C-Based VLSI Design

Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

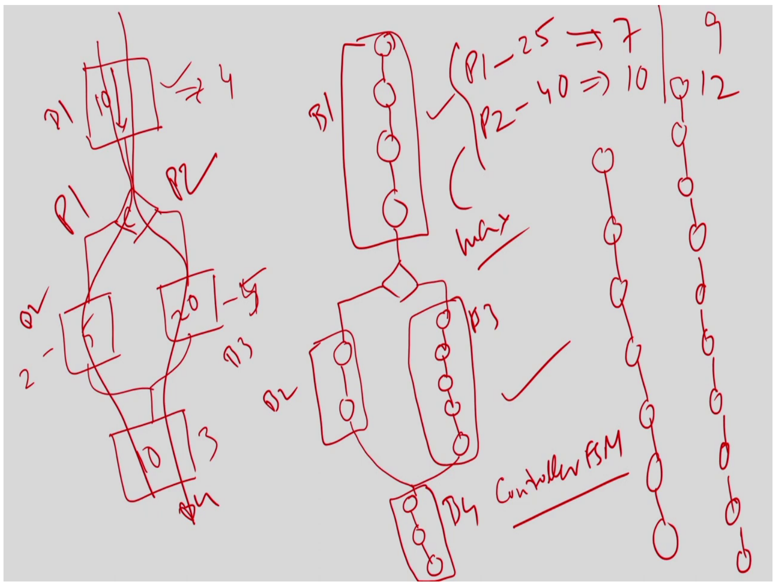
Module - 04
C-Based VLSI Design: Scheduling
Lecture - 14
Path-based Scheduling

Welcome everyone. In today's class, we are going to discuss about Path-based Scheduling. So, if you remember we have, in the start of this scheduling we have discussed that we are going to take the program, because I have given a program, but what we do we did is actually break that program into basic-blocks.

And whatever the scheduling algorithm I have discussed so far, it is basically scheduling of that basic-block, one basic-block at a time. That means, we are not considering the control structure of the behavior, but taking only the basic-block of operations where there is a sequence of sub processor are there. And then, we try to identify the minimum number of time step or minimum number of resource under resource timing constraint, those kind of problem we have solved.

And finally, we will get the schedule for each basic-block. And then, how the final behavior will look like? It is combining this basic-block into one.

(Refer Slide Time: 01:40)



So, just to give an example, suppose I have one program like this, so you have a say there is a basic-block here, then there is a condition here. So, this is the if-else. So, and then you have something here. So, the way it is now discuss is that if there are say 10 operations here.

So, there are 5 operations here and there is 20 operations here and there are 10 operations here. So, these are the number of operations are there. And what we have done so far is that I take this basic-block at a time and say I have schedule in say 4 time step. And then, I schedule this into say as a 2 time step, this is in 4 time step, or say 5 time step, and this is in say 3 time step. So, and how do you combine this? It is basically for this schedule is like this.

So, there are 4 time step. So, this is my basic-block 1, then I have a conditions. So, here I have 2 states and I have 5 states here. So, then I will combine this into this, and then this will be my 3 states for this. So, this is say basic-block 1, B 1, this is basic-block 2, this is basic-block 3 and basic-block 4. So, this is B 1, this is my B 2, this is my B 3 and this is B 4.

So, I did all this basic basic-block. Basic-block means sequence of operation, there is no control flow there. So, I take one basic-block at a time. I do a base scheduling to solve this problem, and then I combine each basic-block into this control structure, and this is the final controller FSM.

So, this is my controller FSM. So, this is what we have done. And then, we know what are the operation is happening in every state and based on that we will construct the data path. That is what the way we have discussed so far.

Now, here you try to understand; obviously, one of the drawbacks here is like I am, so I have come up with very efficient algorithm very optimal algorithm, but it cannot solve the it only applicable to a one basic-block. It does not; you cannot merge this scheduling with this with this that because they are taking it independently.

So, two basic-block scheduling cannot be combined together, because I have not taken them together. As a result whatever the optimization you do here the final latency of the design is not something optimized across basic-block. Its optimized within basic-block its not optimized across basic-blocks.

So, in the path-based scheduling it tried to solve that problem. So, it tried to achieve minimum latency by try to share the operation across basic-block. So, how it takes the code? So, instead of taking this basic-block what it does now it takes the path. So, how many paths are there? You have two paths. So, this is one path. So, this is my path 1 and this is another path. So, there are two paths. Obviously, you can ask me how what will happen to the loop. I will come to that point.

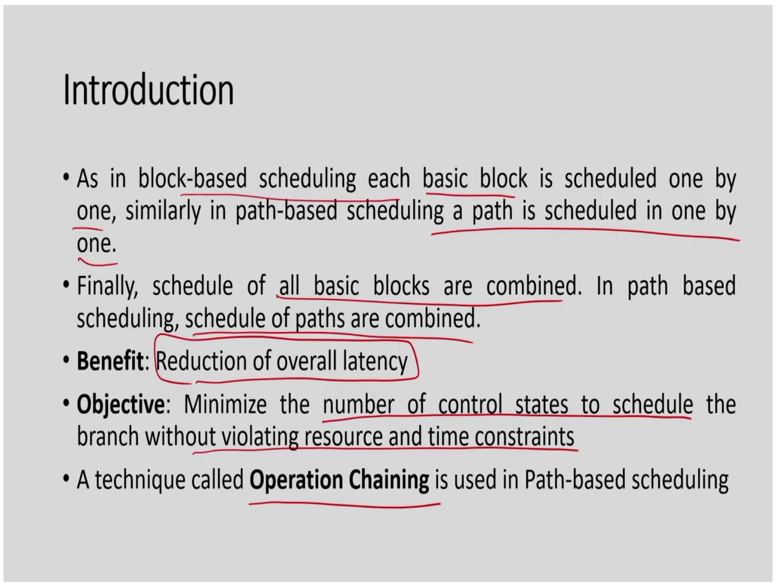
So, there are two paths now. So, you now think about that I am going to take the path P 1. And how many operations are there? 10 plus 5 plus 10 equals to 25. So, now, there are 25 operations in path 1, and in path 2, I have 10 plus 20 plus 10 equals 40 operations. So, now what we can do is I am going to schedule path 1 independently.

Now, if you try to find out the minimum number of time step needed for path 1; obviously, since I can have a sharing of time step between operations of two basic-blocks because if they are not independent I can actually schedule this probably and say 7 time step. And then, and this and say this particular time step is basically say its scheduling 10 time step. So, that something is understood.

And now you think about that now it is 10 time step. So, earlier you think about that it is 4 plus 2 plus 3, so 6 plus 9 time step. If you take this path, it is basically taking say 9 time step, and this was taking say 4 plus 5 plus 3, 12 time step. Now, it is taking only 7 and 10, which is less than the actual time step it was taking on the basic-block.

So, the basic point here I try to convey is that if you take the paths at a time, since there are some operation which is basically there is no dependency on operations because paralyzed these operations, we can schedule them in 10 time step. So, the number of time step needed to schedule this behavior will be much lesser than the number of time step needed if you go by basic-block by scheduling. So, the path-based scheduling is actually try to do that thing.

(Refer Slide Time: 06:56)



Introduction

- As in block-based scheduling each basic block is scheduled one by one, similarly in path-based scheduling a path is scheduled in one by one.
- Finally, schedule of all basic blocks are combined. In path based scheduling, schedule of paths are combined.
- **Benefit**: Reduction of overall latency
- **Objective**: Minimize the number of control states to schedule the branch without violating resource and time constraints
- A technique called **Operation Chaining** is used in Path-based scheduling

So, as I mentioned here the basic-block based scheduling, schedule the each basic-block one by one. In path-based scheduling, we schedule the path one by one. So, that is understood that I will identify the paths and I want to schedule the paths one by one.

And then, in basic-block based scheduling what we have done? We basically combined this schedule of all basic-block together. So, combine means is basically just creating this FSM that I have already explained. So, similarly in path-based scheduling, I have to combine the schedule of two paths. So, for example, here you see I have mentioned that there are 7 states here.

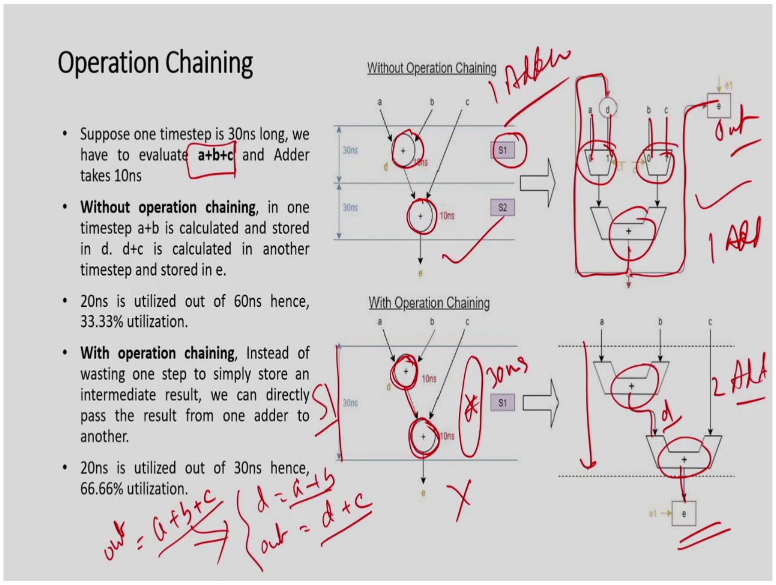
Say probably it has 7 states and in path 2 I have 10 states. So, obviously, the number of states together will be 10 because it will be max of this. So, for path-based scheduling, similarly I have every path, we have found the minimum of schedule for each path, we have to combine them.

So, how to combine these two paths together? That something we are going to discuss in this class. But the basic approach is similar to the basic-block that while we are taking one basic-block at a time schedule, and then, you combine the results here, you identify the paths, schedule the paths independently, and then, combine the schedule of the paths. That is what the path-based scheduling algorithm.

