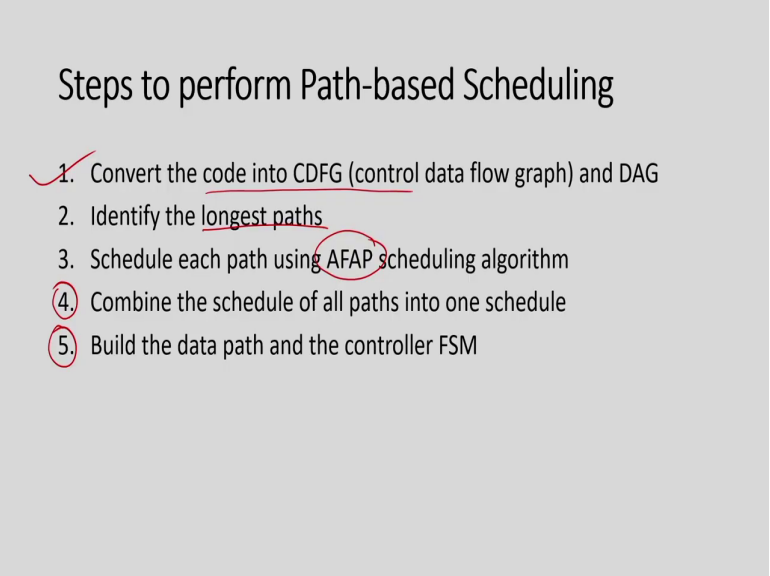
C-Based VLSI Design
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 04
C-Based VLSI Design: Scheduling
Lecture - 15
Path-based Scheduling

Welcome back student, in today's class, we will continue our discussion on Path-based Scheduling.

(Refer Slide Time: 00:49)



So, as you have seen in the last class that, the path based scheduling is completely different approach compared to the normal scheduling approach, where we consider the paths at a time, instead of the basic block. The idea here is that, you try to schedule the operations of a path.

So, we will identify the paths in the behavior and within the paths, we try to schedule the operations within the path. Once we schedule the all paths, we are going to com combine all this path into one and get the final schedule.

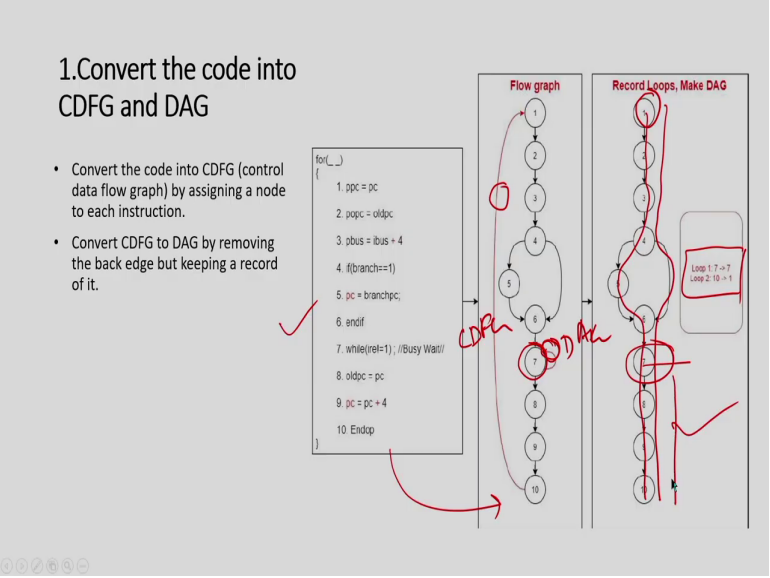
So, the steps of the path based scheduling that we have understood in the last class that, you I actually convert this code into a CDFG and then you identify the longest path; the paths in

the behavior which is termed as longest path and then you schedule each path using AFAP algorithm. So, AFAP is As Fast As possible algorithm that we have already understood.

So, that means, we have already scheduled each paths. So, what is remaining now? It is combining the schedule of all these paths into one. So, that part is still remaining and build the data path and controller of the FSM. So, these two things have not discussed in the previous class. So, we will continue our discussion on this.

So, what I am going to do is that, I am just briefly recap what we have done in the first 3 steps with an example, so that you can repress your memory and then I will go into detail about step 4 and step 5.

(Refer Slide Time: 02:15)



So, as you remember that once you have given a program; you have we will represent them as a CDFG, where each operations becomes a node in the CDFG and the if else everything become a branch and there is a loops and so, this is how we represent the program as a CDFG.

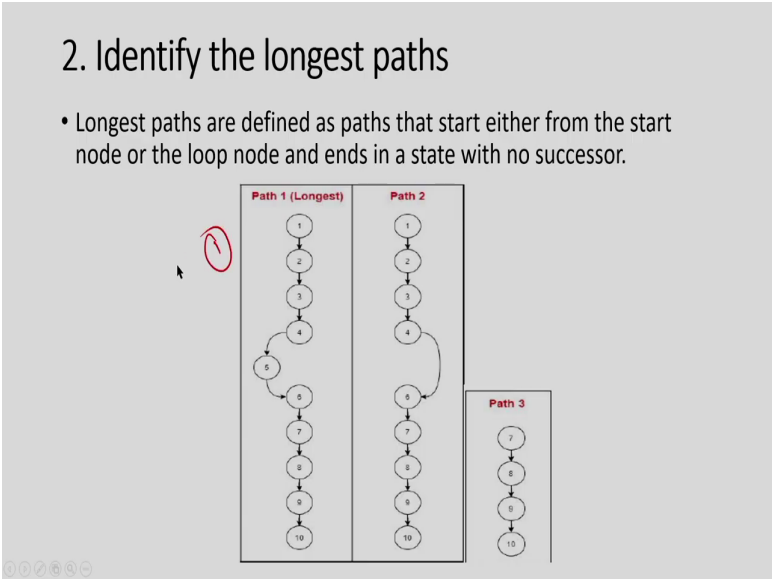
So, this is quite obvious and we have discussed. So, once we have that, so then we convert this CDFG into DAG in the sense that, we remove all this back edge. So, we just remove this back edge and we will keep that information with us, because I need these things in the final generation of the final controller FSM. So, we just remove this back edge and I convert this

CDFG which is a kind of a cyclic graph into an acyclic graph. So, this is the conversion we have done and we have this information.

