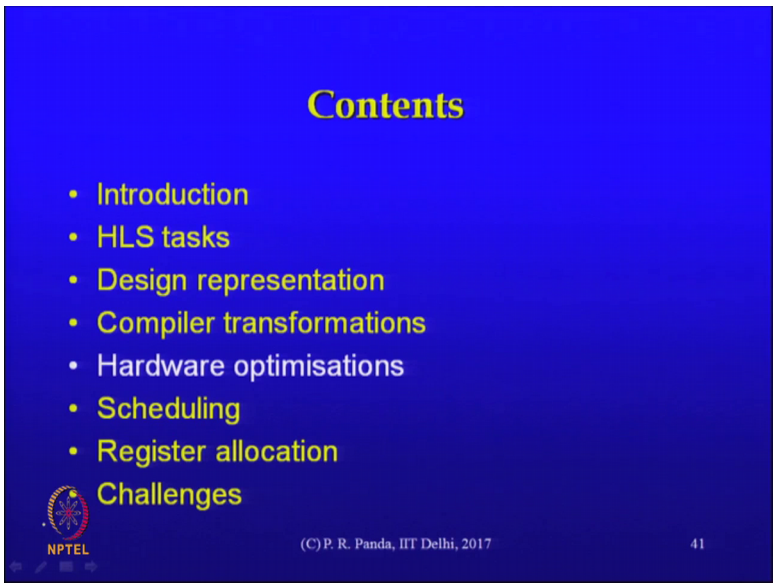
Synthesis of Digital Systems
Dr. Preeti Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 12
Hardware Transformations & ASAP / ALAP Scheduling

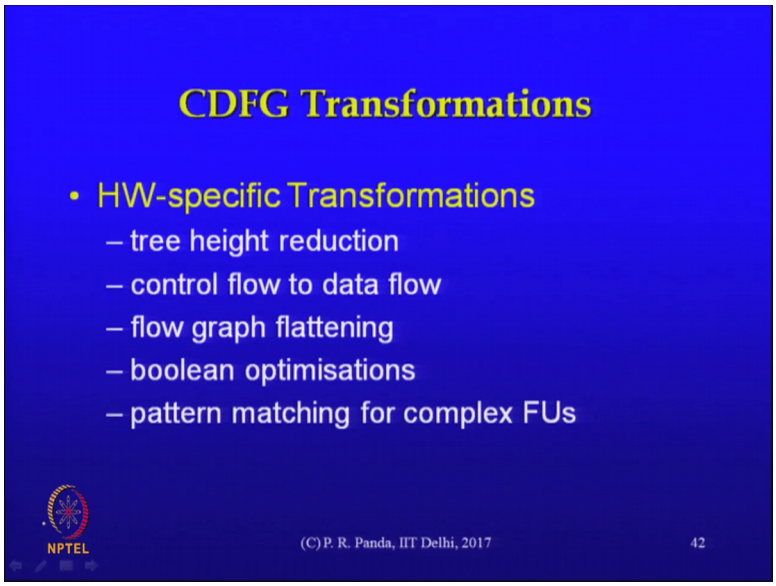
(Refer Slide Time: 00:19)



We will look at a few other optimizations that could fall in the category of hardware specific optimizations these might not make obvious sense in the context of a compilers sometimes it might, but usually it might not, but in the synthesis context it definitely does we still have not started with the actual synthesis tasks which would be the scheduling register allocation and so on.

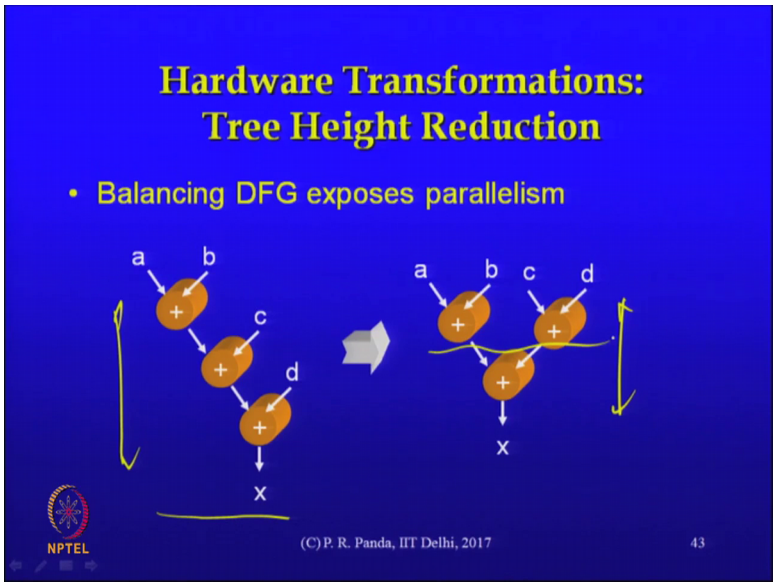
But this is still a preprocessing of the internal representation, but this keeps in mind the fact that the targeted synthesis and not compilation.

