  - Simulates the biological process of evolution.
  - Each solution (generation) is improved in each iteration by using operators which simulate the biological events in the evolution process.

These algorithms are probabilistic because they depend on some kind of probabilistic behaviors. They are also iterative, there is an iterative loop. So simulated annealing is a very common algorithm and simulated evaluation is evolution is well it is an extension you can say. As I told you genetic algorithm is it also falls under the simulated evaluation category. Now the algorithm of simulated annealing this is a generalized algorithm frame this is not specific algorithm which can be applied to partitioning only. Simulated annealing is a generalized algorithm where there are certain steps of the algorithm which you can modify suitably to apply it to any particular domain.

So we will see how we do it for partitioning. Now as the name implies, the algorithm of simulated annealing it tries to simulate the annealing process which is used for metals. Normally in metals what happen? You raise the temperature to a very high value. The metal becomes a liquid state then following a cooling curve you slowly cool down the temperature. So the metal forms a means a depending on whether it is a metal or glass just it has a solidifies or crystallizes depending on certain requirements which you want. So the cooling curve will determine exactly what you want as the final thing. So the concept is similar you raise it to a very high temperature.

Means, at high temperature something random can happen. But as it slowly reached the freezing point, the mobility of the atoms or the molecules they get less and less. So the idea is like this. So at high temperature may be something random is happening, you can go to a worse solution. But as the temperature cools down slowly, the chance of going to a worse solution becomes less. Well you are basically cooling down to a local minimum kind of a thing. You are approaching a good solution. Towards a good solution you are going. So the idea is something similar. So let us see what this simulated annealing looks likes. Simulated evolution, we will be looking at later.

(Refer Slide Time: 32:24)



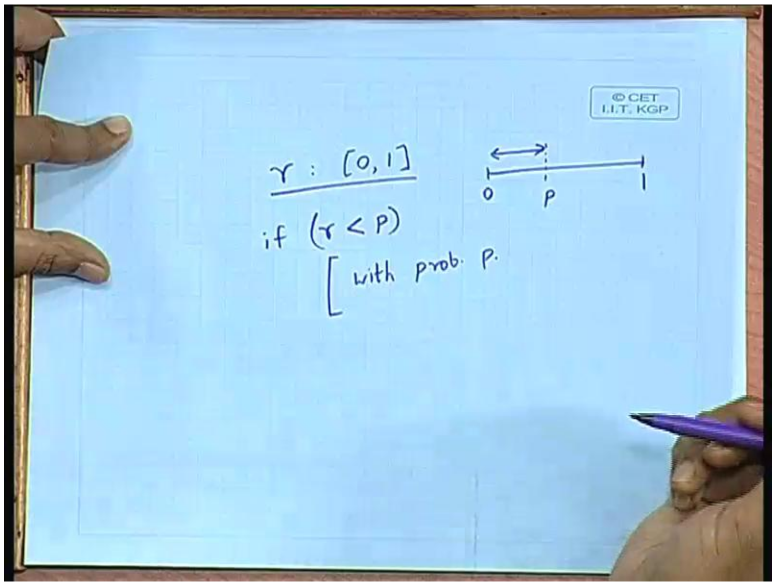
Simulated Annealing

- Concept analogous to the annealing process for metals and glass.
- A random initial partition is available as input.
- A new partition is generated by exchanging some elements.
- If the quality of partition improves, the move is always accepted.
- If not, the move is accepted with a probability which decreases with the increase in a parameter called *temperature* (T).

Well, so here we are talking with respect to our problem domain which we are looking at the domain of circuit partitioning. So to start with we have a random initial partition. This can be generated randomly also. Well, like Kernighan-Lin algorithm, we also do some kind of trial exchanges or movement like Fiduccia-Mattheyes algorithm. So we make some changes to the partitions. We make some changes and evaluate that what is the quality of the new partition we get whether it is a better one or a worse one. So if the quality of if the quality of partition improves, then the move is always accepted.