So, after that what we do is, basically we identify the longest path in this program and the longest path starts from the start state obviously, and it ends in node, where there is no further state. So, there is the state width has no successor. So, if you think about this, so if I start from this; one path will be like this, another path will be like this, there are two paths.

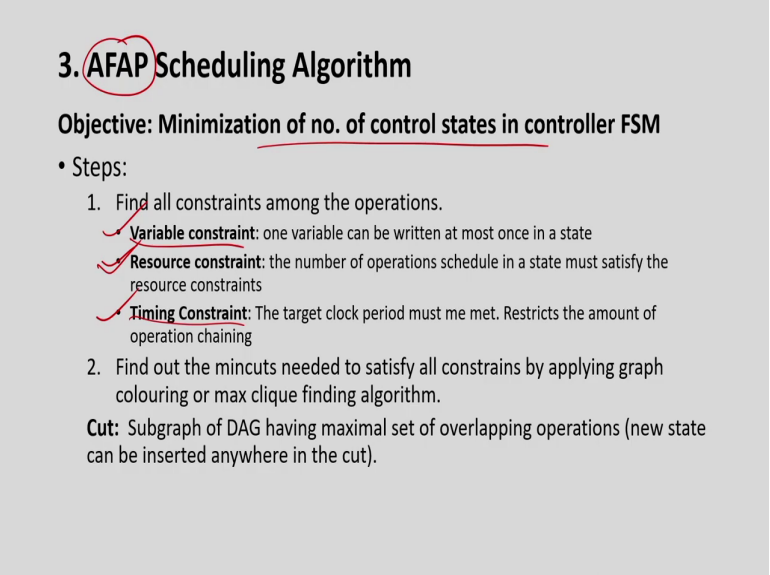
So, you do not know, we do not only consider the paths only from the starting state to end state; we also start path from any loop point, if you remember that we have a back edge here, so we have a loop point here, so I am going to start another path from here.

(Refer Slide Time: 03:49)



So, there are three parts in this particular behavior that we have already identified. So, now, we have these three paths; we will take one path at a time and I am going to schedule it.

(Refer Slide Time: 03:58)



So, the idea of this scheduling is the AFAP algorithm. So, the idea of this algorithm is to minimization of the number of control states. So, there are two steps primarily in this algorithm. So, basically you have the operation in the particular path; what I am going to do? I am going to identify the constraint, what is the kind of constraint the schedules should satisfy in this path.

So, and that we identify, there are basically three type of constraint; the first one is the variable constraint, which says that if there is a variable is redefined, it cannot be done in the same state. So, if the variable defined twice, both the definition should be in two different states.

Then the second constraint says the resource constraint. So, if you have say the number of resource available; so the number of resource used in a state should be must be less than equal to the number of resource available to us. So, that means, I have my schedule must satisfy the resource constraint. So, for example, say suppose I have only one adder; so I can only schedule one operation, one addition operation in one state.

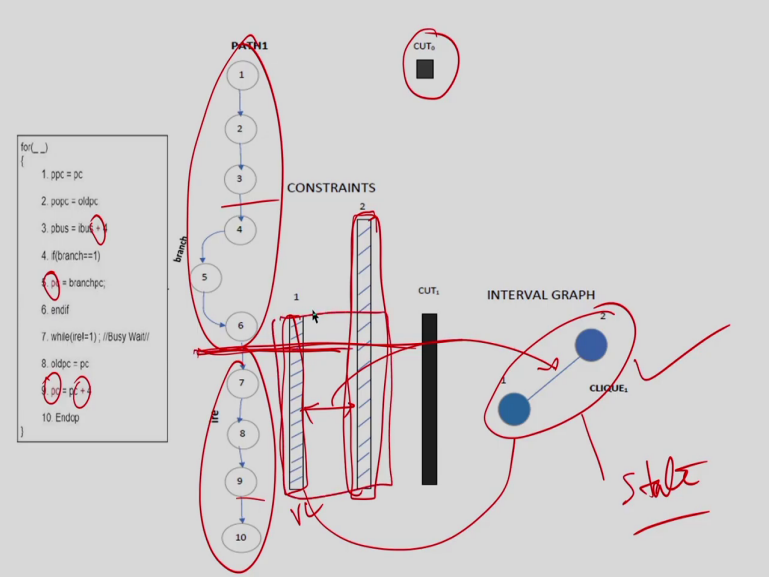
So, if there are two operations, it has to be scheduled into different state. So, that is my resource constraint. And then the timing constraint that I, when we have already discussed in

the previous class that, in the idea of this doing this AFAP algorithm that we actually enable operation chaining.

So; that means, we are performing a chain of operation in a, but single state; but if you just put two operation in a chain, so the delay will become double. So, if there is adder followed by adder, so the total delay of that path is combinational path is two adder delays. So, as we can actually do the chaining as long as it is supported, it does not violate my timing constraint.

So, just to emphasize this example that, if the delay of the adder is say 10 nanosecond and if you just if the target clock is say 25 nanoseconds; so then I can only put at most 2 adder in the chain. So, that is something is a timing constraint. So, we have to remember that, we have to add all possible resource constraints, all possible timing constraints, all possible variable constraints.

(Refer Slide Time: 06:05)



And then what we have done is basically, so if you just put those things; if you take a path and if you put this constraint, so the constraint says that, you have to break this path somewhere between 6 to 9, this is one of the variable constraint, it is coming from this option pc's defining twice. And then we have another resource constraint says that, there are two

addition operations and I have say only one adder to me. So, I have to break somewhere between 3 to 9.