So, the overall objective of this path-based scheduling is that reduction of the latency. So, whatever the latency can be achieved through this basic-block based scheduling will be much higher than this path-based scheduling, because in the path-based scheduling I have an opportunity to share time step across basic-blocks.

So, the primary objective of this path-based scheduling is to minimization of the number of control states, to schedule the branches without violating the resource and timing constraints. So, that is the objective I understood that I want to schedule the whole behavior in minimum number of time step.   So, without violating the resource, timing and other constraints.

(Refer Slide Time: 09:20)



Operation Chaining

- Suppose one timestep is 30ns long, we have to evaluate $\boxed{\textbf{a+b+c}}$ and Adder takes 10ns
- **Without operation chaining**, in one timestep a+b is calculated and stored in d. d+c is calculated in another timestep and stored in e.
- 20ns is utilized out of 60ns hence, 33.33% utilization.
- **With operation chaining**, Instead of wasting one step to simply store an intermediate result, we can directly pass the result from one adder to another.
- 20ns is utilized out of 30ns hence, 66.66% utilization.

$$out = a+b+c \Rightarrow \begin{cases} d = a+b \\ out = d+c \end{cases}$$

So, for path-based scheduling one important factor is operation chaining. So, that is something is quite used in path-based scheduling. So, let me explain that operation chaining.

So, far suppose you have an expression a plus b plus c. So, the way we have discussed earlier that I will do say d equal to, I will break it into 3 address form.

So, the way I will do is basically I will do a equal to a plus b, and then, say out equal to d plus c. So, this is how I just break a plus b plus c. So, say out equal to a plus b plus c is there. So, I can break into this form.

So, this I have argued earlier that I will go to do it because I want to do this operation, I individual operation I want to schedule. If you try to schedule everything together my delay will be more. But since this operation this path-based scheduling my objective is to schedule everything in minimum number of time step, so here probably if my timing permits probably I am going to do this a plus b plus c in the one clock. So, that means, I have to chain 2 adder.

So, that is what is called operation chaining, so this is there because if say you already know that things that if there is a multiplier which is say a it is much complex operation than a adder operation. So, if you try to combine this you put schedule on a multiple adder and 1 adder in the same time step some of the time this adder will remain idle.

So, you can instead of doing one addition one time you can actually perform two addition operations, in chain in the same time. That is what is called operation chaining. So, let me explain in the context of a plus b plus c.

So, as I mentioned that in normal whatever you have discussed that I am going to do this a plus b in time step 1, and then, in a plus b plus c. And then, the d plus c in time step 2. So, this is two time step to perform this operation. But if you do this in two time step I need only 1 adder. I need only 1 adder. I can actually share the adder between these two operations because they are actually happening in two different time step, and this will be the architecture.

So, basically I need a mark at the input of the adder. So, this is my adder. And at the input I need a mark, so in the first time step, I am going to do a plus b, in the next time step, I am going to do d plus c. I am going to do d plus c. So, I am going to multiplex this input, and then, output will one time it will go to d and next time it will go to e or out. So, that is what is going to happen in normal scenario.

In operation chaining, say as I mentioned there may be say some another big multiplier schedule here which is say 30 millisecond, say 30 nanosecond and this adder is only 10 nanosecond. So, I can easily do two operations at the same clock, because otherwise the adder will remain idle for 20 nanoseconds. So, here I am not performing this two addition operation in two cycles, rather I am doing the whole thing in same cycle S 1.

Then, what I am doing here? So, I am going to; I am going to put a adder here and the output of adder is not going to any register, it is going to the another adder and this is going to happen in the same clock. The architecture you understand here that I have a plus b adder here, and then, the result is coming here and I am just doing this I am adding with c. And then, I am directly I am going to update e. So, I do not need to store the intermediate result, d into some register. So, this is something called operation chaining.

And operation chaining is nothing new and it is nothing related to only path-based scheduling. You can actually do the same thing in normal scheduling as well. But so far the way we have discussed the scheduling, we did not consider operation chaining so far. But in path-based scheduling this is very obvious because our objective is to schedule everything in minimum number of time step.

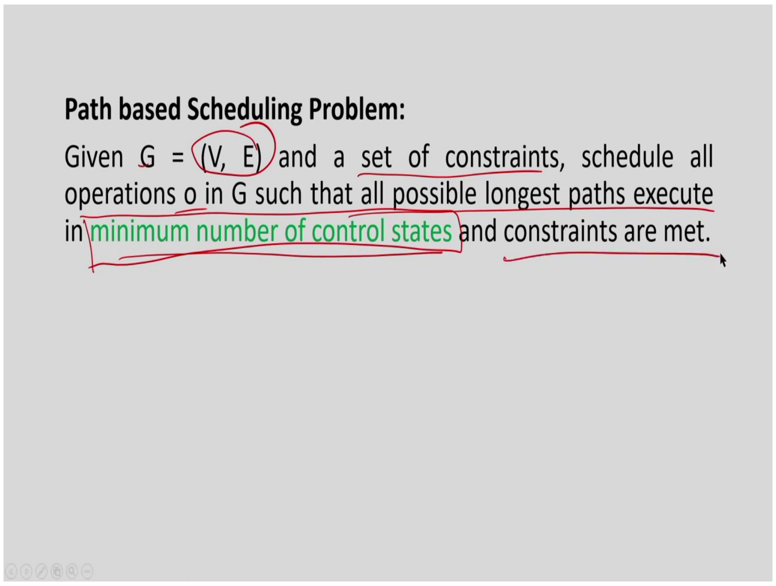
So, if the timing satisfied, if the say the target clock is 30 nanoseconds and there are 2 adders in chain, this path-based scheduler always try will schedule them in series, so that they can be schedule in the same clock. Since, the objective of this path-based scheduling is to identify the minimum number of time step, so it does not look into the resource overhead.

Because here you can see if I schedule these two operations in two different time steps, and in only 1 adder, here I need 2 adders because they are going to happen in the same clock. So, I need 2 adders. But because this path-based scheduling does not look into that aspect it will do that. But in other sense, if you say that I have only 1 adder, it will not do this. It again, it will give these results.

So, its not that I mean to precisely state that it depends on your resource constants. So, if you say that I have only 1 adder it will not generate this resource, but it is going to generate this particular schedule, where I can schedule this two operations in two time steps and hence I can share that.

So, but in general if timing constraint satisfy resource constraint satisfy, it try to schedule as many as operations in chain as possible without violating the resource and timing constraint. So, that something is operation chaining and which is quite useful in path-based scheduling, as I argued that here objective is to minimization latency without violating my constraint.

(Refer Slide Time: 14:32)



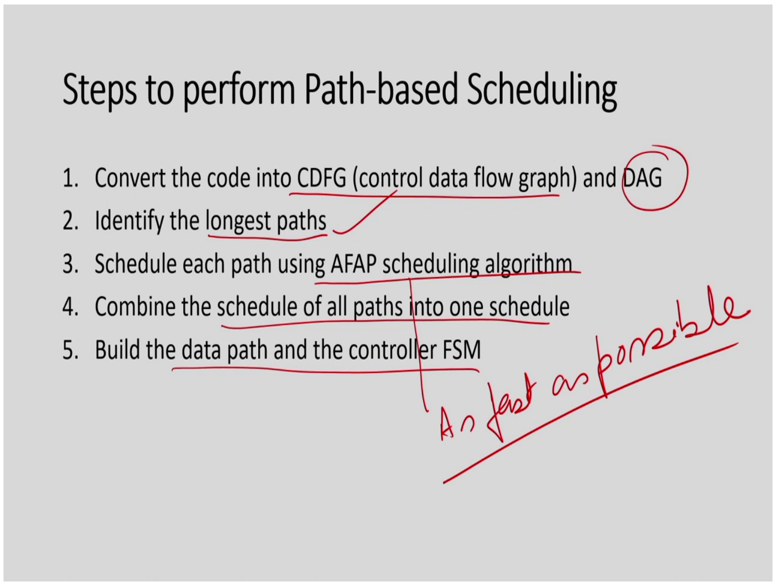
Path based Scheduling Problem:

Given G = (V, E) and a set of constraints, schedule all operations o in G such that all possible longest paths execute in minimum number of control states and constraints are met.

So, now if I just define this path-based scheduling problem. So, I have given this graph my overall program, and I have given a set of constraint. So, I am going to talk about what are the constraint, as I mentioned the resource constraint, timing constraint, all, everything we will give together.

And then I am going to schedule this operation all the operations on in G, such that all possible longest path I am going to talk about that. So, you can think about the paths can be schedule in minimum number of control states. So, this is very important. Here I want to schedule everything in minimum number of control states and without violating the constraint. So, this is my path-based scheduling problem.

(Refer Slide Time: 15:11)



Steps to perform Path-based Scheduling

1. Convert the code into CDFG (control data flow graph) and DAG
2. Identify the longest paths
3. Schedule each path using AFAP scheduling algorithm
4. Combine the schedule of all paths into one schedule
5. Build the data path and the controller FSM

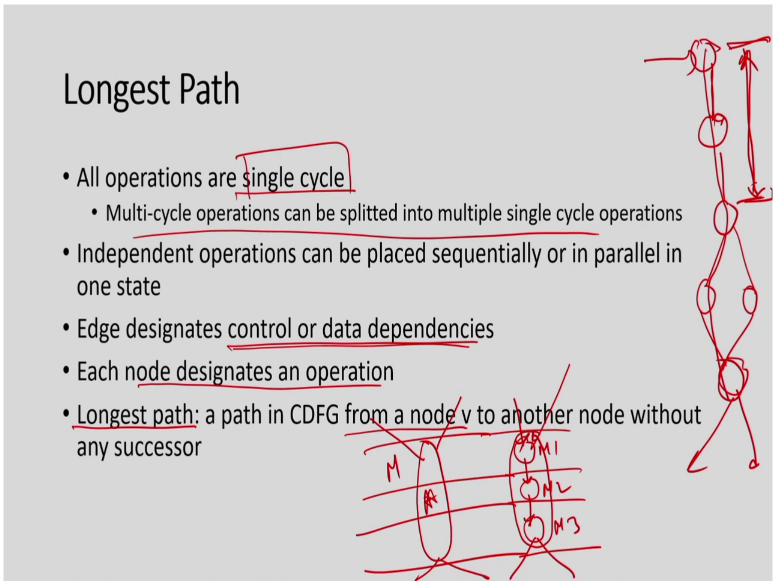
As fast as possible

So, what are the steps to be performed in path-based scheduling? So, obviously, you have given a program you have to represent them as a control and data flow graph that we already understood and the dependency probably using the DAG within the basic-block. And of as I mentioned here, I have to schedule the paths one at a time. So, I have to identify the paths.