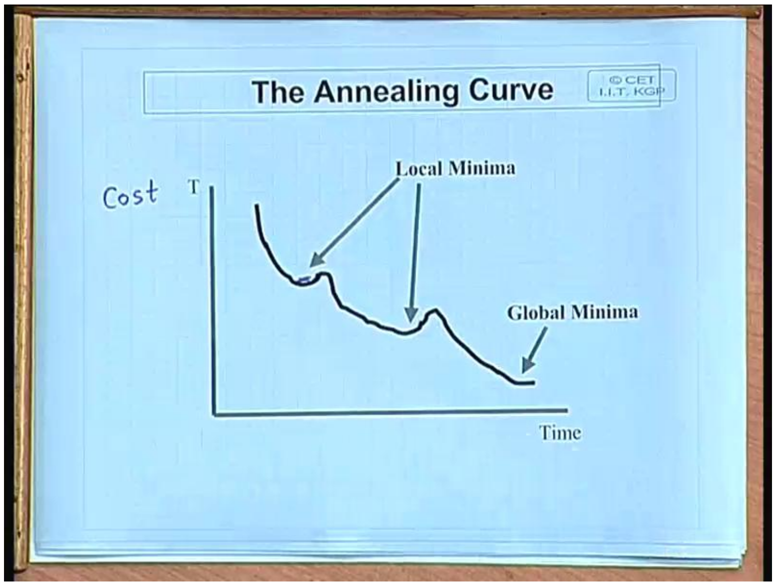
So as we move towards a better solution, we always accept it unconditionally. But if we see that the new partition is a worse one. It is not improving rather it is degrading still we can accept it with a probability which decreases with the increase in a parameter called temperature. Or means it will be decrease actually not increase really. It depends on that whether you call increase, decrease with respect to temperature it decrease. So at higher temperatures the chance of accepting a worse move is higher. At lower temperatures the chance of accepting a worse move is lower. Now you know how you do this kind of program randomly in a program.

(Refer Slide Time: 34:08)



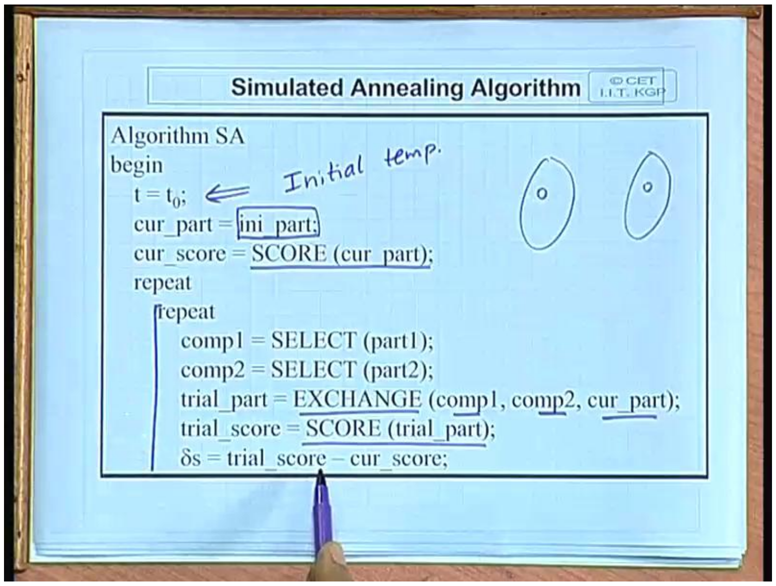
You can generate a random number say r in the range of 0 to 1. And if p is your probability, you can check if r is less than p or not. Now if this is a uniform random number, uniform it is I mean it follows a uniform distribution. Then the probability of any number being generated between 0 and 1 are equally likely. So if I set some where a line p, so the chance of numbers being generated in this area will be having a probability p. So if r less than p then do this with probability p. This is how we implement this probability thing in the program. So with probability p we can do whatever you want in this block. So we generate a uniform random number between 0 and 1; and check whether that number lies within the probability where we specified then we do whatever we want.

(Refer Slide Time: 35:23)



So in terms of the annealing curve. Okay. Well, with respect to the partition, you can call this axis to be the cost, this axis to be the time. So we normally go from a higher cost to a lower cost. But sometimes we can accept a move with a what solution. So sometimes we may move up. Sometimes we may move up. So this moving up may sometimes prevent us from falling from the local minima, come out and go towards the global minima. So the idea of accepting what's move is this that you don't get stuck into the local minima if you strictly follow a greedy approach. So this is how it is. So now let us look at this simulated annealing algorithm. How this algorithm looks like with respect to partition? Well I am showing you the pseudo code.