So, there are two constraints. Now, the next problem is that, you have identified all this constraint; so now, you identify the minimum number of cuts that actually satisfy all this constraint. That was the problem to be solved and we have already discussed enough that, basic idea is that you construct an interval graph. In the interval graph each constraint becomes one node. And if there are two constraint overlaps, then there will be an edge in the corresponding interval graph and then in this graph you identify the clique, maximum sub graph.

And one sub graph is one state. So, the basic idea if you go into this example that, you the first constraint says that you cut this path anywhere between 6 to 9; that means you put a cut somewhere between 6 to 9, so that this the first part becomes state 1 and the second part becomes state 2 and the second constraint say that you cut somewhere between 4 to 9 anywhere, you can cut anywhere, that is why this bar talks about.

And say I want to identify only one cut that satisfy both of them. So, suppose I cut here. So, this cut actually satisfy both of them and that is what is identified by this interval graph, where each if there are some set of constraint which is construct a complete sub graph; it means that that particular constraint can be cut by a single cut, can be satisfied by a single cut.

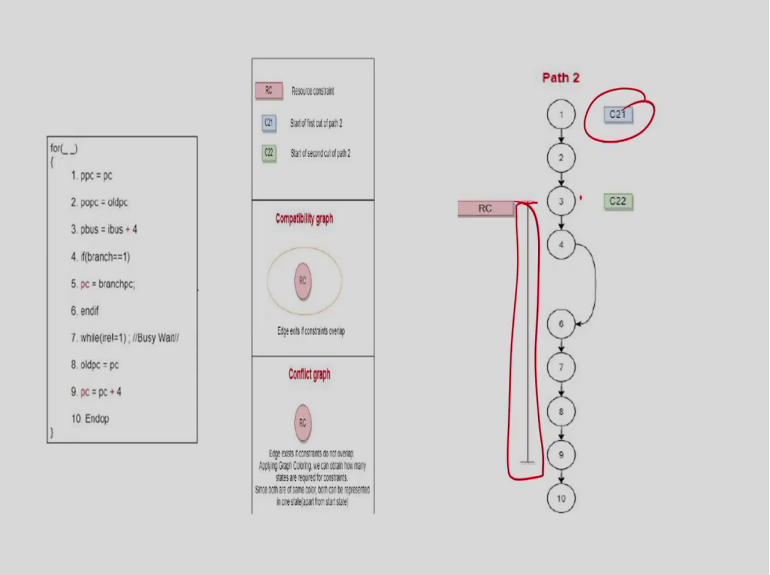
So, this is how I actually I take one path at a time, identify all the constraint and then I construct a conflict graph or interval graph and then I identify the sub graph and each sub graph is corresponding to a one cut and I will identify how many cuts I need to satisfy all the constraint here. So, and each cut represent one state in the final behavior.

So, for this path we have seen that one cut is here, which is basically corresponding to this path and there is another cut which can be done anywhere between this part. So, just to keep the things option open what I do? I do not say that I am going to cut it here, I am going to say that the revised the cut can be anywhere between 6 to 9, because this is the sub common between these two constraint.

So, this is my cut. So, there are two cuts you must cut here; that means my one start one state must start from here and one start one state can start anywhere between 6 to 9. So, that I want to keep option open; because finally I have to combine all this path into one. So, this will help me to identify is there any common part between two cuts of two different paths.

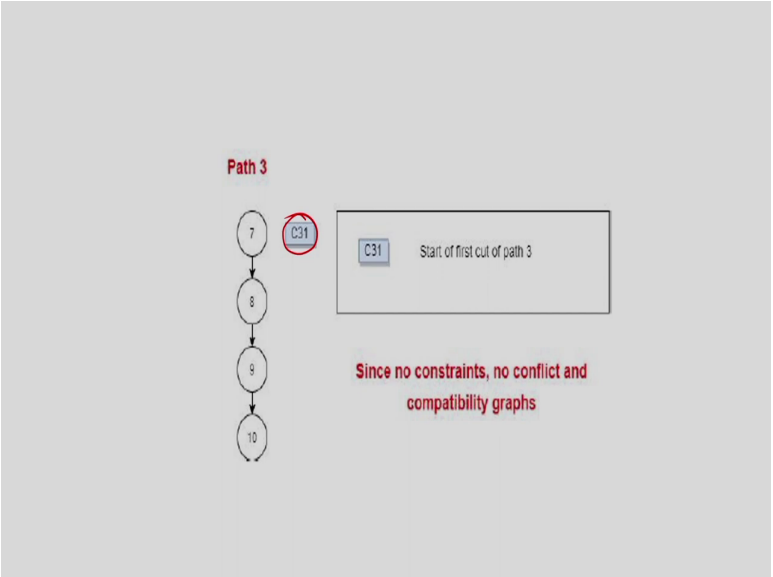
So, this way I am going to apply this AFAP algorithm for all paths. So, that is what I have done; this is for path one and I identify there are two cuts is needed.

(Refer Slide Time: 09:14)



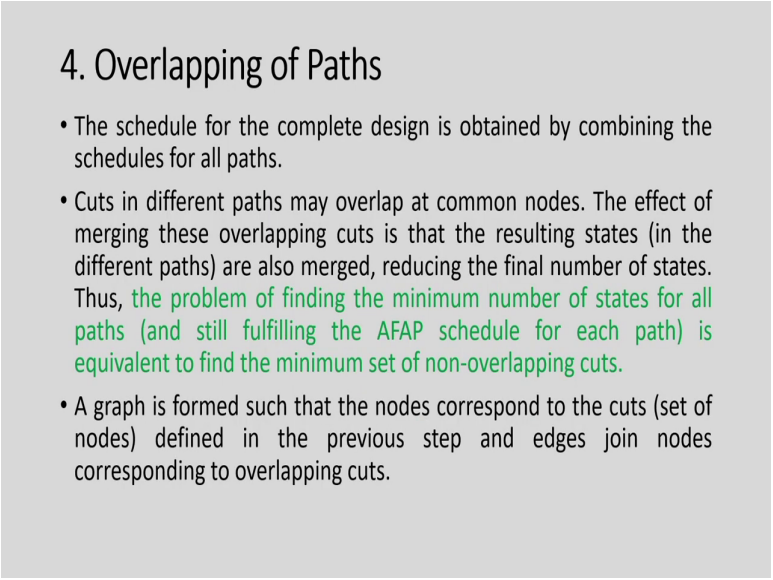
So, if you take the path 2, you also identify there are two cuts needed. So, this is one cut and this is another cut; in this case I have only one constraint.

