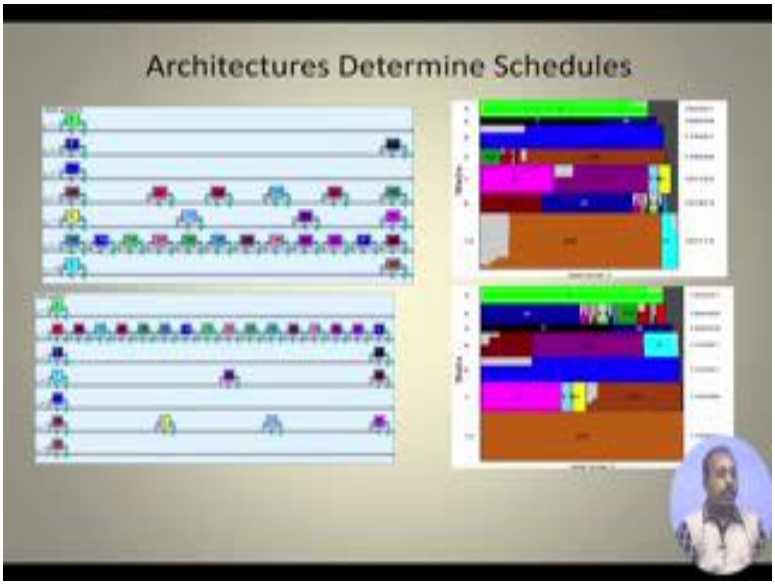
Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 49
System/Network –On-Chip Test (Contd.)

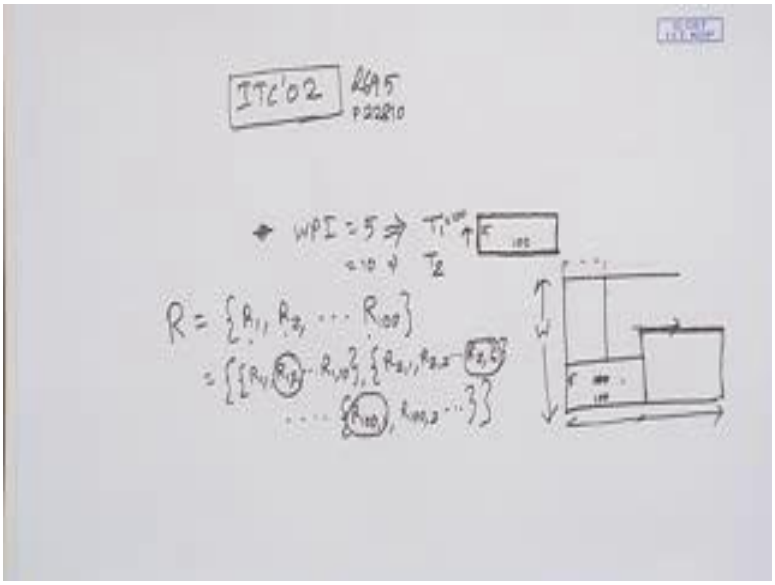
So, test architecture it is going to determine the schedule.

(Refer Slide Time: 00:22)



So, in this particular slide you see that this is a TAM distribution of different cores of this particular SOCP 22810. So, this is a benchmark SOC that has got a number of cores in it. So, there are I think 26 cores in this particular benchmark. So, they come under the ITC 02 series of benchmarks. So, they are called ITC 02 benchmarks.

(Refer Slide Time: 00:58).



So, this ITC 02 benchmarks it has got a number of SOCs in it. So, they are the SOC benchmarks. And there are many such benchmarks in it, like it starts with say d 695, then we have this thing we have got d 695 in this p 22810. So, this is another benchmark. So, these types of benchmarks are there so that we can it they are often used for testing the component; testing the different test algorithms for this purpose.

So, this 22810, so it has got some 26 cores in it; so in this case this TAM is divided into number of sub tams. So, this first one is of width 4, second one is of width 2, third one 6, then 4, then 7, then 5, then 14. So, then the individual cores are put on to the TAM like this so the on first TAM we have got only core 2, second TAM we have got core 8 and core 7, third TAM we have got core 1, fourth TAM they are 25, 4, 19, 17, 20, 12 etcetera.