(Refer Slide Time: 36:22)



Simulated Annealing Algorithm

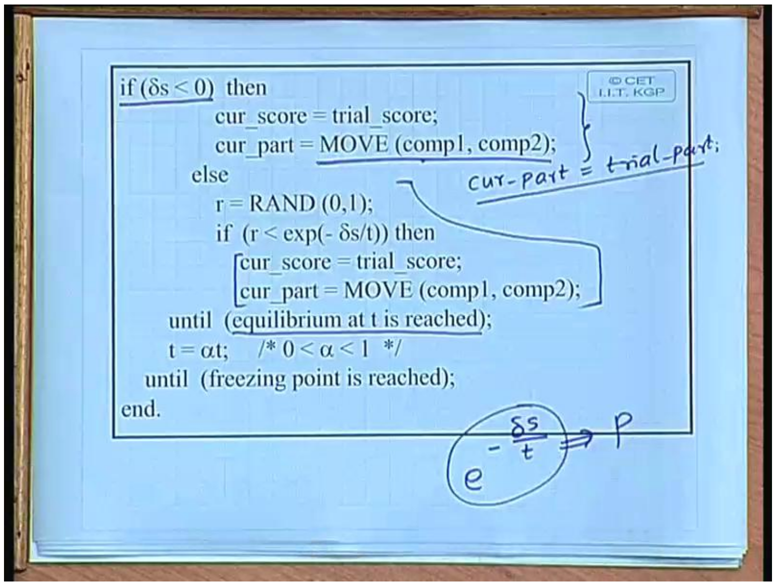
```
Algorithm SA
begin
    t = t_0;
    cur_part = ini_part;
    cur_score = SCORE (cur_part);
    repeat
        repeat
            comp1 = SELECT (part1);
            comp2 = SELECT (part2);
            trial_part = EXCHANGE (comp1, comp2, cur_part);
            trial_score = SCORE (trial_part);
            δs = trial_score – cur_score;
```

Initial temp.

Fine. In this algorithm you just look at. This t is the temperature. You start with an initial temperature. In terms of the program this is just a variable. You initialize the variable with some high value typically 1000, 10000 depends on what value. Current partition equal to initial partition. So you initially generate a partition. This can be randomly generated partition and you call it cur part. This can be structure, this can be anything. I mean just in the program I am just showing it like this. Current partition is the initial partition we have. I compute the cost of the current partition by calling a function called score. Score of current partition will give me the current score. Current score is actually the cost or the goodness of the partition you have. Then there is a repeat loop. In fact it is a nested repeat loop.

So in the inner loop what I do? So I select two parts or two nodes of the graph. These nodes may be selected at random or this may nodes this nodes may also be selected based on some heuristics. We will just see this heuristics later. But somehow we are selecting two nodes. One from this partition, one from this partition. This you call them component1, component2. And as a trial we exchange them. Trial partition is obtained by exchanging component1 and component2 with respect to the current partition. So here we get a new partition that we call the trial partition; by exchanging these two. We evaluate the cost of the trial partition. This is the trial score, then we see what is the what is the increase in cost delta s is equal to trial score minus the current score. This is the increase in cost. If this is negative it means we are going towards a better solution.

(Refer Slide Time: 38:36)



© CET I.I.T. KGP

```
if ($\delta s < 0$) then
        cur_score = trial_score;
        cur_part = MOVE (comp1, comp2);
    else
        r = RAND (0,1);
        if ($r < \exp(-\delta s/t)$) then
            cur_score = trial_score;
            cur_part = MOVE (comp1, comp2);
  until (equilibrium at t is reached);
  $t = \alpha t$;    /* $0 < \alpha < 1$ */
until (freezing point is reached);
end.
```

cur-part = trial-part;

$$e^{-\frac{\delta s}{t}} \Rightarrow P$$

So after computing delta s we check if delta s is negative. So if it is negative we are moving towards a better solution. So a better solution is always accepted. So now the trail score will become the current score and well now we are making an actual move say. Means earlier this was just a trial exchange we are not actually modifying the current partition. See this you can write in two different ways this was trial partition. So either you can write it like this or you can simply write current partition equal to trail partition so either way means the same thing. So here you are making the actual change to the current partition. But otherwise you are trying to move towards a worst solution. Now this worst solution you can accept only with the probability that I had mentioned. Now the probability is computed like this.