(Refer Slide Time: 09:22)



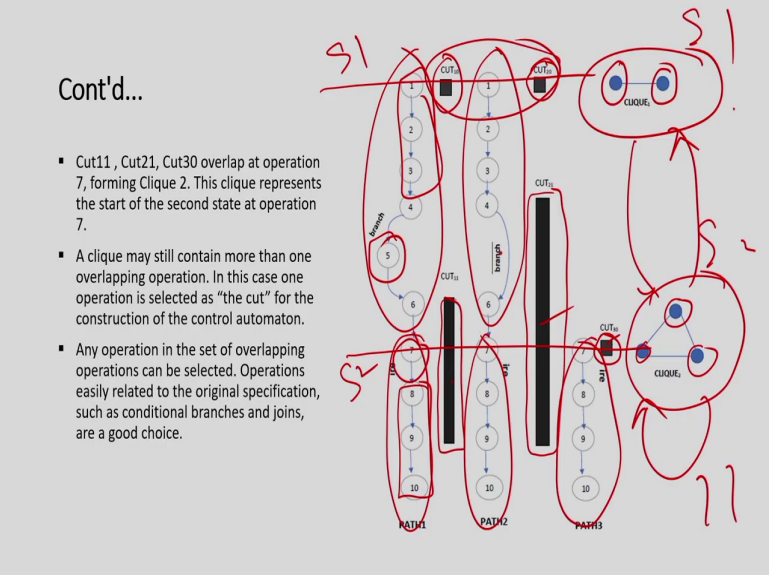
And for the path 3, there is no constraint, so I have only one cut needed. So, that means I have done with 3 paths; because in this particular behavior, I have 3 paths.

(Refer Slide Time: 09:31)



So, this is done and now I have to combine the paths. So, that is also I have discussed in the last class.

(Refer Slide Time: 09:37)



The idea is that again that, I as I discussed that for the path 1, this is the one possible start state and this is the path for places where I can start the next state. For path 2, this is the place where I have to start my state one and this is the place anywhere in between I have to start the next state 2 and this is the path 3 where I have to start the state from here. It is clear that if I just cut it here, it will satisfy all these three constraint and it will say that your first state is nothing, but 1 to 6 and the next part is 7 to 10 and similarly 1 to 6 here and 7 to 10 here and this is this.

So, again I understand that, I have this cuts and I have to solve it with this problem automatically. And to do that what I have the easiest thing is that, I will apply the same logic that; now instead of constraint, I am going to consider this cut as a node in my concrete graph. And then, so I have one node for this, another node for is this, and another node for is this and this is a complete sub graph.

So, it says that this is a complete sub graph; that means all these three constraint can be satisfied by a single cut and this is where I and I have to cut it 7, then only these three will be satisfied. Similarly, another complete sub graph will be created between these two constraints; because they are overlapping and this is another complete sub graph. And it says that you cut here, so you start a state S1 here and you start S2 from here that is all.

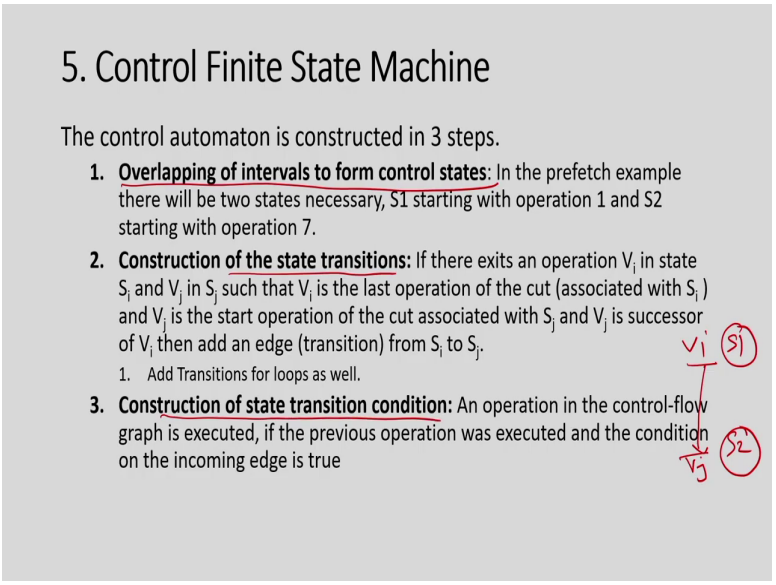
So, this we have discussed tills last class that, given this example, I schedule all the paths; I identify the number of state needed for all paths and then I combine this path using this again, this interval come construction and then I identify exactly the how many states are needed. And you understand that for this example, although there is a while loop, there is if else, there is an outer for loop; but I can schedule the complete behavior in two states, which is not possible if I just do it by basic block by scheduling.

Because in the basic block I am going to take one basic block at a time and there; so this will be on one basic block, this will be another basic block, this will be another basic block and this part will be another basic block. So, there are four, five basic blocks here and just scheduling them, we have it will take more time than two state. So, this is where we get the benefit, if I go by path based scheduling.

So, so far we understood. But, so far what we do not understood that, we only identify the states, there are two states; but what about the transition between the states? So, this is my state 1, this is my state 2 we know that; but what about the transition between the states? So, that we have to identify.

Let us understand now how we can identify the transition between the states and what will be the condition of the states. And once it is done, how this operation will be executed in within the state. So, let us try to understand that part now.