And these paths called longest path. I will talk about what is longest path. So, first thing is that you identify the paths possible paths in the program. And then, you basically take each path and you schedule it. So, here it is schedule using AFAP algorithm. So, this is not ASAP or ALAP, it is AFAP, As Fast As Possible. As our target is to execute as fast as possible, so I operation chaining is very important in this context.

So, this is a new algorithm I am going to talk about that what is AFAP algorithm, it is not ASAP or ALAP, it is AFAP. And then, you combine the schedule of all paths and into one schedule. And then, you build the data path and controller FSM. So, these are the 5 steps to be performed in the path-based scheduling algorithm. So, let us try to go into each step at a time.

(Refer Slide Time: 16:29)



Longest Path

- All operations are single cycle
  - Multi-cycle operations can be splitted into multiple single cycle operations
- Independent operations can be placed sequentially or in parallel in one state
- Edge designates control or data dependencies
- Each node designates an operation
- Longest path: a path in CDFG from a node v to another node without any successor

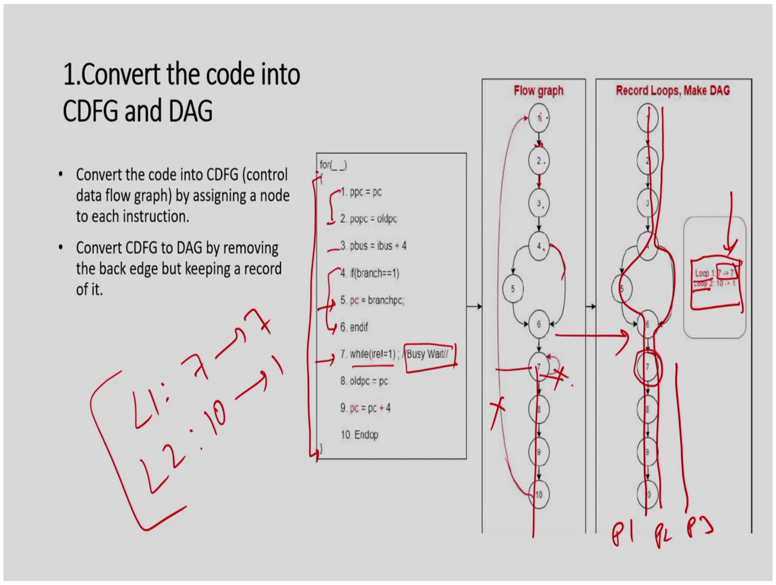
So, the first problem is that you basically have the program, you represent them using paths. So, how to do that? What we do we actually construct a special kind of CDFG here. So, you have the programs. And then, what you do? You basically you have every operation you going to represent by number of states. Every operation is a state now and whenever there is a control flow or data flow you are going to add an edge. So, if there is a if-else, so it is going to be edge like this. So, this is something I am going to do that.

So, I have each node is designates an operation and control, this edge designates the control or data dependencies. Since, this operations are actually you can see here I am actually putting one operation at a time, so the data dependency already taken care. So, this edge is basically saying there is a kind of data dependency or control dependencies.

So, now we actually assume here all operations are single cycle, but again you can actually break a multi cycle operation into multiple single cycle operations. So, for example, to give this idea that if say there is a multiplier which is a 3 cycle operation, I can actually in my graph, I can just represent this multiplier as like this say I have. So, this is my multiplier.

So, I can just say this is nothing, but 3 small operations in chain, so M 1, M 2, and M 3 and there will not be any other edge in between. So, this is just; so, if there are say two inputs are coming two inputs will come here and if it is going to two other output it will go like this. So, I can just represent this multiplier using 3 small multiplier of single cycle. So, I can handle multi cycle operation, but just transferring a multi cycle operation into those many single cycle operations.

(Refer Slide Time: 18:22)



1.Convert the code into CDFG and DAG

- Convert the code into CDFG (control data flow graph) by assigning a node to each instruction.
- Convert CDFG to DAG by removing the back edge but keeping a record of it.

```
for(__)
{
  1. ppc = pc
  2. popc = oldpc
  3. pbus = ibus + 4
  4. if(branch==1)
  5. pc = branchpc;
  6. endif
  7. while(irel=1) ; //Busy Wait//
  8. oldpc = pc
  9. pc = pc + 4
  10. Endop
}
```

So, basically just to explain this, so suppose I have given this behavior there is a for loop here out outer loop and these are the operations. So, and there is a if-else you can see here there is a if-else, and there is a while loop here. So, how to represent this as a CDFG here?

So, I am going to put one operation. So, there are how many operations are there? 10 operations. So, I am going to put each operation as each node. So, this is operation 1, this is operation 2, 3, 4, 5, 6, 7, and so, for each line or each line there is a node here. So, that is done.

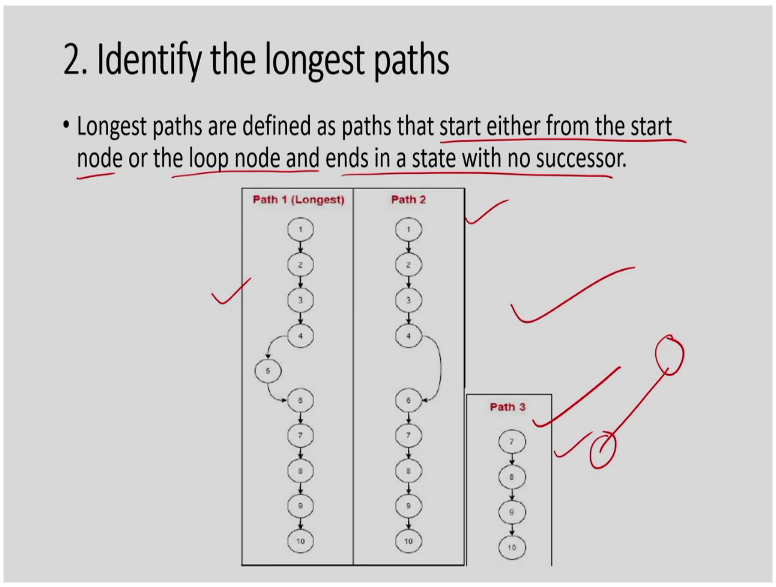
And then, since there is a control flow here because after this operation 1, this operation 2 is to be executed I am going to add an edge here. Similarly, I am going to add an edge here. And so, after this node 3 there is a if-else. So, I am going to add a if-else here and this is my the basic-block corresponding to this is 5 and this will come to here. You can understand that and this while loop there is no body, the it is basically while one, it is busy wait. So, I just have a loop here.

And then, this 7, 8, 9, 10, this operations will be there and there is a back edge. So, because this is of this for loop. So, this is what the CDFG representation of the programs. So, that we have done first. This is my step one. You convert the code into CDFG and data flow graph. So, that is something is done here.

So, what is the longest path? The path in the CDFG from a node v to another node without any successor. So, this is very important that it is, there will be many paths. So, you can think about that this is also a path. So, this is also a path. But I am not going to consider this kind of path, I am going to consider only path which start from some node it can be from any node, but it will end to some node where there is no other further transitions.

So, that means, I cannot take, I am not going to take any path of this length and it is sub path. These are the smaller path. But the longest path is those path will start from any node. Its not that it has to start from the start node, it can start from any other node. I am I am going to come into that from where you it can start. And then, it will always end to some node where further there is no successor. This is the kind of last state in your program. So, those are the longest path.

(Refer Slide Time: 20:50)



2. Identify the longest paths

- Longest paths are defined as paths that start either from the start node or the loop node and ends in a state with no successor.

Now, the question is that the first thing is you identify this program you convert into this and now the question is that how to identify the longest path. What is the longest path? We have to identify.

So, here you have to understand that if for if-else its fine, but there is a if there is a loop, so that loop is going to iterate many times. So, the important point here is that the longest path obviously, it starts from the end state and that will going to end in the end state. But it I should also consider path that is starting from a loop point, loop is loop starting point, because if I do not consider such paths then the repetition of the loop body will not be will be lost.

So, we need to consider two types of paths, the paths that are starting from the start node, and ending some node where there is no successor node and also the paths that are starting from the loop starting point. So, there are two types of longest path here. So, the paths either start from the start node, or the loop node, and ends at the state which has no successor. So, these are the path I am going to consider, longest path are the defined as the paths that are starting from the start node and the or on the loop node.

So, before that one preprocessing to be done here. So, because this loop can create infinite number of paths, so what I am going to do? I am going to remove these loops. The back edge, it is not I am going to remove the loops rather the back edge. And I will keep that information handy for me for future use. So, there are two loops here. So, there is a one loop is 7 to 7.

So, I am going to just keep them my loop 1 is basically 7 to 7. So, I will just put the back edge 7 to 7. And there is a loop another there are two there are actually two loops. So, this is one loops and this is another loops. So, there are two loops here, but for both the loops this is the back edge. So, if I just remove this edge, I can remove the loops. So, there is another loop for which there is a back edge from 10 to 1. So, this is what I am going to do.

So, what I am going to do? I will just keep two information my have loop 1, for that back edge is 7 to 7 and there is a loop 2 for is the back edge from 10 to 1. So, I am going to remove this edge and this edge. So, I am going to remove this edge, and my revise CDFG looks like this. So, this is done to avoid having infinite number of infinite number of paths, longest path.