Now if you see the corresponding schedule, so core 2 is such that it requires a huge test time. So, it required this much test time. Similarly this says in the second TAM, so this 8 and 7. So, this 7 is tested first, so it takes this much time and then 8 is tested. So, it is not mandatory that if the cores are placed earlier on the TAM. So, it will be tested first it is not that it may be due to some precedence constraint or may be due to some other this the scheduling algorithm the ordered in which we fix up the core. So, that may determine the schedule.

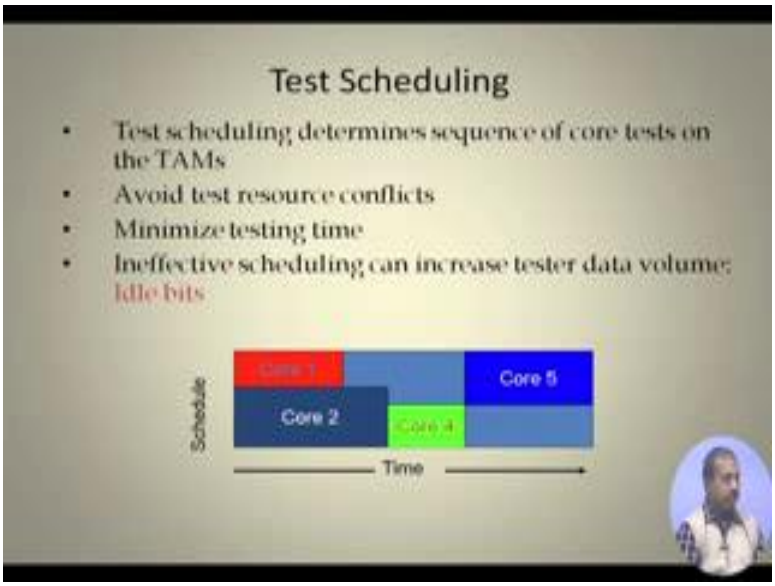
So, that here in this case what has happened is that this TAM this schedule has taken up core 7 first this algorithm, so it has been scheduled first. So, this part is; so this core 7 is tested here then core 8 is tested here. Now, then this core 6 is put on to this TAM, but you see that this is taking this much of time. Then we have got different other cores on TAM 4, so there are some cores. And what has happened is some time some parts are blank. So, actually in this part no testing is done for this; no TAM lines are used in this part. It may be due to the test patterns that we have the test pattern with may be different and all that. So, it may be there may be some gaps here like this.

So, this way we have got the test times and this test times are plotted here. And the maximum test time is given by this one; this TAM 4 this TAM of with 14 and that has got the test time of 187712. In the entire schedule that actually constitute the maximum time, so the total time needed for testing is 187712.

You take some other schedule. So, may be here on TAM 2 we have got on the first TAM we have got this core 2, on second TAM we have got all this cores and rest of the TAM so they are lightly populated. So, you can see that that test times that we have, so that is two here for the 2; 2 it takes this much time. For other cores they are taking some different amounts of time, but you see that the total test time needed in this case becomes 176900. So, it is less than this one.

So, that is what it is said the architecture they are going to determine the schedule, and as a result it is going to determine the test time that will be needed in the process.

(Refer Slide Time: 04:49)



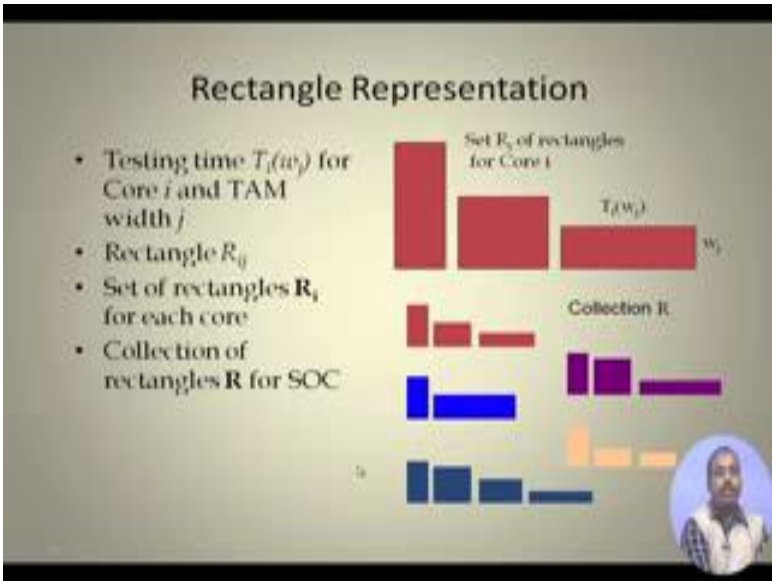
So, what is the test scheduling problem? Test scheduling it determines the sequence of core test on the tams. So, tams are there, so which core will be assign to which TAM and in which sequence they will be tested. At what time that particular core will be tested? Ever test resource conflicts. So, this is another issue that is if you have putting two cores on the same TAM so you cannot put that test times over lapping, because in that case the there is a TAM resource conflict. So, this resource conflict has to be avoided and then we want to minimize the testing time. So, this TAM distribution and this scheduling should be done in such a fashion that this test time is minimized; that in effective scheduling can increase test are data volume or idle bits.