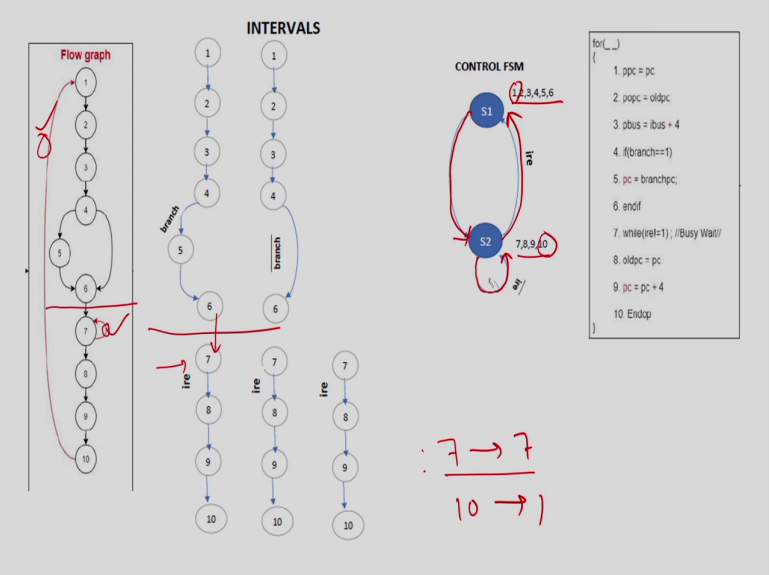
(Refer Slide Time: 12:28)



So, as I mentioned that, I have identified the states; the now the find is that, you how you identify the overlapping of interval to form the control state. So, that is already done, you identify the overlapping and this is my control state 1. And then you, so and similarly this is my control state 2, which is the overlapping of these three cuts.

So, this is done. And now, how will add the state transition? So, let us try to re understand this statement here; it says that, say suppose you have a state S1. So, before that let us understand; so what are the operation to be done in state 1 and what are the operation to be done in state 2?

(Refer Slide Time: 13:13)



So, you can understand here that I cut the path here. So, basically at this point; so that means operations 1 to 6 is going to happen in state 1 and operations 7 to 10 is going to happen in state 2. So, that means in state 1, I am going to do this operation as well as the if else part; in state 2 I am going to perform this while loop as well as the operations following the while loop. So, this part we understood and this is what is already done.

So, next is, how to add the state transition; because I do not know what are the transition to be added and the rule is saying that, say suppose you have a state S1 and S2 and say there is a operation V i here and there is a V j here and say in this program V i is basically the last operation in S1 and V j is the start operation in this S2 and this is basically and this V j is basically successor of V i that is what I have written here.

So, if there exist an operation V i in state say S i and V j in say S j, such that V i is the last operation of this cut; that means in this state and V j is the start operation of this state and V j is a successor of V i. So, that means, in the original code, this V i and V j is the successor of node this. And if you look into this particular example, so 6 is the last operation of state 1, 7 is the start operation of state 2 and this is the successor operation in the actual behavior. So, then I am going to add as transition here.

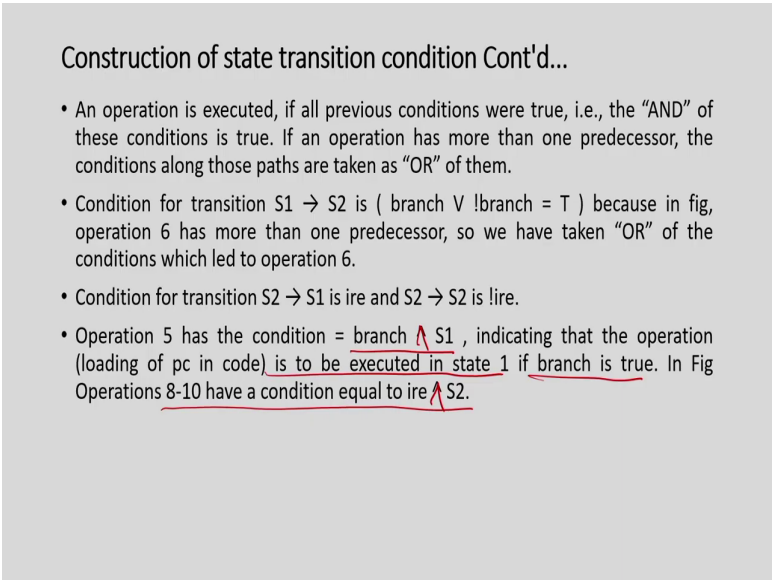
So, this transition comes because of this. And there is no other transition needed here for this; but you remember that I convert this program that, this CDFG into DAG by removing this back edge. So, you have to bring this back edge back these back edges I have removed earlier. So, this and these edges are removed earlier; I need to bring these edges back, because that is the control flow. So, how to do that?

If you just remember here the back edge was 7 to 7, because in this states. So, now, 7 and 7 both are in this state. So, I am going to add a transition here; because 7 was in this state and 7 is also in this both, the loop is from 7 to 7 and both 7 and 7 in this here. So, I am going to add on this and this back edge from 10 to 1, 10 is here and 1 is here. So, I am going to add an edge from S 2 to S 1. So, that is complete my transition.

So, basic idea is that, I will identify the states; I will identify the successor predecessor relation, so is there any operation, which is the just the predecessor operation of no node successor. So, predecessor operation of a operation of some state; then I am going to add an edge and I am going to consider all the back edge that I have removed earlier and bring them back. So, I understood that there are three transitions possible in this controller FSM understood.

So, the next what is that, what is the condition of these transitions that we should understand.