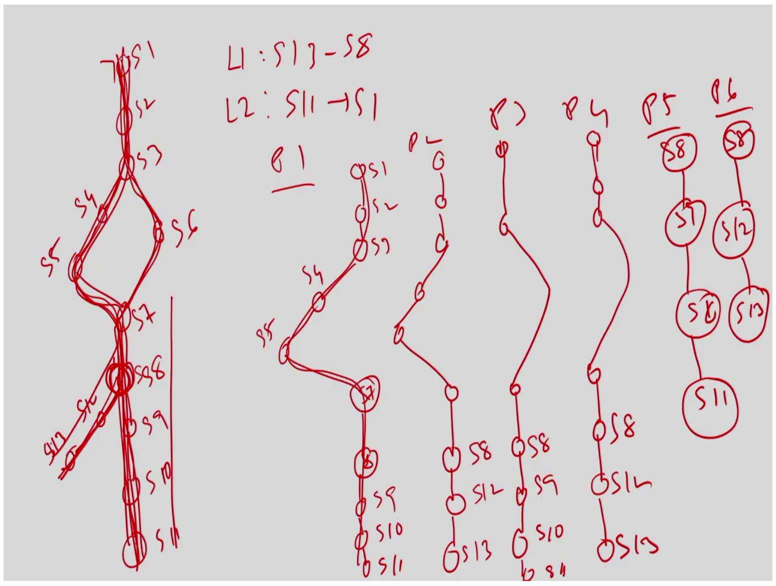
So, if I have this, so you can you will come here. And then, it will go on, go on, go on, so it might have any infinite number of paths. So, just to avoid that I will remove the back edge, but since I am starting my longest path again from the loop point, so it will not create a problem because I have eventually I know that this is the starting point and from there if I just add the back edge whatever the schedule I am going to get, if I add the back edge it will make sure that it will actually happen in loop. I am going to discuss that thing in detail in subsequent part.

But let us assume that I am going to remove the back edge before identifying the longest path. So, now if you just remove this and then, you get this how many longest paths are there. So, obviously, if you start from this node, so this will be one longest path. So, this is another one, you start from here you go to the else branch. So, then this is my path 2.

And as I mention, 7 was the loop point. So, I am going to start the third path which is from this to this. So, there are P1, P2, and P3. So, basically in this behavior there are 3 paths. So, that is what given by this. So, this is my path 1, this is path 2, and this is path 3, which is starting from 7 to. So, this is something understood.

So, now, the interesting things here you see that here that loop was busy varying, there is no operations inside this. Let me take an scenario where if there is some operation inside the loop how many paths well identified.

(Refer Slide Time: 24:57)



Let me take a very simple example, say I have this, and then, there is a if-else. So, there is a if-else like this and here you have a loop and the loop has say two state and it will coming like this. And then, you have the end state. So, if this is the behavior and there is a back edge from here as well. How many paths will be there? So, let me give the name as say S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13.

So, I am going to remove this back edge. So, the back edge is basically loop 1 is S13 to S8. So, I am going to remove this. So, I am going to remove this and I am going to remove this one also. So, I have another loop which is loop 2 which is from S11 to S1. So, this information I will remove.

Now, we can see this is my modified behavior. So, how many paths are there? So, basically I have path 1 which is basically S1, so it will take like this. So, let me just draw properly. So, path 1 is basically this S1, S2, S3, then it will go to S4, S5, then S7, then it will go this way, so S8, S9, S10, S11.

So, I have one path like this and we will have another path like this as well. So, that was missing earlier. So, I have path 2 which is basically again this S1, S2, S3, S4, S5, S7 and this is S8, S9, S10, S11. So, my path 2 will be similar, but from S8 it will go to S12 and S13. This is my S8, it will go to S12 and S13, S12 and S13.

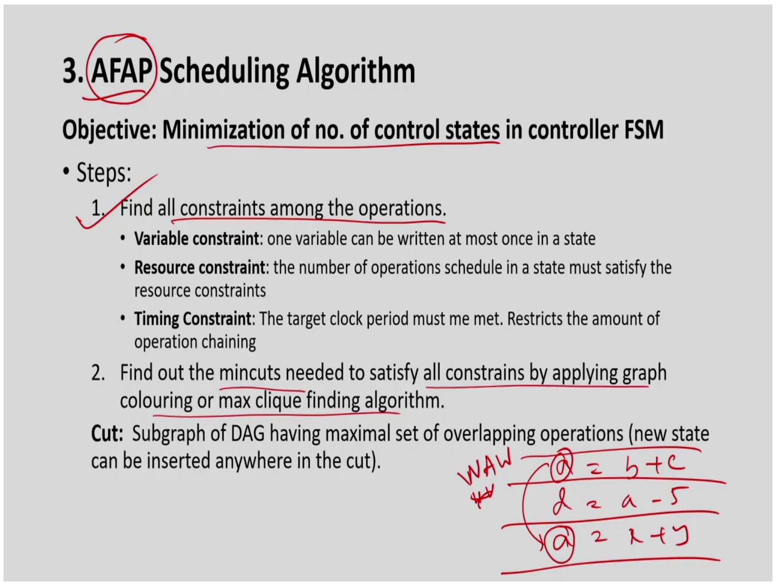
So, basically when you come from here one path will be like this and one path will be like this. Earlier, since there is no state word this was missing, so similarly, when I when you come through this, you will have one path through this, and you will have one path through this. So, there will be P3, P4, so which is I am just drawing it quickly. So, this is my S8, S9, S10, S11. So, this path and there will be another one which is P4 which will come like the same way, it will take the else branch and it will take the loop path S12, S13; S12, S13. This is my S8.

Similarly, so there are 4 paths and from this point because I have to start path from loop point there will be two paths, this one and this one. So, P5 will be S8, S9, S10, S11, and there will be P6 which is S8 to S12 and S13. So, there will be 6 paths. So, here in this is the same structure I obtained 3 paths because from the loop point the loop body was missing.

So, I have only considered one path here. So, but if we consider there is a loop point it will become two and for these two path, again it will multiply this path. So, basically there are 6 paths. I hope you understand this point. So, the point here is that given this program, I will I represent this as a CDFG, and then, I am going to identify all the longest path. So, if the behavior is like this, it will have 6 paths and it is a similar behavior.

Only thing is that there is no content in the loop body I will get 3 paths. So, this is what I we have already identified.

(Refer Slide Time: 29:26)



3. AFAP Scheduling Algorithm

Objective: Minimization of no. of control states in controller FSM

- Steps:
  1. Find all constraints among the operations.
     - **Variable constraint:** one variable can be written at most once in a state
     - **Resource constraint:** the number of operations schedule in a state must satisfy the resource constraints
     - **Timing Constraint:** The target clock period must me met. Restricts the amount of operation chaining
  2. Find out the mincuts needed to satisfy all constrains by applying graph colouring or max clique finding algorithm.

Cut: Subgraph of DAG having maximal set of overlapping operations (new state can be inserted anywhere in the cut).

So, the next is the AFAP algorithm. The first thing here you should argue with me that why we need any new algorithm because I have one path I understand. I have one path. So, what I can do? I can just put all the operations of that path, sequentially and do a scheduling the least based scheduling ILP or say force related scheduling, whatever we have already discussed, we can actually apply them why we need a different algorithm. So, let me justify why we need a new algorithm called AFAP.

The first thing is here, obviously, the objective is to minimize the number of control state overall. And specifically here we actually apply operation chaining. So, if you just see think about this force related scheduling or rescheduling, operation chaining is not considered and adapting operation chaining there is little bit difficult because I do not know how many operation chaining can be done. So, that is something is a first reason.

And the second reason is most important is say for example, suppose you have a operation a equal to b plus c, and then, you are doing say d equal to a minus 5, and then, again you are doing a equal to say x plus y. So, you basically a gets redefined. So, in that kind of; so, there is a dependence it is a read, write after write. So, this is called write after write dependencies.

But that kind of things is not captured in basically in sequence graph. And as a result if you schedule them in two different time step its fine, but if you schedule them in same time step it will create a problem. So, because you cannot update one variable two times in a single clock. So, you have to schedule this into two different time step. But if you just think about a sequence graph this may not be reflected. You have to add this kind of things there as well.

So, and specifically, so in this particular AFAP algorithm these things are already taken care very efficiently. So, that is why here instead of applying the normal basic-block based algorithm I am going to apply this algorithm called AFAP.

So, what are the steps in the AFAP algorithm? So, what is the input to the AFAP algorithm? You have it will take one path at a time. So, it will take one such path. Path means all the operation in the path and it will give you the minimum number of control state where to schedule all these operations.

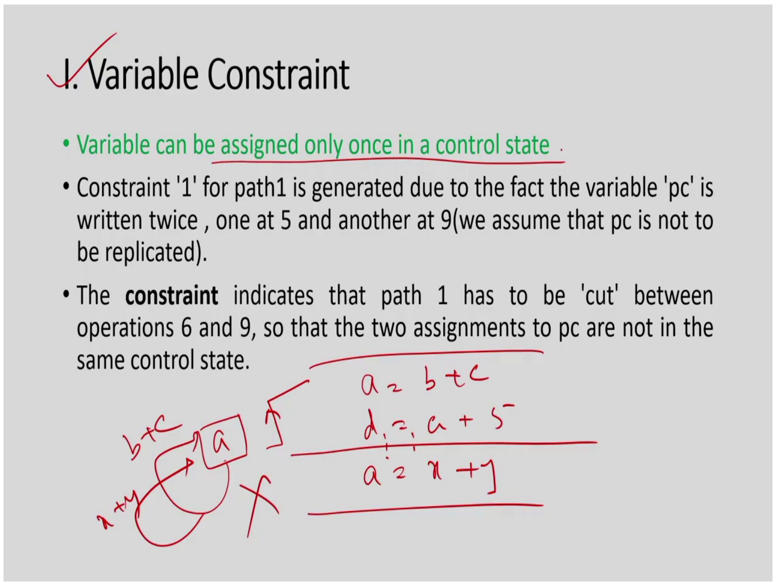
And this is basically very conservative approach, it is initially assume that everything can be done in one control state. And then, you think is it possible to do everything in one control state? If not, what is the problem, what is the thing causing problem? And you try to satisfy that constraint, and you try to increase the number of control state minimally, so that things can be done in minimum number of control state. So, what is done here?