(Refer Slide Time: 00:57)



So, these are some of these transformations again the extent to which we will study these individual transformations would be primarily to point out the intuition, we will not necessarily look at the detail algorithm for doing.

(Refer Slide Time: 01:16)



Tree height reduction, the objective is to balance the data flow graph in some way wherever there is a choice. We have data flow graph that might have been inferred in that fashion from the way the user wrote the code and therefore, the representation looked like this, but you can see that we can transform that expression structure in some way

which would lead to the data flow graph changing it is structure. What would be the advantage of doing so?

Student: (Refer Time: 01:50).

The critical path which could be defined as the length of the path from any of the inputs to any of the outputs this is a data flow graph. So, there are no cycles here. So, there is a defined maximum path length, but in general the point of the balancing is to try and reduce the maximum height of the structure this, what is called tree height reduction. Doing this transformation leads to a reduction in the length maximum length or height from 3 down to 2.

What is the advantage of doing? So, now, if I submit this transformed graph into scheduling and if I have enough resources, then I can finish this faster essentially I will need only 2 cycles if each operation takes only 1 cycle. What if I did not have 2 resources, if I had only one adder then does the transformation make sense or not it does not improve, but does it make it any worse it also does not.

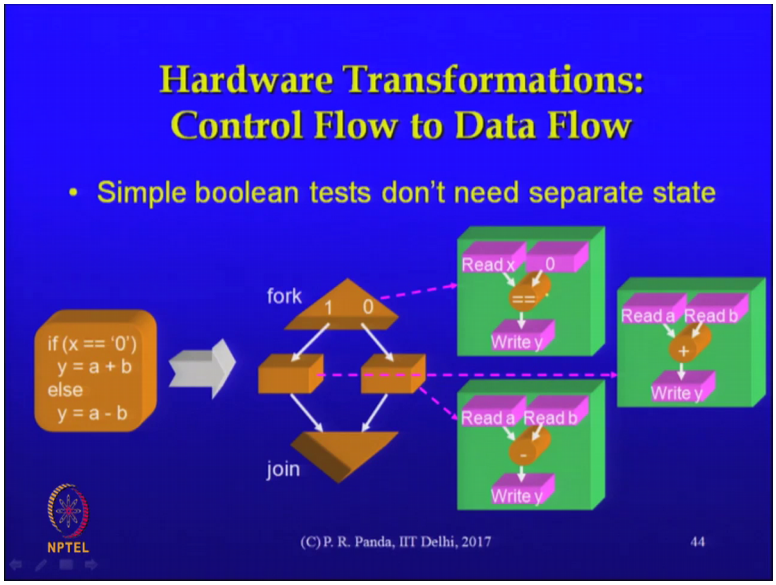
So, we will say that we will go ahead and perform this anyway, but on the other hand whether you should do it or not then depends on how many resources you have? We could still take the view that we will go ahead and do this anyway, it cannot be worse right it can only be better that depends on what resources you have later on you could also make it dependent on the resources, but the whole point of considering this as a preprocessing step is that we want to decouple these phases to the extent possible.

If we can avoid bringing in later stages like the scheduling is certainly a later stage. If we can somehow assure ourselves that the transformation that we are making is at least no worse will lead to no worse results for any of the subsequent passes. Then it may be to go ahead with the transformation right. So, that is a tree height reduction the objective is to the extent possible balance the DFG.

How do you do this you have to give this through a sequence of rotations of these nodes? Even though here the example is simple enough, in the general case this has to be done through individual transform you take each of the nodes and some rotation has to be performed essentially; something like this got transformed to something like this right and that caused a reduction in the overall height.

So, we would not go into the algorithm for doing so, but just pointing out that such a possibility exists and it may improve the results of synthesis.

(Refer Slide Time: 04:54)

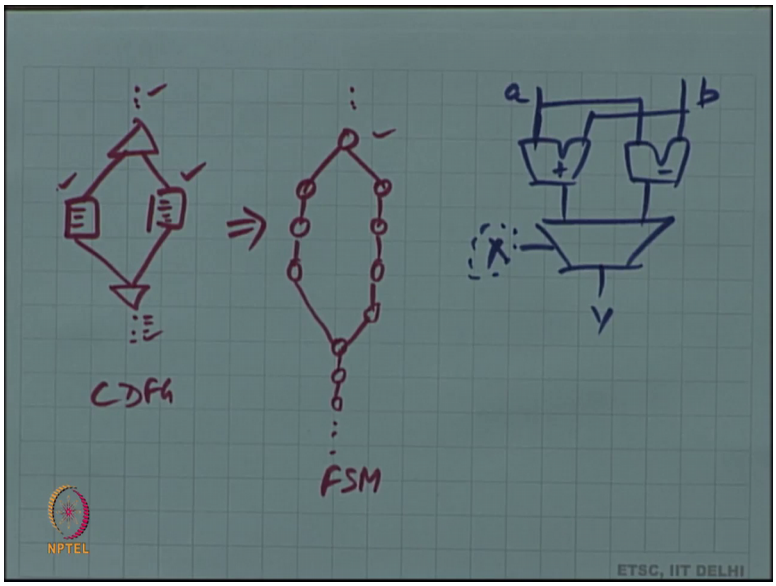


One more control flow to dataflow refers to another kind of simplification that could be done on the CDFG again with the objective of simplifying the final design.