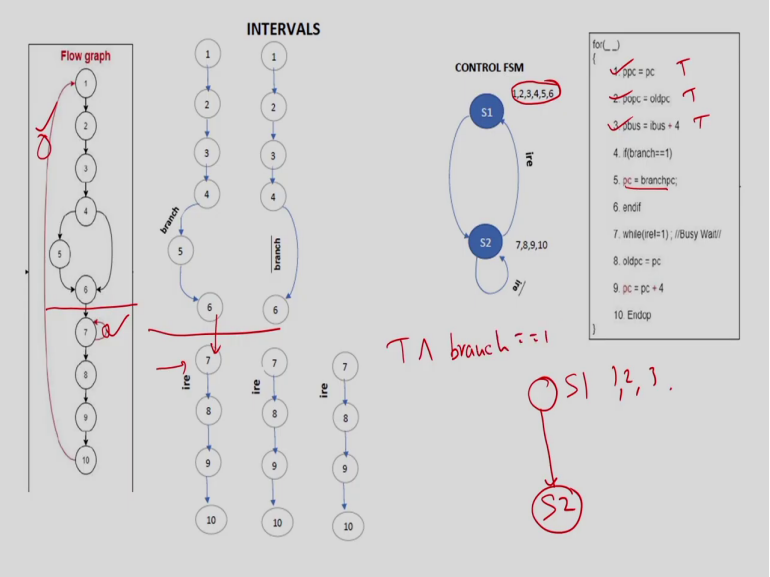
(Refer Slide Time: 16:23)



So, the next state next portion is that, you identify the condition, construct the transition condition and that is very interesting. So, let me explain.

So, the idea here is that, what will be the condition of this. So, when this transition should happen, let us try to understand. So, this is going to happen once I execute all these operations, then only this operation happen.

(Refer Slide Time: 16:47)

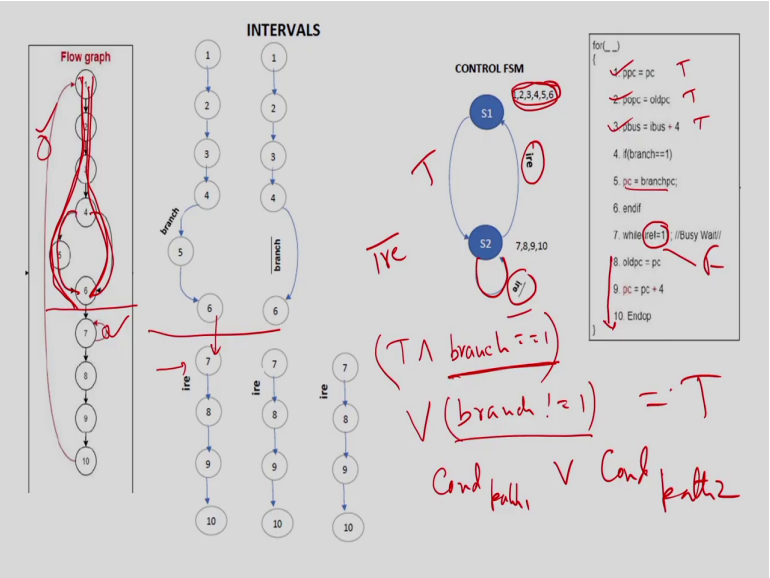


So, suppose I am in this state S1 and I want to go to there is a state S2, I have a transition here. So, I am going to have this transition once these operations are all done. So, if there is a 1, 2, 3, these three operations are done. So, that means the condition should be AND of condition of all these operations. So; that means, the AND of whatever the condition for this operations that should be added.

So, if you look into this program, so operation 1, 2, 3 all are actually the unconditional. So, they will always going to happen; so there is no condition, their condition is true so, true. So, this condition is true, condition is true, condition is true. So, for this operation, operation 5 the condition is branch equal to 1. So, if the branch equal to 1, then this condition should happen.

So; that means, have to have this transition; so condition is true and branch, branch equal to 1. Now, if so, if you now think about this; so this is for the if part. So, this is for the if parts operation 5.

(Refer Slide Time: 17:54)



So, now for this branch, the condition is basically branch bar. So, branch basically not equal to 1, then only you will shape it. Since this if and else both the branch is happening here, so the condition should be OR of this. So, that means if you look into this program; the idea is that you identify the number of paths, there are two paths here.

Two paths are happening in this state; you identify the condition of path 1, path 1 condition should be AND of all the condition of all operations. And for this example I identify the condition of this path 1 is just branch equal to equal to 1, because all other operation has no condition.

And then because you can come to 6 to 7 any of the path is traverse, so I have to take the OR of all path condition; so that means condition of path 1, condition of path 1 OR condition of path 2. So, the basic idea I just to reflect that you I take a path. So, how many paths can execute in this state you identify. There are two paths, for each path you identify the condition and the condition will be the condition of each operation AND of condition of each operation.

So, since here this operation 1, 2, 3 if has no condition, only 5 has a condition; so the condition of this path become branch equal to 1 and the condition for this other path is basically branch not equal to 1 and I have to do OR of this, so it will become true. So, that is why there is no condition here. So, it is always going to execute. So, this is how I am going to identify the condition for each transition.

And if you remember that this condition actually, this transition happened because the loop; so I have to just put the condition of the loop and the condition of the loop is basically this ire bar. So, I just put it here. And once when this going to happen? This is going to happen when this condition becomes false and that is basically ire. So, that I can add this condition.

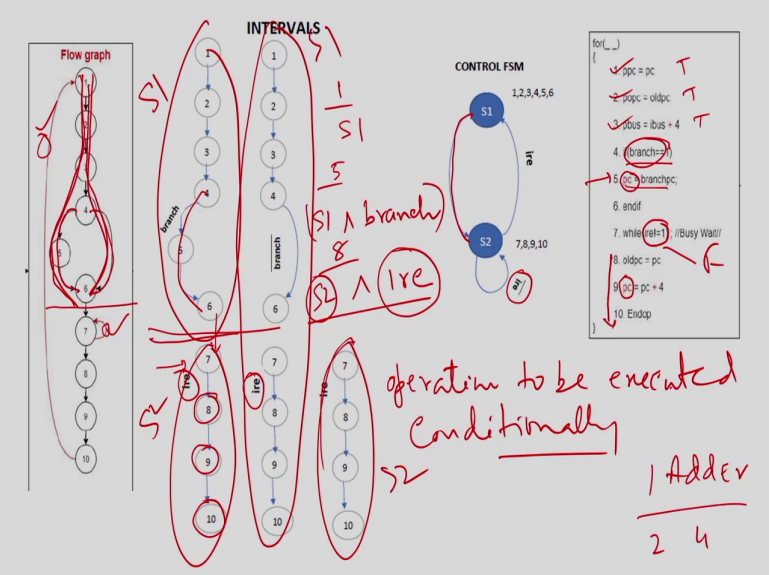
So, this is how I can actually add the condition for each transition. And so, for the loop it is whatever the loop condition and for other transition, you have to identify how many paths can follow through this transition.