So, if you have some gaps, so in this case you see that in the total time. So, you have got suppose the one schedule generated is like this that initially core 1 and core 2 they are being tested in parallel, then after sometime core 1 finishes but this available TAM width is not sufficient for testing any of the remaining cores core 4 and core 5 either of this. So, it is not possible to test them. So, what we have? We have got some blank squares. So, core 4 can be tested only with this much of TAM width. So, with this much of TAM width this can be tested only after core 2 has finished. And core 5 can be tested after this core 4 has finished as a result; so these two regions. So, they are actually some blanks that are coming so they are known as idle bits.

But you see in the test from the tester we are sequentially sending the test bits for individual cycles. So, you can think of it like this as if tester initially it has given me the test pattern for the first cycle to be shifted into the system, there it giving me the second test pattern second test pattern part to be shifted in the second tests cycle. So, that way it was progressing fine. So, here also I cannot ignore this time because I am going to same some valid data for core 2 in this time and some valid data for core 4 at this time and some valid data for core 5 at this time.

So, this bits from the tester it will be transmitted, but will not be used for testing. In the tester memory, so these bits will occupy some space that cannot be avoided. So, this actually gives rise to idle bits. And as we already know that this 80 cost is determined by partially by the memory requirement that it has. So, this memory requirement will make this idle bits to be very costly; so its idle bits storage that will become costly. So, any schedule that we make; another idea behind generating this schedule should is that it should be compact there should not be such many such idle port portions or blank portions in my schedule.

(Refer Slide Time: 07:59)



So, one technique for reducing these gaps in the schedule is by means of another scheduling policy where this test, this each core is assumed to be presented by a set of rectangles. So, says that you have seen previously that if a core is in that wrapper design phase, so if a core is tested by say this WPI equal to say 5, then it is w WPI equal to 5

then it is going to say take some amount of test times. So, that let the corresponding test time be T 1 and if WPI equal to 10 it is going to take some other test time T 2.

So, that way it can be represented by a rectangle. It can be represented by a rectangle like this; on the one side we represent the number of WPI lines that we have kept. So, this side is say 5 and this side is say T 1 equal to say 100 this side is a 100. So, if I give it TAM of width 5, so it gives me rectangle whose length is 100 and width is 5. So, this is as if on the tester on the schedule; so on the schedule it is going to occupy some part of the schedule and that part of the schedule will be filed up by means of this rectangle. So, you can think of this whole thing as if there is a bin where this bin height is fixed. So, this total bin height is given by W. So, that is fixed, and in that bin we are trying to put the rectangle. So, if I put this rectangle at this point, so this is the situation. So, this is 100 and this is this side is 5. So, this side is 100 this side is 5.

Now at no point of time I can put rectangle so that I cannot put a rectangle like this, cannot put a rectangle like this because it exceeds the width W, but this side there is no restriction. So, this side if I proceed further then what will, so t also if we if there is only this core in the system and this is my test time. Now if I put another core which is going like this another rectangle like this rectangle of another core like this then I have got the test time extended to this much.

But that is also a valid schedule, but may not be a good one but that is also a valid schedule. So, what is required is that for every core we have got a set of test rectangle. So, this are the different cases like, we have got one rectangle its width represents W j that is the number of timelines that are allocated to it in the wrapper design. And this T i W j is the corresponding test time. So, from the wrapper design will know what is the corresponding test time, so this will be the T i W j.