So, as you understand that you have to identify first the all the constraint. So, what are the constraints among the operations that you have to figure it out first. So, I will talk about them in detail. And once you identify the constraint, you find out that minimum number of states that actually satisfy all the constraint.

So, that is the next type. You identify the mincuts. I will talk about what are mincuts needed to satisfy all constrains by applying graph colouring or max clique finding algorithms. So, I am going to talk about that, do not worry.

So, let us first identify what are the constraints to be satisfied. In a path there are many operations. For example, this path has 11 operations 6 sorry 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 operations. So, there are 10 operations, and I want to identify the minimum number of control state to schedule this 10 operations.

(Refer Slide Time: 33:13)



I. Variable Constraint

- Variable can be assigned only once in a control state .
- Constraint '1' for path1 is generated due to the fact the variable 'pc' is written twice , one at 5 and another at 9(we assume that pc is not to be replicated).
- The **constraint** indicates that path 1 has to be 'cut' between operations 6 and 9, so that the two assignments to pc are not in the same control state.

So, the first constraint is variable constraint. So, as I mentioned that one variable can be assigned only once in a control state. The example that I have just taken that a equal to b plus c, then you are doing d equal to a plus sorry, 5, and then, again you are doing a plus a equal to x plus y. So, it is possible there may be some operations here and there.

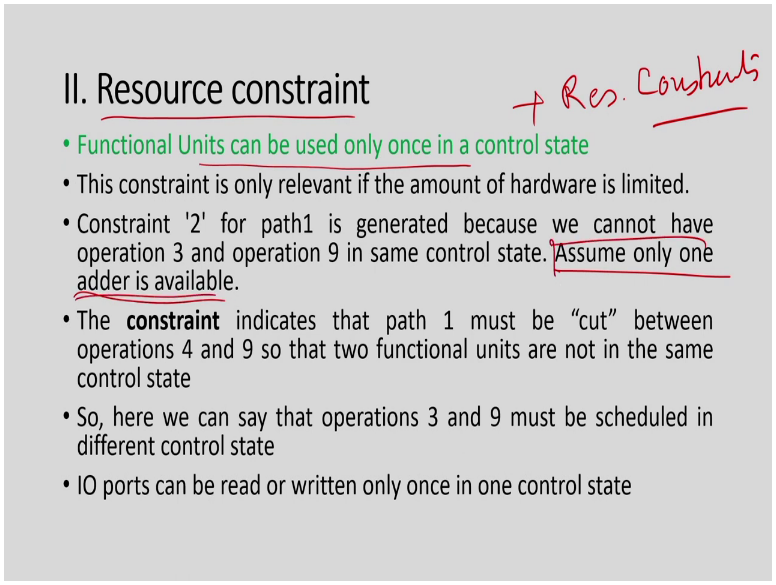
So, see, finally, what is going to happen? This is going to store in some register in the hardware. So, and now if you assume that I am going to schedule everything in one clock, what is going to happen? You try to first update this register with b plus c, then you try to update with x plus y in the same time step, which is not allowed, because you cannot update because that update of the register is going to happen at the next clock.

And say, if you try to update twice it will actually create some problem because it is creates some risk condition.

So, basically the first constant says that if one variable get redefined you cannot schedule that two operation two operation in the same steps. That means, you have to break somewhere here. You cannot do these two operations at the same time. So, that is what is the first constraint, that variable can be assigned only once in a control state.

So, that means, you have a path, you identify is there any variable is getting defined twice and if it is getting defined you have to break the whole path between these two. So, if you can assume that I can actually execute everything in one clock, now because of the variable constraint I under understand that this cannot be done one, you have to cut the whole path in two times. And then, the path the first path is one state and the second path is another state.

(Refer Slide Time: 35:00)

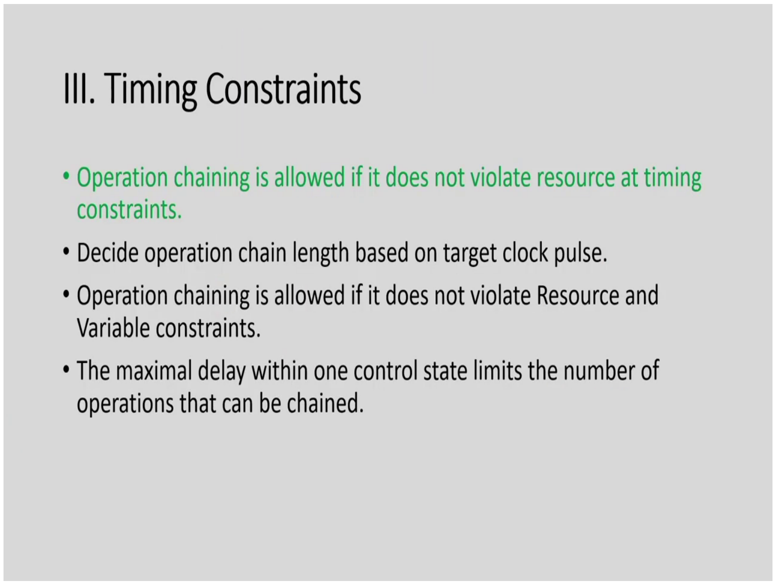


II. Resource constraint

+ Res. Constraints

- Functional Units can be used only once in a control state
- This constraint is only relevant if the amount of hardware is limited.
- Constraint '2' for path1 is generated because we cannot have operation 3 and operation 9 in same control state. Assume only one adder is available.
- The **constraint** indicates that path 1 must be "cut" between operations 4 and 9 so that two functional units are not in the same control state
- So, here we can say that operations 3 and 9 must be scheduled in different control state
- IO ports can be read or written only once in one control state

(Refer Slide Time: 35:01)

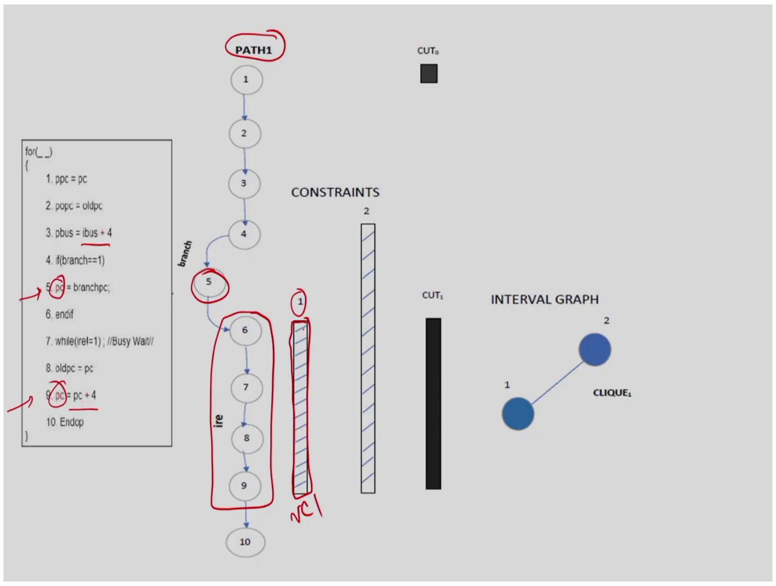


III. Timing Constraints

- Operation chaining is allowed if it does not violate resource at timing constraints.
- Decide operation chain length based on target clock pulse.
- Operation chaining is allowed if it does not violate Resource and Variable constraints.
- The maximal delay within one control state limits the number of operations that can be chained.

So, let me take the example that I have taken, and then, explain this.

(Refer Slide Time: 35:02)



```
for(_ _)
{
    1. ppc = pc
    2. popc = oldpc
    3. pbus = ibus + 4
    4. if(branch==1)
    5. pc = branchpc;
    6. endif
    7. while(irel=1) ; //Busy Wait//
    8. oldpc = pc
    9. pc = pc + 4
    10. Endop
}
```

Say, for this behavior is there any variable constraint? Yes, say suppose you take a path 1. So, path 1 is this. You can see here I am updating pc in line number 5 and again in line number 9. So, that means, I am updating this pc in line number 5 and again in line number 9. So, as I mentioned that this two cannot happen in the same state. So, initially assume that if the whole path can schedule in one control state, but now I understand that no it is not possible.

So, where to cut this path? I have to cut the path anywhere between this from this here to this. So, because after this you can cut here itself, but there is no problem if you cut it here or you can cut here or you can cut here, because even if you cut here this two operation will give different time step. So, that is what called the variable constraint and you basically mention this constraint as a bar.

So, you are saying that essentially that since operation 5 and 9 is two operations which is updating the variable pc, I have to cut this path somewhere between 6 to 9 which is given by my this constraint pc constraint, pc 1, so that this variable constraint is getting satisfied. So, if you assume that, so you can as I mention that you can cut at 6, 7, 8, and 9, but wherever you cut it satisfy my constraint.

So, this constraint I represent by a bar because there is a multiple option to cut. So, I am going to chose this later. I do not take a decision at this moment that I am going to cut 6 or 7 or 8, but I represent the range where the path must be cut to satisfy this variable constraint. So, this is variable constraint. So, you can have many such constraint, but I just for our example I have only one such constraint.

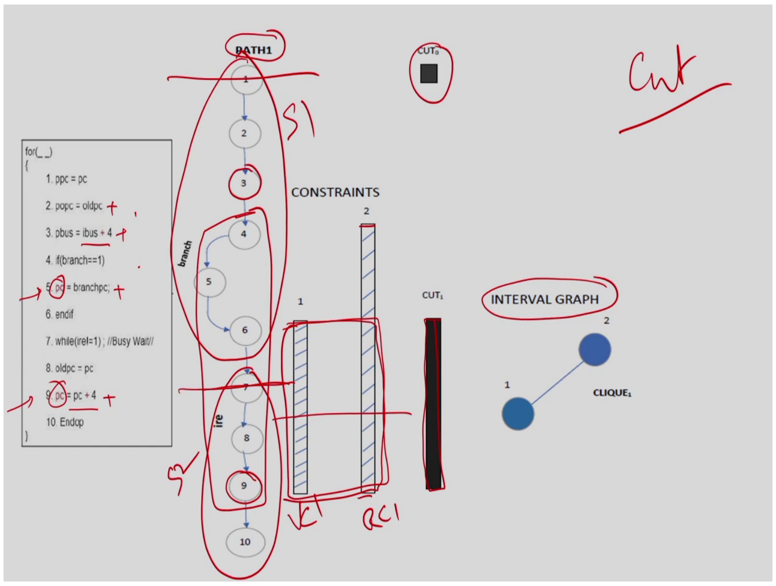
So, next was the resource constraint. So, as I earlier also clarify that if there are say 20 of addition operations and if you schedule all 20 unit, you need 20 adders. But it may be that you do not have 20 such adder available say you have say 10. So, you have to cut the behavior such that in every state there are 10 operations. So, the basically the constraint says that you can only, you can only execute some operation in one state as long as long as it satisfy the resource constraint.