First you generate a random number between 0 and 1. This is r and the probability value here is computed as e to the power minus delta s by the temperature. Now larger the value of delta s less will be the probability. So if you are going towards a very bad solution the chance of accepting it will be less. But if delta s is less then this value will be more. So this is the value of the probability p which I was talking about. So you check if r is less than this probability value. If so, then you accept it. So here we accept it. The same thing same code out here. Otherwise you do not accept it. So this inner loop is repeated until equilibrium at the current temperature t is reached. Well now in terms of a practical program this will mean that the inner loop will be repeated certain n number of times.

This n can be 1000, it can be 5000 depends on the application again the number of times you have to repeat the inner loop, that will depend on the problem while it can be less also it can be 100 also. So this is the inner loop. Now the outer loop now means you recall there were two. [Students noise not audible (Time: 41:10)] Well equilibrium with respect to the annealing process that is meant. That for a particular temperature you reach the equilibrium then we again drop the temperature [Students noise not audible (Time: 41:21)] Yeah. For our case what you do is that for a particular temperature we carry out a number of exchanges still we see that we do not get significant improvement anymore. So we are trying to reach some kind of equilibrium at temperature t.

Then we again decrease the temperature. So this particular loop is executed, then we come to the outer loop. This outer repeat loop, this outer repeat loop is actually dropping the temperature. So the outer repeat loop what it will do? It will multiply the temperature with a parameter alpha which is number less than one. So t will get less, so until freezing point is reached. Well in terms of program this can mean until t less than some 0.01 one or something may be. So you start with an initial temperature. The outer loop will continuously reduce the temperature till it comes below a certain piece specified value. And the inner loop again is repeated certain number of times. Now you can see that an algorithm like this depends on number of different parameters. Well of course, well how you compute the cost that may not be very difficult to get.

But this moves how to select the nodes to move. That also for a particular problem you can specify. Well these are the possible move you can define. You select one of them and make a move. But with respect to the number of iterations you do, the initial temperature we select, the value of alpha these are all parameters of this algorithm. And for a particular application you will have to do extensive experimentation and fix up those parameters which give you the best result. Now once you fix up these parameters you can see that the quality of result you get using these will be much much better as compared to Kernighan-Lin for example. So in cad there are many algorithm algorithms which are there for this is one partitioning for placement, for routing also. There we use some sort of simulated annealing or some extension of it.

And these are actually used in cad tools not just these are these are just used for academic interests. [Students noise not audible (Time: 43:42)] In one temperature whatever you are doing in the whole of Kernighan-Lin it is true. But in the Kernighan-Lin we are always following a greedy path. We are always trying to find out the pair of nodes whose exchange give the best. But here we are not sticking to that restriction. May be we are coming out of the global minimum well. [Students noise not audible (Time: 44:09)] So Kernighan-Lin will converge much faster. But here we are exploring much more more of the search space. [Students noise not audible (Time: 44:16)] No, no never this will ensure a good solution in reasonable amount of time. Global minima never. But I suggest you take a very small application if you try to quote this program.