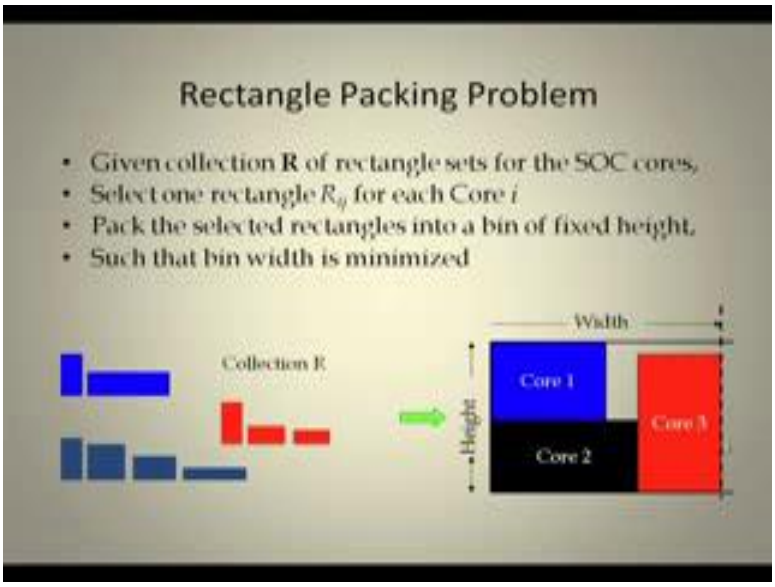
Similarly another case may be this is the W j value and accordingly we have got this as the T i W j. Third case; so this is the W j value and this is the T i W j value. So, testing time T i W j for core i and TAM width W j so that is taken. So, that represents the rectangle R ij. In general I can say that any core can be represented by a set of rectangles capital R i. So, for the testing purpose I can represent any core by a set of rectangles. and if we take all this sets then we get the collection capital R of this sets.

Now the situation becomes like this; this capital R is a set of this R 1, R 2; suppose I have got 100 cores, so R 1 R 2 up to R 100. Now this each of this R i's. So, they are basically again R set. So, it is again represented by say rectangle 11, rectangle 12. So, suppose it has got 10 rectangles in it then the second core it has got say 5 rectangles in it R 21, R 22, R 25. So, like that 100th core it has got rectangles R 100, 1; R 100 2; like this.

Now in the test scheduling process what I have to do is I have to select exactly one rectangle from each of this sub sets. Like from this one may be I take I pick up this one, from this one I pick up say this one and from this one I pick up say this one. Now these three rectangles are have been fixed now, they are to be put into this structure such that the total test time is minimum fine. So, I have got choice in the sense that I can choose different rectangles from this individual sets and I can pack those rectangles into this structure in a in different fashions.

So, we have got the options for doing the optimization. So, we first generate this collection of rectangles R for the SOC and the next part will be to do selection of core rectangles and placing them. So, this is the collections. So, this is the set of cores for the set of rectangles the first core, these are the second core, third core, 4th core. So, it is not mandatory that all of them will have equal number of rectangles it can be different numbers as well.

(Refer Slide Time: 13:46)



So, this gives rise to a version of the problem which is known as rectangle packing problem. So, this schedule problem is now translated into something called a rectangle packing problem. In a rectangle packing problem it is like this the general statement of the problem is like this that you are given a set of rectangles and those rectangles are to be; you have to put maximum number of rectangles in to a given bin. So, bin has got a fixed width and the height is infinite or if you want to pack all of them then you have to pack them in such a fashion then the that the height is minimized.