And for each path are you identify the condition from the operations and then you make a OR of this condition and then that condition will become the power condition for this transition. So, that is how I will identify the conditions and if you remember for our example, the condition for this is true, for this condition is ire bar and this is ire. So, this is done.

So, what is the next is interesting thing is that, so now, look into this behavior carefully. So, you look into this behavior carefully; how this program will execute by the way? So, you might get confused. So, how this program will effectively execute? Because you see here this 7 is nothing, but a loop.

So, this 7 is nothing, but a loop and you have this 5 which is a conditional statement and then operation 1, 2, 3 is basically is not conditional. So, they are always going to happen; but how do you going to execute 1 to 6 all in state 1. So, the idea here is that, the operations will be conditionally executed.

(Refer Slide Time: 21:27)



So, this is very important statement operation to be executed conditionally. What is this mean? So, that means, so the way I am going to generate the data path is that, this operation is going to be enable only the condition is hold.

So, what will be the condition for operation 1? So, for operation 1, the condition is just is S1. If you are in S1, then in state 1; that means S1, then only this operation is going to happen. So, what about the condition for this operation, this node 5? So, the node 5, because its condition is, it is already condition is branch equal to 1. So, S1 and branch. So, this is the condition under which S5 will work.

So, the way I am going to generate the data path is that, I am going to make sure that although this 1 and 5 both are in S1; so whenever my current state is S1, I am going to execute S operation 1. But in S1, if the condition is branch is true, and then only I am going to execute operation 5, that is the idea.

So, similarly if you look into this operations 8, let me just put it into this. So, when this operation is going to happen? When this loop condition does not hold? So, I am just putting it here like ire. So, ire is the loop condition; so that means if S 2 and ire, then only I am going to execute this. Till I am in 7, I am not going to execute 7, 8 or 10, 8, 9, or 10. So, I am going to do only this operation 1 this condition hold.

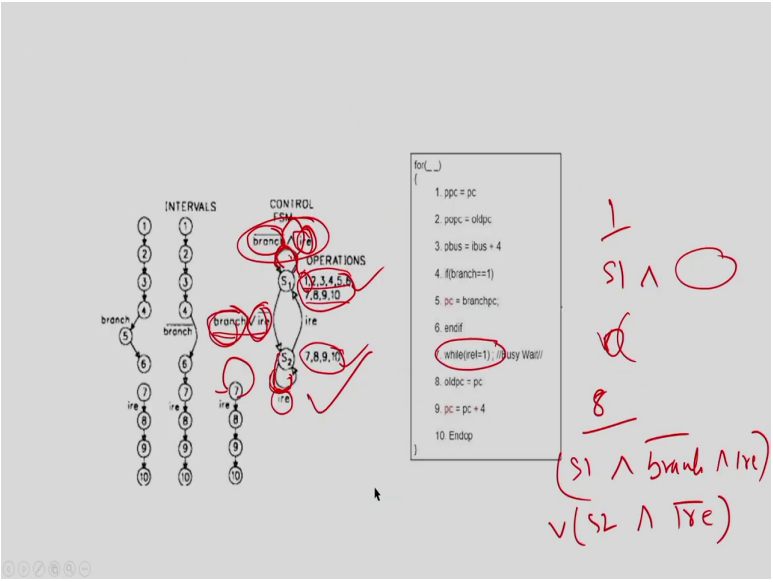
So, that means the condition for operation 8 will be it is S2, the current state is S2; so that means I am not going to execute operation 8 in S1. So, S1, S2 ensures that operation 8 only to execute in state 2 and under further condition is that, I am going to execute operation 8 only when its current state is S2 and also I am my condition of ire is also true.

So, this is how I am going to generate the data path, this is very important. So, for operation 5, it is the condition is this; indicating that the operation is to be executed in state 1, if the branch is true. Similarly for operation 8 to 10, the condition will be ire and S2, this is and.

So, this is how you can understand that, I have generated; this will be this controller FSM and the data path will be conditional data path. So, the conditional data path in the sense that, it is that operations is not going to happen if anything in current state, it might be blocked by certain conditions.

So, you it would only the when the current state is true and the condition is true, then only I am going to execute that particular operation. So, this is the overall path based scheduling algorithm.

(Refer Slide Time: 24:28)



So, I will take one more interesting example before I will wind up this class is. So, in some cases say for example, the example we have taken so far is this graph and say for this resource constraint. So, if in this example I assume that there is 1 adder. So, if you assume

that there are 2 adders. So, then path 2 there is no variable constraint. So, this variable constraint is not there, because this is conditional statement.

So, and since there are 2 adders now available, this resource constraint is also does not hold for this path 2. So, that means for path 2 there is no constraint; so that means I can execute my whole path 2 in a single state. So, if you just apply if AFAP, just assuming that there are 2 adders available; I can execute the whole path 2 in single state. So, operations 1 to 10 in single state.

So, in such cases how can I combine? Because here it live our life becomes simple; because I also assume that I am going to cut my path 1 also at state 6, path 2 also at state 6. But if I just tweak my condition that I have 2 adder now; I am saying that for this path 2 I do not have this resource constraint and hence I can execute my whole operation 1 to 5 in single state.

So, in such case my combined FSM will look like this, this is interesting; you see here, I am actually putting operation 1 to 10 in S1. So, I have stills two state; because I have to break this thing for state 1, for path 1. So, this will be for path 1, this will be my state 1 and this is S2 for path 2 and this is my S1 for this path 2 and this is my S2, because this will happen in S2 for path 3. So, this is the problem. So, there are two state here, there is one state here.