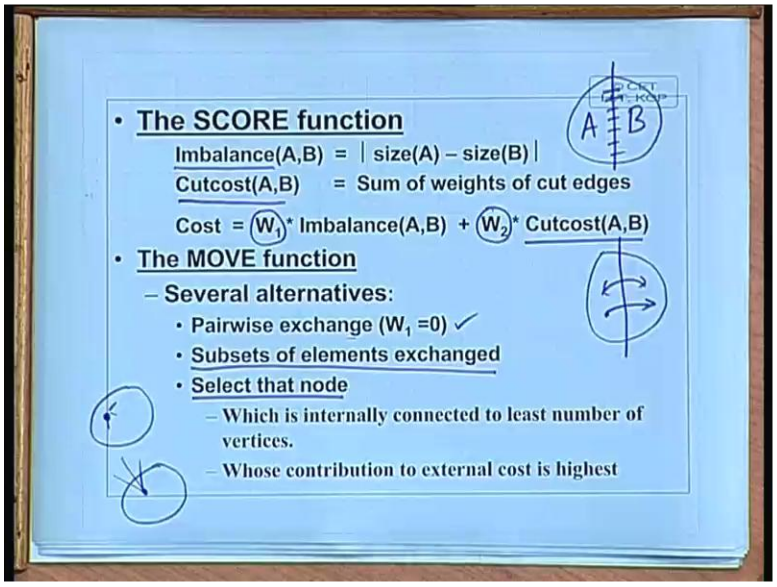
And you can just run and see that how the quality of the solution comes. So just you will get some a you just take some problem which is known to be completely hard like the graph coloring problem for example. You just code it using simulated annealing run the program and see that how it performs. You will see that you will get a very good solution in a reasonable amount of time. [Students noise not audible (Time: 44:56)] No, equilibrium t is reached means you are giving some chance for some exchanges to take place at each iteration. Because you are just picking up two nodes at random. Well may be its possible that the nodes are such that the cost in increasing and you are not accepting that move at all. So the solutions will not change. So your giving some chance for the solutions to change [Students noise not audible (Time: 45:26)]

Yes, there is a delay means it is the [Students noise not audible (Time: 45:34)] you can say that yes. [Students noise not audible (Time: 45:41)] Well, delay you really do not call it a delay; t is more like an iteration you can say it is the iteration number some kind of measures of the iterations you are doing. That is the outer loop you know. So it is [Students noise not audible

(Time: 46:04)] That is the iteration number and as the iterations are going on, the value of that p is also decreasing that value of probability e to the power minus delta s by t. So as t is decreasing, that value of e to the power minus delta s by t is also decreasing. So in the future iteration the chance of accepting a worse move is less. [Students noise not audible (Time: 46:27)]

In the inner loop t is constant, yes. In the inner loop probability remains same [Students noise not audible (Time: 46:35)]. Now inner loop we are trying to make some some exchanges. Well most likely it will lead to a better solution. Sometimes with some probability we can also accept the worse solution. So means we are trying to get a better solution with each in a loop most of the time that is our objective. But some of the time we can also move to worse solution that also we want. We want it consciously so as to come out of the local minimum. [Students noise not audible (Time: 47:10)] For the number of iteration. Yeah. So this can either be a fix number of iterations or the solution is not changing over the last two iterations or three iterations. You can do it either way.

(Refer Slide Time: 47:32)


- The SCORE function
  - Imbalance(A,B) = | size(A) – size(B) |
  - Cutcost(A,B) = Sum of weights of cut edges
  - Cost = $\boxed{W_1}$* Imbalance(A,B) + $\boxed{W_2}$* Cutcost(A,B)
- The MOVE function
  - Several alternatives:
    - Pairwise exchange (W_1 =0) ✓
    - Subsets of elements exchanged
    - Select that node
      - Which is internally connected to least number of vertices.
      - Whose contribution to external cost is highest

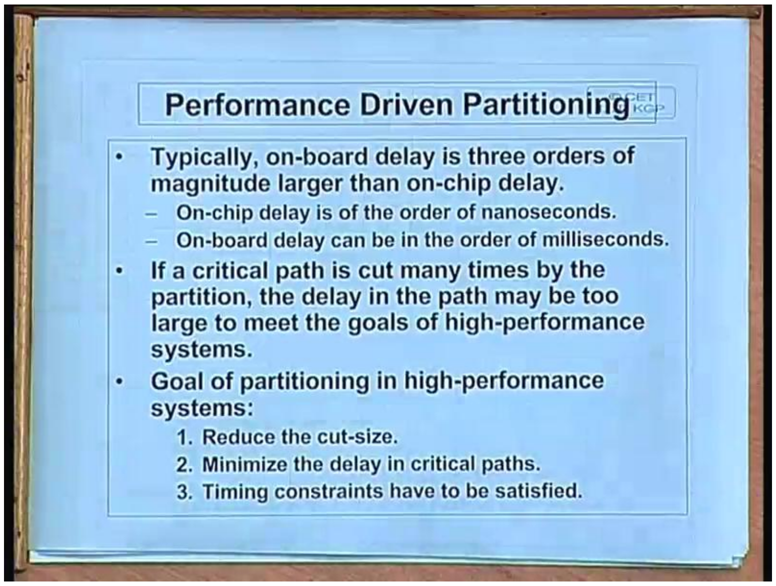
So in terms of the details this score function, the way you can compute the cost for partitioning is fairly easy. Say if you have system, we were your partitioning into two sub systems a and b. So you can define a factor called imbalance. Size of a minus size of b, magnitude of that, so the difference in sizes. Well if you are, if your initial objective is to obtain exactly balanced partition, then this imbalance should be 0 ideally. But in some cases it may not be so, where you can accept imbalances which are not 0. [Students noise not audible (Time: 48:21)] Nodes well, it depends if the sizes of the nodes are different then sum total of those numbers and similarly cut cost. The number of or the weights of the lines which are crossing the partition some of the weights of cut edges.