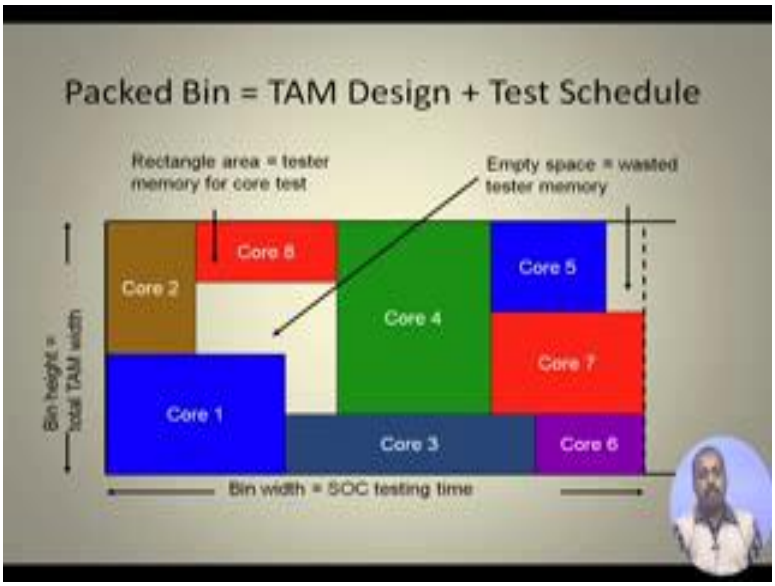
So, here also I have got a similar version of the problem, but the problem is likely more complicated, because now I have got choice. So, in the original bin packing problem I do not need to consider any choice of rectangles, so all those rectangles are to be packed. But here I have a choice from each set I can choose a single rectangle to be packed. So, that gives rise to this rectangle packing problem.

So, given collection R of rectangles set for the SOC cores select one rectangle R ij for each core I and pack the selected rectangles in a bin of fixed height such that bin width is minimized. So, this height is fixed which is given by the TAM width and then the bin width has to be minimized which is basically the maximum test time for any core.

So, this is the collection that we have. And then this is the height; so height is fixed. So, height is given by the TAM width and the width is variable. So, I have to pack one rectangle from each set such that this width is minimized. So, maybe first we pick up core 1 and we put this rectangle here, then we take core 2 we put another rectangle here, then this core 3 it requires more TAM width that that does not fit here. So, it has to be kept at the end of core 2, so that is the thing.

So, the total test time is given by this one. So, this is the bin packing problem.

(Refer Slide Time: 15:59)



So, packed bin that gives us the TAM design plus test schedule; you see suppose after doing this packing and all that we see that this is the distribution, fine. So, core 1 is tested in this TAM, core 2 is tested at this TAM, core 3 is tested here, core 8 tested here. And this width are also known, so how many bits we have assigned for the rectangle that we have chosen from for core 1 that gives me the width of this TAM that is used here.

Similarly, this here core 2, so that this one gives me the width of the TAM that is used for testing core 2; so like that. So, there are some empty spaces; you see here there is an empty space and this empty space. So, this empty space is giving me the wasted tester memory. So, bin height is equal to the total TAM width, so that constraint is always maintained here.

So, you have got some empty spaces which is the wasted tester memory. And this rectangular area, so these gives me the tester memory for core test; because in each of this bits that means over time steps so the time is test time is advancing and for every time step I have to give 1 bit to core a number of bits to core 8 whatever be the TAM width allocated to core 8 the bits had to be given, similarly bits had to be given for.

Actually for every time instant you can say that this entire set of bits had to be given. So, you can find out what is the total TAM memory requirement which is given by this TAM width multiplied by this total test time SOC test time or bin width. So, bin height into bin

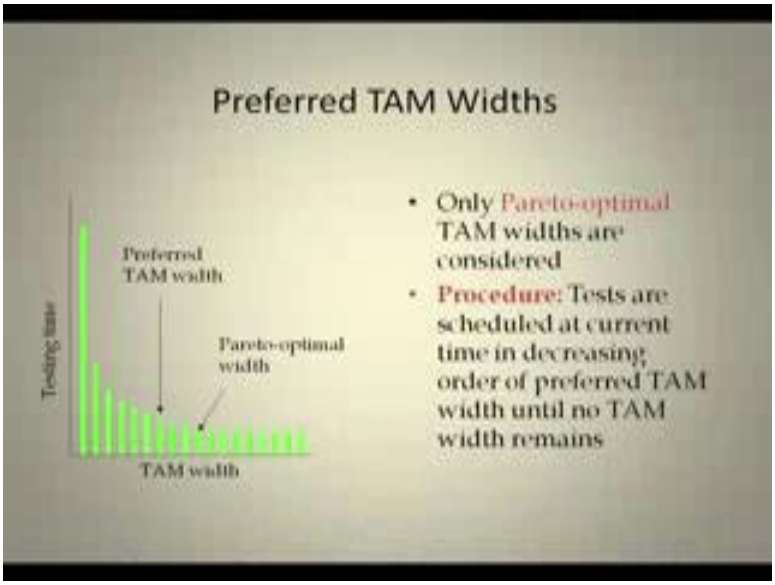
width, so that will give us the total tester memory requirement for the thing. And the test time is also obtained from this one.