So, first is the resource constraint is gets satisfied and the second point is that you cannot reuse the one function in two times in a state that is very obvious. So, if there is 1 adder it has to be used I can do only one operation at a in a state. So, the resource constraint basically clarify both thing that you use the adder in once in a state and you actually satisfy the resource constraint. So, basically satisfy the resource constraint. So, this is the resource constraint.

So, if you now take or example and say I have only 1 adder available. So, this is my resource constraint, there is only 1 adder available. And see how many addition operations are there? So, if you take the path 1 again you have one addition operation and one addition operations. So, there are two addition operations.

And if I assume that, so I cannot do this to addition in the same time step because if I one function unit I can only use only once, that is my resource constraint. So, I have to add a constraint between after 3 and before by 9.

(Refer Slide Time: 38:43)



```
for(_ _)
{
        1. ppc = pc
        2. popc = oldpc
        3. pbus = ibus + 4
        4. if(branch==1)
        5. pc = branchpc;
        6. endif
        7. while(irel=1) ; //Busy Wait//
        8. oldpc = pc
        9. pc = pc + 4
        10. Endop
}
```

So, my next constraint say that; this is my, so if you just now think about this, so this is my 3 which is addition operation and this 9 is another addition operation. So, I have to cut somewhere between 4 to 9. So, this is my range where I should cut. So, that is my resource constraint 1, that you can cut at 4, we can cut at 9 also because cut at 9 means you are cutting here. So, 9 will go to the next time step. So, effectively this is what the resource constraint you are saying.

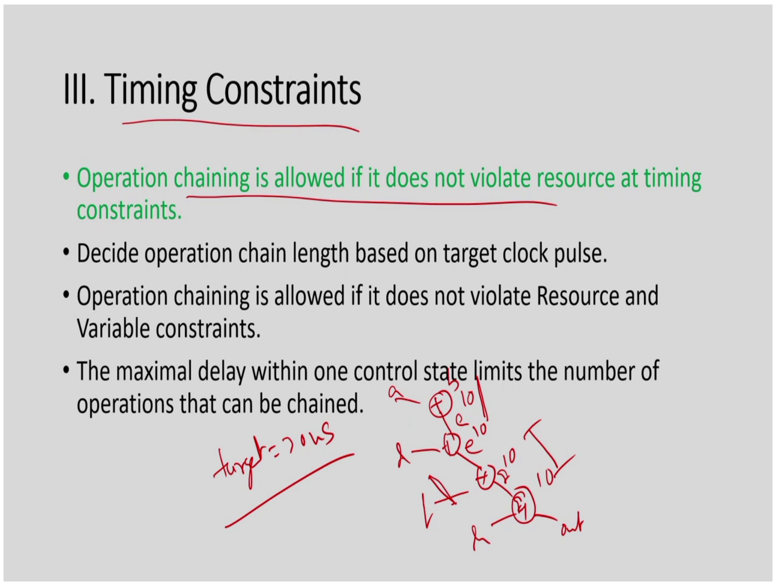
So, now, if there are say, you can understand if there are say 4 such addition operation and only 1 adder you will have many resources. So, you have to satisfy any pair you have to cut. So, here I am taking very simple examples. So, only one constraint, but you may end up say you just think about this is also plus operation, this is also plus operation, this is also plus operation, this is also plus operation.

So, there are 4 operations and say I have only 1 adder. So, that means, all this 4 operation cannot be schedule in 1 time step, I need at least 4 time step to execute this. And in the constraint I will identify one constraint between these two, I will constraint I will consider one constraint between these two, and I will also identify one constraint between these two. So, there are 3 constraints to be added here.

Obviously, you can think about there must be some constraint here to here, but because this two constraint satisfy this. So, I do not need this. So, the point here I am trying to cover that this that resource constraint may add many such constraint. So, this kind of constraint you have to add for all possible resource conflict. So, this is what is my second constraint.

So, I have discussed about the variable constraint, I have discussed about the resource constraint. So, now what is the third constraint? Next, we discuss the timing constraints.

(Refer Slide Time: 40:21)



III. Timing Constraints

- Operation chaining is allowed if it does not violate resource at timing constraints.
- Decide operation chain length based on target clock pulse.
- Operation chaining is allowed if it does not violate Resource and Variable constraints.
- The maximal delay within one control state limits the number of operations that can be chained.

target = 20ns
a
b
10
c
d
10
e
f
10
10
g
h
out

So, as I mentioned that operation chaining is something is the default option in timing this path-based scheduling because here I want to schedule everything as soon as fast as possible. So, whenever there is no timing conflict, so I am going to meet timing I am going to allowed operation chaining as much as possible.

So, by default I will assume that everything can be all whatever the chain of operations are there I can schedule in single clock as long as it meets my timing constraint. For example, say there is a adder chain is like this. So, I am doing say a plus b, then this is c, and then, you are doing say d plus c, and then, you are getting e, and then, you are doing e plus f you are getting g, and then, say you are doing again a plus operation of which is say g plus h and you are getting say out.

So, there are 4 such operations in chain. So, it will create a chain of operation in my behavior because that is how I have explained. And by default I will assume that all 4 can be done in single clock. Now, assume that this all every adder is 10 nanoseconds, taking 10 nanoseconds to execute.

And if you are target clock is 30 nanoseconds, so it says that this 40 nanosecond is path length if you do a chaining of 4 additions, it will create a path length of 40, but your target is 30, so your timing constraint not met here. So, you have to break this path. And where to break? You have to break either between this here. So, you have to you have a add a constant between this third and fourth operation or you can also have can break here.

So, you just break it here, then this part you do or you can break here also because you just do two operations in clock 1 and in the next two operations in clock 2 in state 1 and state 2. So, there are many such constraints going to happen. So, if we just take this example, how many constraints will have? I will have 3 constraints.

One constraint says that you break between this third operation and 4 operation and you execute first 3 operation in state 1 and fourth operation in state 2 or you break between this second and third operation and say that first two operation execute in state 1, third and fourth you execute in step 2 or you break it between 1 and 2, and say that I am going to do only one in state 1 and 2, 3 4 in state 2. All are valid.

So, for this timing constraint you have to add this kind of constraint. In our example there is no such timing constraint because these are all only one addition operations are there, all are assignment operations. So, there is no timing constraint for these. But you actually can have many such timing constraints.

So, what I am going to get after this 3 constraint? So, I will take a path because it is taking one path and 6 schedule it. So, I am going to take a path and the path is longest path and I identify 3 kind of constraint, the first constraint is variable constraint. Is there any one variable getting redefined? You break it in between. Is there any resource is there any long path which is not satisfying my resource requirement?

So, you break it in between. And also you have any timing violations. So, there is a chain of operations is it violating my timing constraint, then you break it. And you break it all possible way, so that it actually satisfies my timing constraint. So, once you have all these constraint, what is my objective? My objective is to identify the minimum number of state which we can satisfy all the constraint.

So, here I am going to talk about the state is now by cut. So, the point here is that, so the problem statement here is that I have given the constraint, the constraint is given by this bar because it is not a single state, but its a range of state where I can break the path. So, I have such constraint and I need to break the path into minimum number of time, so that all my constraint gets met. So, that is my problem statement now.

And this is what is very important and interesting problem. So, if you just think about my graph which is very simple. So, I have one constraint, variable constraint which is this and this is another constraints. So, if you just cut anywhere between this plus, this both will satisfy. So, I do not have to say if I just cut somewhere here, again I have to cut it here. So, this is yeah redundant cut, I do not need to cut here.

To satisfy this both the constraint, I have to cut somewhere between this 6 to 9. I should not cut at 4 or 5 to meet my resource constraint. Then, I have to again another cut to satisfy the variable constraint. So, the point here that as a mentioned that the problem statement here as a given the constraint, you identify the minimum number of cut or the range, one cut is a one range, where you can cut and such that the minimum number of cut ensures that all the constraint will be satisfied.

So, if I take my example here, so obviously, I had a cut here, you have to start something here. So, this is my state 1. And I have to cut somewhere between 6 to 9 and that is given by this. So, suppose I decided to cut at node 7. So, what does it mean? Abruptly that I am going to execute this part of the behavior in time step 1 and I am going to execute this part of the behavior in time step 2. I need S1 and S2.

Then, understand that there are 10 operations and at least for my example I need only two states. So, that is the benefit I am going to get through path-based scheduling here.