So, basically in this case, one operation can happen in both states. So, that is what I have done here. So, here I just put operations 1 to 10 in state 1 and operations 7 to 10 in state 2 and it is here very interesting, you see here that the condition gets changed. So, earlier that state transition constraint was true. So, because either it will have the branch or not branch; but here I am going to take this branch only when the branch condition is true. So, not branch is, so that else path is not there.

So, that is why I see here, I just put this condition now branch or ire and I will talk about talk about this. So, because now if you just execute this program. So, there are two paths. So, basically you if you just execute this; then the condition will become ire, because you traverse this path and then if you just traverse this way, it will be branch. So, that is why I just put the condition is that, if this condition holds and this is ire bar; because this is something where this loop will execute. So, you can understand this.

And then here you can see here that, I am putting a transition here also; because here the loop will come, there is a back edge. So, this back edge will come here as well and here as well and the condition of this branch is this. So, if I the loop do not execute and the false condition is hold; then I am going to be in this state and I am going to continue this all operations. If condition is hold and this loop is true; then I am going to come here and then I am going to do the loops here.

In this case, the loop does not occur; so because this ire is true. So, this is how I can actually construct that transition condition. So, this is more complicated scenario, but just removing one resource constraint; I can I just try to highlight that this operation can repeat and the condition of this operation to be different, if we just repeat there. So, the operation 1 will execute now if S1 is true and this condition is hold or that is alright this is for operation 1.

For operation 8, it can execute when this you are in S1 and this branch bar and ire bar or you are in S2 and I sorry this is ire and then it is ire bar. So, here this loop means that, you are not going into this while loop in this case; but this loop is going to actually happen here, because you see that this path is actually put into S2

I hope you understand this complication of adding this conditions and transitions into this controller FSM; because it all depends on how the individual path is being executed. So, and then you have to also make your data path such a way that, this operation can only execute conditionally, it is not that is going to happen unconditionally.

So, obviously this path based scheduling is giving you lot of benefit in terms of latency can scheduling only two time step; but it comes with a cost, the cost is something its controller FSM is really complex and the data path is also becoming really complex, because the operation is not basically now unconditioned, it is basically it is getting going to execute conditionally.

But this is completely very useful in the kind of processor level code, where because you can see here that for different instruction, you are going to do different thing.

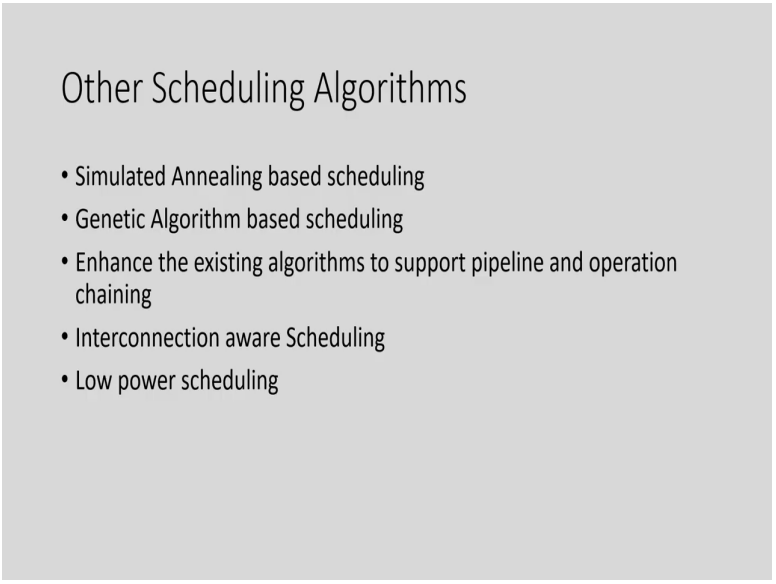
So, obviously in the processor, if you try to automate, try to get a RTL from a processor behavior; so here, so there are many paths where those paths are basically mutually exclusive

and the operations are basically not so overlapping. So, in that kind of cases, this path based scheduling is really useful; because your number of control step will be very less to schedule this behavior.

So, this is very interesting things I have covered. So, with this, we are basically come to end of this discussion on scheduling; but to be honest, because the scheduling is NP complete problem, there are many scheduling algorithm and I can take one whole semester course just on this particular scheduling algorithms.

So, there are many other very interesting and important scheduling algorithms are there; just like simulated annealing based scheduling, genetic algorithm based scheduling. So, these are all evolutionary algorithm and you can actually apply to in the scheduling problem; because this is something a NP complete problem.

(Refer Slide Time: 31:13)



And there are some other new techniques like low power scheduling or say interconnection aware scheduling. So, in which you try to schedule the behavior making sure that, your power consumption will be minimum or say the way you are going to schedule, you are making sure that your inter connection cost will be less. So, this are a kind of way forward approach and also you can apply this genetic algorithm to do the scheduling, where you actually create some initial solutions and then you do crossover mutation and then you calculate the cost of

the new upspring and then you decide which one to keep, which one to discard and you keep doing it and then finally, you end up having a good algorithm.

Similarly, like simulated annealing, where you actually have one solution; you just do some kind of perturbation, you create some new solutions and you keep the good solution and also some random bad solutions, so that finally your optimizer will reach to some global optimum points.

So, I hope you understand that and if you are interested, you can actually go through of them. And also there are one important aspect I actually have not covered in the scheduling discussion is like, we have already talked about the pipelining or operation chaining is something very important; but I have not talked about that during ILP formulation or list based scheduling or force directed scheduling, I have not talked about how to enhance those algorithm to support pipelining or operation chaining.

So, again that is if you are interested, you can actually go to the many research papers; you can go into that and you can learn them. But due to the completeness of this course, I just skip those important things and I keep them for future study for you.

So, with this I conclude discussion on scheduling; from next class onwards, I am going to discuss about the allocation and binding phase of high level synthesis.

Thank you.