So, getting this; so there are many heuristics for this bin tacking problem so we will not go in detail for them; any book on optimization that will discuss on this bin packing problem heuristics and all that. So, this is an empty hard problem, so you cannot guarantee to get an optimum solution planning a polynomial time algorithm, but there are many good heuristic that are been developed and any of them can be used for this one.

So, the some of the heuristics that we have; so they actually try to figure out say best feet decreasing heuristic; so the large rectangles they try to put fast like this here in this case also you see that core 2 is a very large rectangle; so that has been packed. Then somehow in the remaining part you see the core 2 could be pack, so core 2 has been selected. Then core 8 was selected by the heuristic and then ideally you try to put core 4, but core 4 could not be fitted because core 4 does not come into this case whatever we do.

So, some heuristics can be done and none of the heuristics can give us optimum result, so that is true. And there are techniques based on this evolutionary algorithm like genetic algorithm particles from optimization. So, all those techniques are been tried for solving this bin packing problem and particularly this core SOC testing version of the problem.

(Refer Slide Time: 19:18)



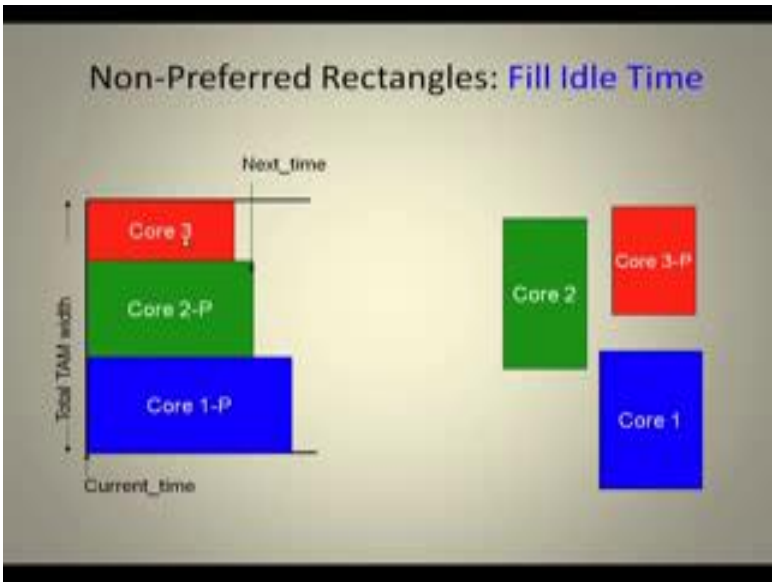
Now, sometimes so if you look into this the test times have a particular core as the TAM width varies. So, you see that it shows some sort of Pareto-optimal behavior. Like you see when the TAM width is equal to 1 may this is the test time when TAM width increases then the test time starts decreasing. Now you see that there is a significant decrease up to this point and then after this TAM width decreases, but the decrease may not be that high. So, if it is so then we say that that is this is going to be the preferred TAM width for the particular core, because here the width is also not that high, but test time is quite low.

And after this point you see that even if you increase your TAM width, so it is not going to increase decrease the test TAM further. So, that is called a Pareto-optimal point. So, beyond this point the test time is not decreasing. Only Pareto-optimal test width TAM width need to be considered. So, it is not required to consider all this TAM width. So, where this Pareto-optimal width is coming, so that has to be considered. Again if there are a number of points actually this, so some set of staircase behavior you have seen previously that at maybe for some time the TAM increasing TAM width does not decrease the test time, but after some step again there may be improvement in the test time with decrease increasing TAM width. So, that can happen. So, all those staircase points they are actually the Pareto point.

So, we have to we have to considered the testing with respect to those Pareto-optimal points only. So, tests are scheduled at current time in decreasing order of preferred TAM width until no TAM width remains. You see that for every core we have seen, we have got this preferred TAM width. So the procedure, this is this is the heuristic actually for this bin packing of this SOC testing problem.

So, here what is done at current time we try to see what are the preferred TAM width and then we try to allocate the core with the preferred TAM width till no width TAM width remains available.