[Students noise not audible (Time: 48:38)] It depends because you see that the total cost you can multiply them by some weighted factor w1 and w2 and you can if you can have this cost function. Now if suppose you are starting with a balanced partition and if you are only using moves which exchange two nodes, then the partition will always remain balanced. In that case w1 may be set to 0 because imbalance will never take place. So there, so here you are whole concentrating on cut cost. S0o it depends on the situation where you are using. But in some cases you can define some moves where the imbalance may occur. You can move one node from a partition to the other so that the partition sizes may become unequal.

So there you can also have to take care of imbalance. So for the moves you can have again several alternatives. So I told you the simplest is the pair wise exchange. So you can have a pair wise exchange for this w1 can be set to 0 or you can have it slightly more generalized. Instead of exchanging one element at a time you can exchange subsets of elements at a time. Say you take three from here, three from here, exchange them. Or you can have some kind of an heuristics using which you can select the node to exchange like well. The algorithm like Kernighan-Lin algorithm spends a lot of time to evaluate the nodes which are the ones to exchange next. But you can make it slightly simpler. Well you can look, suppose we have these two partitions well you can try to find out a node inside a partition which is minimally connected to other nodes inside the same partition.

Which means that this node really do not belong to this family. You can take it to the other. Or you can try to find out a node where you see that it has lot of connections to the outside. So either which is internally connected to least number of vertices or whose contribution to external cost is highest. So you can give preference to those nodes for exchange. So it really depends you can either select the nodes blindly or you can have a little bit of intelligence. Just put in there so that you select node whose exchange can possibly lead to a better thing. Well you can have a combination of these things. That means you sometimes you select blindly sometimes you select this. Because if you always select this, again you are trying to follow a path which is greedy. And you are trying to avoid the greedy path. Fine.

(Refer Slide Time: 51:38)

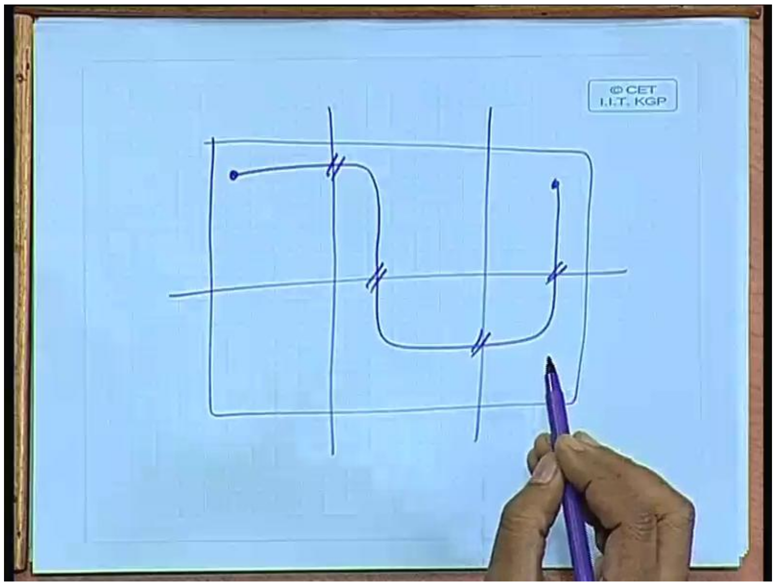


Performance Driven Partitioning

- Typically, on-board delay is three orders of magnitude 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.