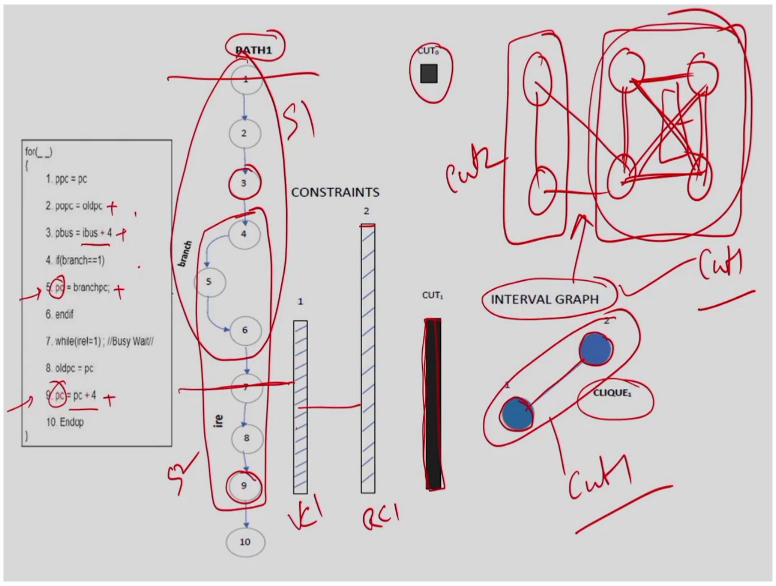
Actually, the number of state get significantly reduced because my objective is to minimize it and I am going to do as much of operation chaining, as much of sharing, as much of operation chaining, as much of resource operation to execute time step as long as they does not violate my resource and timing constraint and variable constraint.

So, here I understand that for this behavior for one path I understand that there are two constraints and to satisfy this constraint I have to somewhere cut between this. And if I cut it here effectively I have two state, in state 1, I am going to execute all operations from 1 to 6 and in second time step I am going to execute all the operation from 7 to 10. So, this is understood.

So, this is manually possible because there are only two constraints under the two such bar. But if there are say 100 such bar, identifying this manually is not a way. And specifically, I am assuming this can be done by a software, the high level synthesis tool that I am going to do it.

So, how can I automate this process? So, that something is to be understood. So, how to automate this process? It is very interesting. So, what I am going to do? I am going to construct the interval graph. So, and then, let us try to understand what is the interval graph.

(Refer Slide Time: 47:16)



```
for(_ _)
{
    1. ppc = pc
    2. popc = oldpc
    3. pbus = ibus + 4
    4. if(branch==1)
    5. pc = branchpc;
    6. endif
    7. while(irel=1) ; //Busy Wait//
    8. oldpc = pc
    9. pc = pc + 4
    10. Endop
}
```

So, in the interval graph each such constraint will be a node. So, I have one constraint pc 1, so this is node 1; I have constraint two the resource constraint 1, this is my node 2. So, the number of node in the interval graph will be the number of such constraints.

And now I am going to add an edge here if this two have an overlap. So, if there is since these two constraints are overlapping. So, I will have an edge. So, this way I am going to construct an interval graph from this constraint. And what is my objective? My objective is to identify the minimum number of cuts that satisfy all my constraint and that can be done using clique partitioning problem.

So, what is that? You identify the clique is complete sub graph. So, you identify the complete sub graph, in the graph, in this particular constraint graph or interval graph, such that all the nodes are getting covered by one of the complete sub graph and there is no overlapping.

Let me take an example of such. So, this is the very simple example. So, I need only one cut. So, it says that this is a complete sub graph. So, complete sub graph means it is have all the edges with them. So, this is my cut 1. So, it says that you need only one cut to satisfy both the constraint. And that cut I can identify by the common part in this two.

Now, let us take a very complex; say my the interval graph become like this. So, now you see here, I identify there is a complete sub graph here. This is my one complete sub graph. And then, in this sub graph I have 4 nodes. So, what I am saying that? You can actually have only one cut, cut 1, which will satisfy all this 4 constraint because they are all compatible to each other.

So, if they are all compatible to each other, so I will have one cut want to satisfy this. And since this is one cut and this I need another cut here which is also a complete sub graph of two nodes and this is my cut 2. And I do not I should not consider this cut of 3 because this is already taken care by this. This is also a complete sub graph say. This is node of node 3, but I am not going to consider this because this node is already taken care by this cut 1.

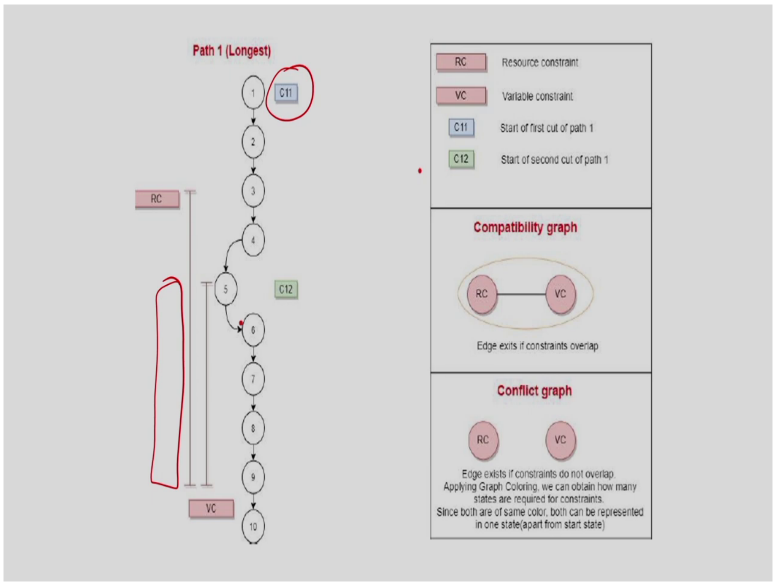
So, the idea here is that if you identify a complete sub graph that means, this all nodes are overlapping each other. So, all of them are overlapping to all of them. So, I can actually have one cut to satisfy this. So, if you ask me how to do it automatically, I am going to do it, I am going to construct a interval graph from this constraint for each node represent one node in this interval graph each constraint, and if two constraint have overlapping then I am going to add an edge.

Once I have that, I want to identify the maximum sub graph which is a complete graph. Complete graph means they have edge between all of them. And if there is a complete sub graph where everybody is connected to everyone, that means, they are all compatible to each other.

So, that means, they can one single can single cut can satisfy all this constraint. So, I am going to add only one such cut to satisfy all this constraint. Similarly, I am going to identify the other set of such complete sub graph and for each complete sub graph I am going to add a one cut. So, this is how I can actually automate the process.

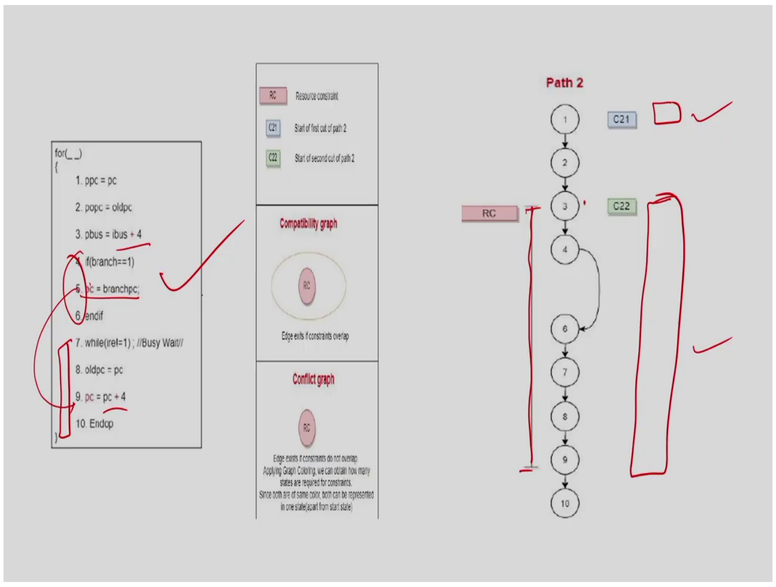
So, this I have to do it for all paths. So, I will going to I will take path 1, I will identify the constraint, I will create the internal graph and I will solve it the clique partitioning problem. This is called clique partitioning problem. And then, I am going to find out such cliques and each clique is one cut. And from this 4 node, I will identify what is the common overlapping time and that will be my cut. This is how I can do it for all one path.

(Refer Slide Time: 51:06)


Path 1 (Longest)
1
2
3
4
5
6
7
8
9
10
C11
C12
RC
VC
RC Resource constraint
VC Variable constraint
C11 Start of first cut of path 1
C12 Start of second cut of path 1
Compatibility graph
RC
VC
Edge exits if constraints overlap
Conflict graph
RC
VC
Edge exists if constraints do not overlap.
Applying Graph Coloring, we can obtain how many states are required for constraints.
Since both are of same color, both can be represented in one state(apart from start state)

And I have to do it for all paths. So, this is done for path 1 and I identify that I need two cuts. So, one cut is from here and another cut is like this, so basically what says that you basically cut between path 5 to path 9.

(Refer Slide Time: 51:23)



```
for(_ _)
{
    1. ppc = pc
    2. popc = oldpc
    3. pbus = ibus + 4
    4. if(branch==1)
    5. pc = branchpc;
    6. endif
    7. while(irel=1) ; //Busy Wait//
    8. oldpc = pc
    9. pc = pc + 4
    10. Endop
}
```

RC Resource constraint

C21 Start of first cut of path 2

C22 Start of second cut of path 2

Compatibility graph

Edge exits if constraints overlap

Conflict graph

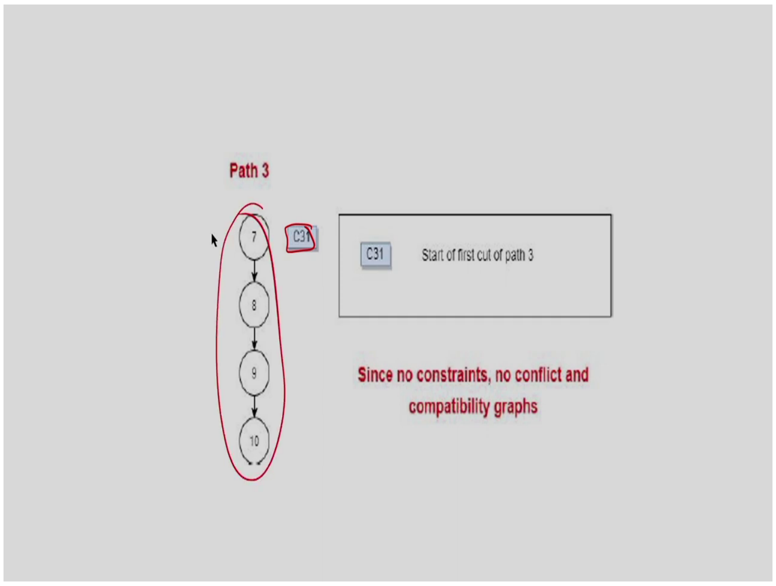
Edge exists if constraints do not overlap.
Applying Graph Coloring, we can obtain how many states are required for constraints.
Since both are of same color, both can be represented in one state(apart from start state)

So, now if you take path 2 because there are 3 paths we have identified in this behavior, so in the path 2, how many constraints are there? So, in the path 2 you see it is go out through the else branch, in the else branch there is no this pc equal to this. So, this variable assignments constraint is not there. But only this addition constraint is there. So, there is only one constraint.