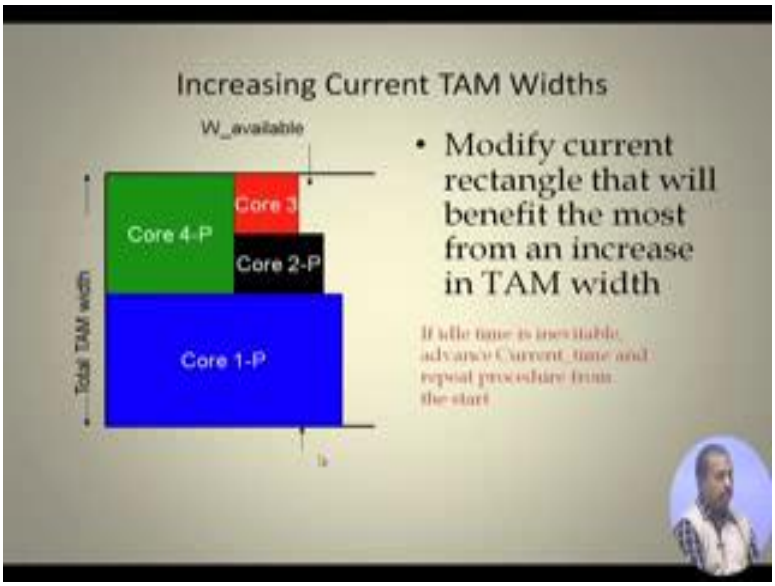
(Refer Slide Time: 21:41)



Suppose this is the situation. So, this is the current time; so core 1 and core 1 preferred time is this one because that is because of the Pareto-optimality and all that suppose this is the thing. So, this one is put into the TAM.

Now core 2: core 2 preferred core preferred rectangle is this one so it is put here. Now comes core 3s, can core 3s preferred time is this one, preferred rectangle is Pareto-optimal point is this one, but it cannot be put here so I have to go to the next time. So, core 3, so this is the other rectangle. So, we see that this fits, so if we decrease that TAM width from here it has been decreased; so when you decrease it fits in to the thing, so it is put here.

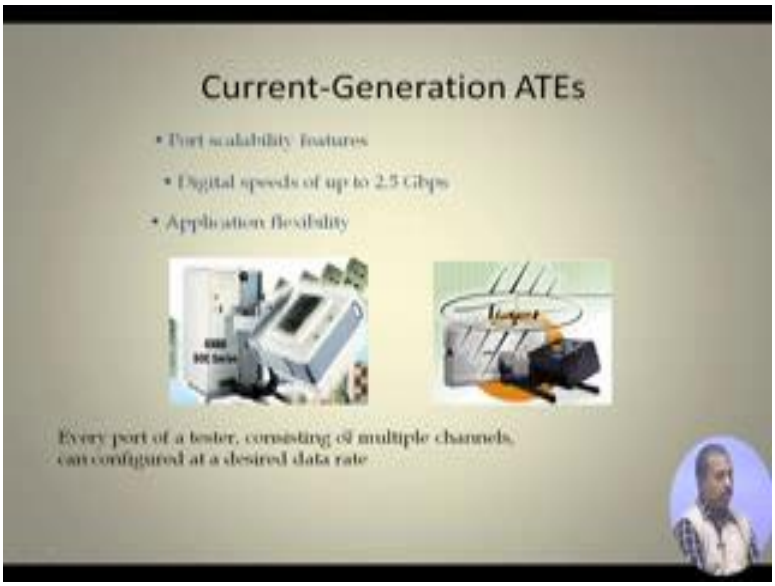
(Refer Slide Time: 22:34)



Now, sometimes we need to increase the current TAM width. So, modify current rectangle that will benefit the most from an increase in the TAM width. So, this example let us see, so core 1 is put here core 1 preferred, core 4 preferred put here. Now we see that this is the current test time this is the current test time. Now they are I can put this core 2 preferred that rectangle. So, this much of TAM is now available. Now there I can put this core 3 preferred rectangle if necessary, but you see here this was the thing but here the remaining TAM width is not sufficient for putting any new core so we try to utilize this much of TAM using some other TAM of core 3. So, you see that here it change it so that it uses more amount of TAM line more number of TAM lines, but the test time is reduced. So, this is basically deviating from that preferred one. So, preferred one was given less TAM width, now this one is given higher TAM width but essentially the test time improves.