So now let us look at performance driven partitioning. As I told you earlier that this performance driven partitioning. These are some requirements which come from the user's point of view and there are no good algorithms which can take care of this restrictions in a very generalized way. Now these are the few things which we already mentioned that the onboard delay is significantly higher than on chip delay. So typically this on chip delay can be of the of the of the order of nanoseconds. Then on board delay can be as high as milli seconds. It can be 1000 times higher also. Because in that in that hypothetical example earlier, I just showed you 20x. But in practice it can be much much higher than that. So now, when you are doing a partitioning.

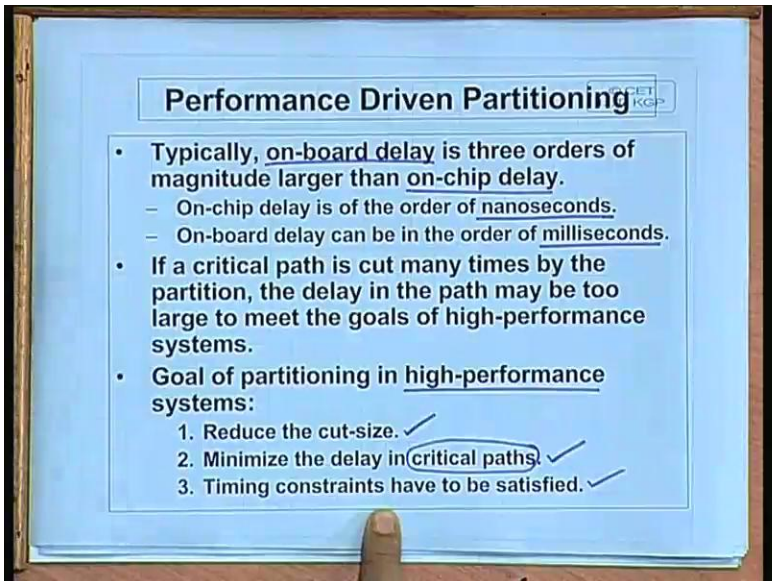
(Refer Slide Time: 52:38)



Suppose I have a big netlist which I am doing a partition. Suppose I have partitioned it into six pieces. Now these pieces will be mapped into either a chip or two different chips. They may be put on the same board or different boards. Now it is possible that in this circuit there is a critical path which is goes like this. I will have to see that how many times this critical path crosses the partitions. [Students noise not audible (Time: 53:09)]
Critical path is that path which contributes finally to the speed of the operation of circuit. For example the maximum clock frequency with which a circuit can operate.

There are some paths which well if you increase the delay in that path slightly the clock frequency will go down. But some other parts, there is there is some flexibility you can add some delay also without hampering the total performance. So you look at the critical paths. You look at number of times it crosses the boundaries and means what kind of boundaries they are. So you try to move things around so that critical path cost gets minimized. And moreover a path which was not critical in the original netlist after partitioning it can become critical because of these added delays. So this performance driven partitioning has to take care of these things.

(Refer Slide Time: 54:10)

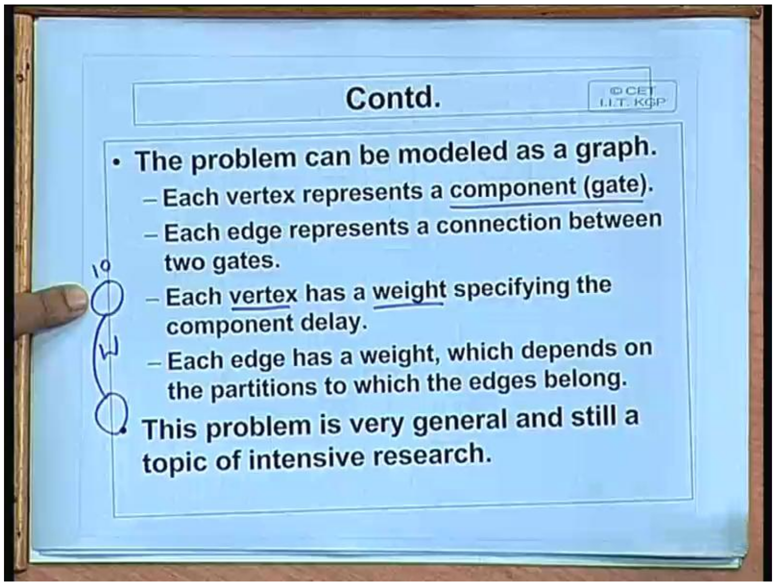


Performance Driven Partitioning

- Typically, on-board delay is three orders of magnitude 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. ✓