So, this cut is very simple because it has only one node. So, I am going to have a cut here and one cut is from this to this. So, I can cut anywhere between this. Again I need two states to execute by path 2. And this second state can be cut anywhere. So, that is for path 2, because in path 2 I do not have any variable constraint; this is because this is not there.

So, this is within the if branch and that is not in the else branch. So, I have only one resource constraint and I know I understand that I have one cut 1 and this is my cut 2, and this is what I obtained.

(Refer Slide Time: 52:22)


Path 3
7
8
9
10
C31
C31
Start of first cut of path 3
Since no constraints, no conflict and compatibility graphs

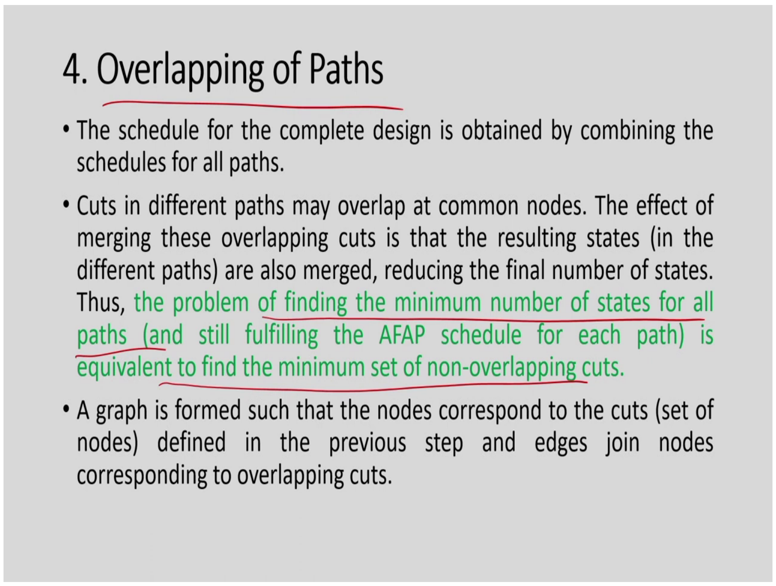
And for path 3, there is no constraint. So, if you just look into this code, in this part of the code, there is no variable constraint, no variable be defined twice. There is no adder constraint because I assume that there are only 1 adder available with there is no addition constraint and there is no timing constraint as well because there is one addition operations. So, for this I need only one cut.

So, basically it says that I can execute the path 3 in single clock. So, that is what the interpretation of this cut. So, I have 3 paths, for each path I identify what is the minimum number of cuts there, satisfy all the constraint, and once I identify that information is ready with me.

What next? So, obviously, these 3 paths are not independent. So, you can understand in this behavior, these paths are actually overlapping. The significant portion of the paths is overlapping. So, I cannot say that I will create a two paths for state 1, two paths for state 2, and one path for state 3, and I will have 5 state in the my combine in the combined graph. This is not true.

So, what I have to do now, instead of doing that I have to combine this path. And how should I combine that? So, basically if every path has different number of time step. So, say may path 1 may have say 3 time step, path 2 may have say two time step 2 state, and say path 3 may have 3 state, I have to create a combined one. So, I have to combine this state into a single FSM from this path. That is my next step.

(Refer Slide Time: 53:51)



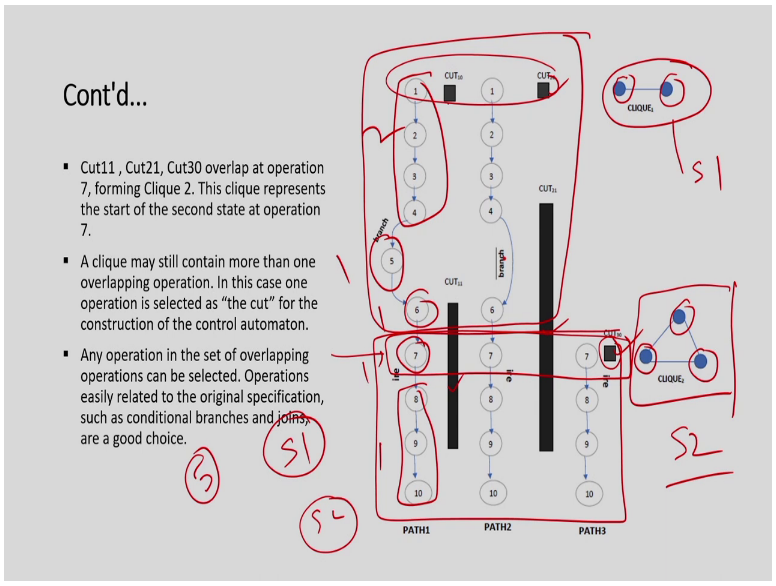
4. Overlapping of Paths

- The schedule for the complete design is obtained by combining the schedules for all paths.
- Cuts in different paths may overlap at common nodes. The effect of merging these overlapping cuts is that the resulting states (in the different paths) are also merged, reducing the final number of states. Thus, the problem of finding the minimum number of states for all paths (and still fulfilling the AFAP schedule for each path) is equivalent to find the minimum set of non-overlapping cuts.
- A graph is formed such that the nodes correspond to the cuts (set of nodes) defined in the previous step and edges join nodes corresponding to overlapping cuts.

So, that is a overlapping of paths. So, how we can do that? So, my objective way is that, I already identify the cuts or the number of states in the each path. I have to combine this paths into one and it is basically problem of finding minimum number of states for all paths is equivalent to finding the minimum number of start in the number of, find the minimum set of non-overlapping cuts.

So, what I am going to do it now, I am going to use the same concept whatever I used to solve this identify the minimum cut to satisfy all constraint.

(Refer Slide Time: 54:27)



Cont'd...

- Cut11 , Cut21, Cut30 overlap at operation 7, forming Clique 2. This clique represents the start of the second state at operation 7.
- A clique may still contain more than one overlapping operation. In this case one operation is selected as "the cut" for the construction of the control automaton.
- Any operation in the set of overlapping operations can be selected. Operations easily related to the original specification, such as conditional branches and joins, are a good choice.

So, now, if you look into this graph what I have? I have path 1, I said that I have to cut the path definitely at start and somewhere in between. I have path 2, and I identify that this path 2 has to be cut at this point and somewhere in between here. That is what we have already identified. And path 3 has to be cut here. So, we can understand this is more similar to this constraint solving problem.

So, for constraint solving, what we have done? We actually construct an interval graph and identify a clique which actually says one clique is one state. So, here I understand that path 1 also need two state, this also need two states, this also need one state, how many total number of states are there, and how we can actually combine the states. So, the similar strategy because this is the graph looks like and the problem looks like the similar, I can actually apply the similar type of things here.

So, what I can do? I can again construct a conflict graph or say interval graph. So, what I am going to do for each cut I will have a node here. So, how many cuts are there? 1, 2, 3, 4, 5. So, I will have a graph where there are 5 nodes. So, this is corresponding to this one, this corresponding to this one, this node corresponding to this one, this node corresponding to this one, and this node corresponding to this one.

So, I am going to construct this. And you can understand that it is the similar problem earlier that if there is a complete sub graph, so this is a complete sub graph of two nodes. So, that means, if I add one state here that will satisfy both, and which is obvious. So, that means, I am going to start a state which is starting from node 1.

Now, this is interesting here. So, I have these 3 cuts and you can see here they form a clique. This is a complete sub graph of 3 nodes. So, that means, if I just add one state here that will satisfy all this constraint. And what will be the state? You can see here I can break this path on anywhere between 6 to 9, I can break this path 2, when you are between 4 to 9, I can break the path 3 only at state 7. So, the common solution is state 7. So, I have to cut the path in state 7 only because this 3 can be then this all this 3 cut can be satisfy by this and that is my final solution.

So, what I obtain? That finally, I am going to break this paths into two states my S1 and S2 and S1 is going to execute this part of the behavior and S2 is going to execute this part of the behavior. So, this is what I have decided so far. So, obviously, there are many other part is left that how to add a transitions, how to put the operations into state and what will be the condition for this operation to be executed. So, these are all things to be discussed.

But I am going to conclude today here with this node that that I will take the behavior I will identify the CDFG, I will identify all the longest path and for each path I am going to identify what is the minimum number of state needed, that I am going to do.

And then, I am going to combine all the paths, states together and for both the problem identifying the minimum number of state in for a path and identifying the minimum number of states to satisfy all the paths, the problem is resolved or solved using the process clique partitioning problem.

Specifically, what you are doing is basically for either from a constraint or a from a cut I will construct a conflict graph or interval graph, and then, identifying a clique is specifying that this conflicts or the cuts can be satisfied by a single state. And then each such clique represents a state. So, this is how I decided that for my program where it has a loop, there is a if-else, there is a while loop inside for everything I can schedule the whole behavior in two state.

So, earlier it was, if you just think about the same program probably you, so if you just think about this program. So, this probably needs a two state, this needs a one state, and this say need one state, this need one state, this need one state, so because these are all loops. So, 2 plus 1 plus 3, 4, 5, so we need at least 5 states to get the complete schedule if I go by basic-block based scheduling. But here I can understand that everything can be done in two states. So, the number of control state obtained by this path-based scheduling is minimal.

So, I conclude here in the next class. I will continue this discussion of path-based scheduling, where I am going to talk about what are the operation to be put in every state, and what will be the transition, what will be the condition of the transition, and how we will generate the data path. So, that part is to be discussed in the next class.

Thank you.