So if idle time is inevitable; like at this point of time it may so happen that idle time is inevitable then we advance current time and repeat the procedure from the start. So, then the current time will be advanced, so it is here. Now you see that this cannot be modified further. So, it advances to the next point and from that point onwards again the scheduling will start.

(Refer Slide Time: 24:10)



Now in this situation; so this is the basic rectangle packing procedure. If you look into one thing that here you see that TAM width is not partitioned properly. So, you see that a typical example of this one; say this one- so here you see that there is no partitioning, no I cannot partition TAM into different partition. So this is basically the TAM width is going to be redistributed as we go to different cores. So, there is no partitioning of the TAM. So, partitioning is based solutions are not suitable for this bin packing approach. So, there we have to use the previous version that p w, p a w, p n p a w like that.

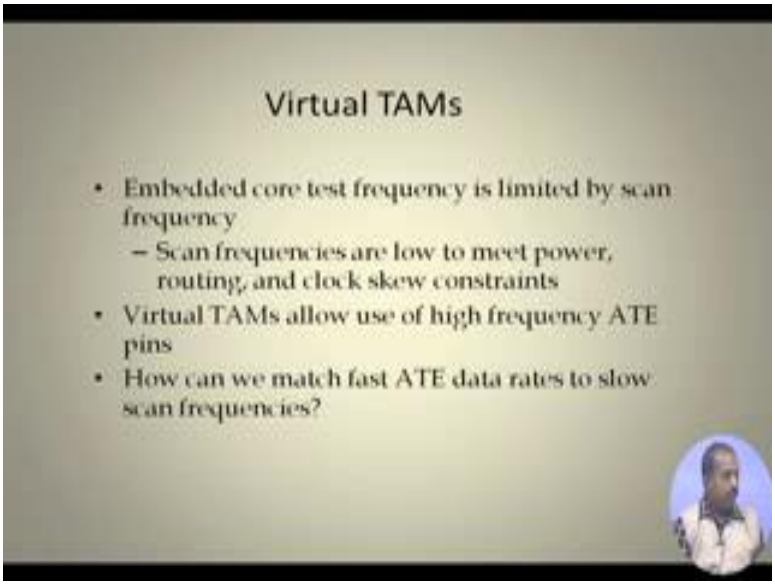
Now after this we look into this the modern modifications in the ATE. So, so current generation ATE is, so they have got many interesting features like; port scalability, so the port can increase the speed of operation then the speed of up to 2.5 GBPS. So, these there are ATEs that can operate at a very high frequency and there is application flexibility. So, application flexibility means I may do some testing for some time then put some idle time, then again do some testing. So, like that there are application flexibility.

So, every port of a tester it may consist of multiple channels and each of those channels can be configured at a particular data range. So, the same ATE it can send data at two different rates. So, if we can send data to a core at a high data rate and the core supports high frequency testing then we can transfer more amount of data to that core but, whereas for a slow core we cannot transfer all those test data to that core because that

core will not be able to utilize those test data. So, then we have to send put them on the on the slower lines.

So, that way we can think about different modifications to the basic test scheduling that we have done using bin packing or using partitioning so that it can take help of this tester features.

(Refer Slide Time: 26:44)



So, the first one in that category is known as virtual TAM. So, virtual test access mechanism. As the name virtual means that the actually the TAM lines are not more, but it gives an illusion that the TAM lines have been increased.

So, embedded core test frequency is limited by scan frequency. So, this is known because these cores scan; so I cannot test at a core at a frequency higher than the scan frequencies suggested by the core vendors you cannot go beyond that. And this scan frequencies are kept low so that this power budget is made and the clock skew is avoided. So, this core is going to be tested at a lower frequency because that is guided by the designer specified scan frequency. And scan testing is normally done at a lower frequency because the scan will create large number of ripples in the chain, so if you do a test do testing at a lower it you can do testing at a lower frequency then this corresponding power consumption will be less.

But this embedded that the ATE may operate at a higher frequency. So, we can use this virtual TAM to allow use of high frequency ATE pins. So, so we will see how can we match fast ATE data rates to slow scan frequencies. So, ATE is transferring data at a very high rate, but the core is doing the testing at a lower frequency. So, in that type of situation how we can improve the test time, so that is actually this virtual TAM philosophy.