So if your critical path is cut many times, the delay in the path may be too large. So in case of high performance system when you do a partitioning, of course the goal is to reduce the cut size. But also we have to reduce the delay in the critical paths. Now here the problem is that the definition of the critical path is changing dynamically as I have just mentioned. So the initially a path was not the critical path, but after the partitioning it is crossing high cost boundary. So it becomes a critical path. And some timing constraints which has some user this can be said user level specification. They have to be satisfied.

(Refer Slide Time: 55:00)

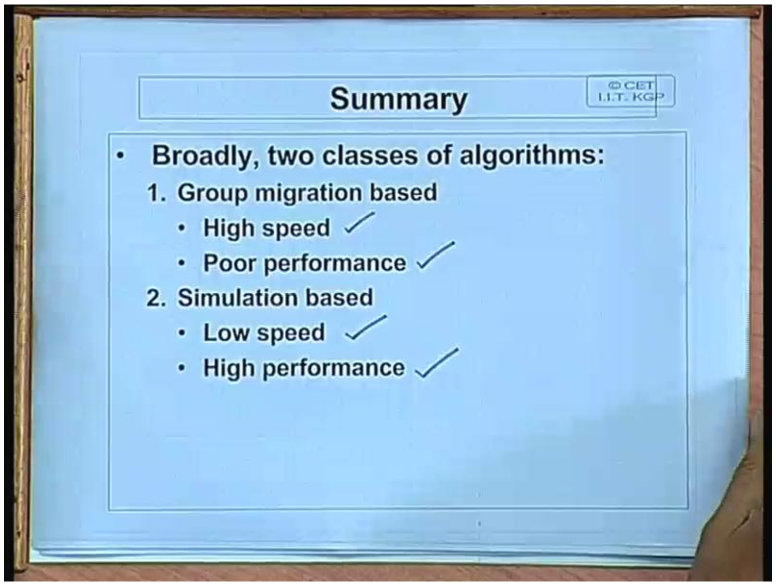


Contd.

- The problem can be modeled as a graph.
  - Each vertex represents a component (gate).
  - Each edge represents a connection between two gates.
  - Each vertex has a weight specifying the component delay.
  - Each edge has a weight, which depends on the partitions to which the edges belong.
- This problem is very general and still a topic of intensive research.

So means one way people have approached this problem is that here you can model it as a graph. Where each vertex represents a component or a gate. Each edges represents a connection. The vertices have a weight. In this model the nodes of the graph will also contain a weight. Thus that represent the component delay. If it is a NAND gate, what is the delay of that NAND gate? Each edges of weight, weights which depends on the partitions to which the edge belongs. If this nodes is connecting to another node depending on whether they belong to the same partition or different partition this weight w, can vary. So you model a graph like this and you try to dynamically obtain a good partition where the total delay will be computed also taking into account the node delays. So the problem is very general. As I told you this, this is still a topic of intensive research; there are number of heuristic algorithms which have been proposed which people use. But there are they are those algorithms are certainly not very satisfactory and there are a lot of scope for improvement.

(Refer Slide Time: 56:13)



Summary

© CET I.I.T. KGP

- Broadly, two classes of algorithms:
  1. Group migration based
     - High speed ✓
     - Poor performance ✓
  2. Simulation based
     - Low speed ✓
     - High performance ✓

So to summarize the partitioning algorithm we have talked about broadly two classes: group migration and simulation based. To compare them group migrations take less time. But in general produce partitions which are not of good very quality. But simulation based take longer time, but they give very good solutions. So in our next class we would be talking about some other step of the physical design process, namely we would be talking about floor planning and pin assignment. Before we can start doing the placement and routing these are some steps we need to do. Thank you.