What we have here is an if statement, if there is a condition then we are assigning something to y, otherwise we are assigning something else to y. What is the internal representation, this we had already seen since there is a an if statement that translates to that for join structure in the IR. What is there in the form join structure each of these nodes translate to respective DFG of that basic block. So, y equals a plus b is this basic block y equals a minus b is this base.

Then I have that condition that also in the general case might have it is own expression, that many do you need ultimately to be merged with the prior basic block like we had seen earlier, but that is what the general CDFG structure looks like. And the most straightforward way to take it to the next level of synthesis is that you schedule one basic block at a time. So, that you have from an if statement like that.

(Refer Slide Time: 06:22)



I will write this could in general will be several clock cycles in length, and what would the overall strategy be we schedule each of the we schedule this basic block separately this separately this one separately and so on.

Each would lead to a partial finite state machine. If this was 3 states then I would have sequential FSM with 3 states if this one was 4 states, then I would have an FSM that looked like that a condition here if that condition check is performed in one cycle wood would look like that. So, we are building the FSM along with doing the schedule and the simplest way is each basic block you scheduled separately and maybe you have a structure like that.

So, you continue with these and the that the result of a schedule will lead to an FSM that looks like this from the CDFG that is the simplest way to generate finite state machine and you can see how the synthesis process will proceed. So, this example here will translate to what FSM I have to look at what are the operations that are happening there, we have to actually schedule that the individual basic blocks. And in any case when you have an, if statement you have a fork in the FSM you have multiple paths starting from the same node in the FSM same state.

But can we do something about this it seems as though it is too expensive a hardware translation considering that our, if statement was so simple. If I did not do all of this if I

did not do the behavioral synthesis at all and this was there in my STL just that statement was there in the STL, what would be the hardware that that translates to.

Student: (Refer Time: 08:37).

You have a mux structure, in Which, What are the inputs, what are the output?

Student: Input selection times (Refer Time: 08:44).

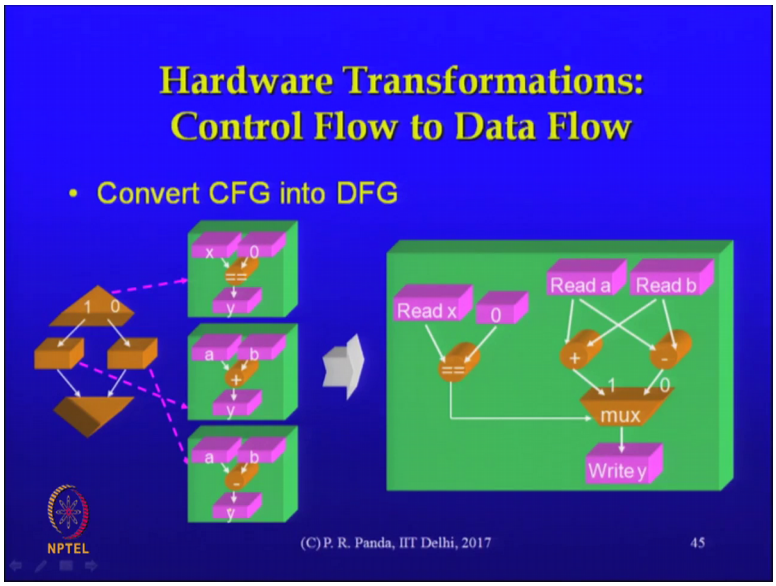
That is just x of course, the x equal to 0, simplifies this if this was a more complex expression, then that complex expression would be evaluated and the result of that anyway is a Boolean, resultant that is what would go into the select signal. The output is y .

Student: A plus (Refer Time: 09:01) a minus.

So, I have it is minus that kind of structure right this sort of a structure would be the hardware that is influenced. Could be influenced such a thing given our methodology for high level synthesis. Actually the mux was not introduced as one of those data path components, this is inferred from the fact that there is a sharing somewhere or in in this case actually it is not, because there is a sharing, but it is just that different control paths are there and different computations happen in those control paths.

However anticipating a hardware like this, we can do some at least local transformations that take a control flow graph and generate a data flow graph from it.

(Refer Slide Time: 10:03)



So, that is why the original CDFG was what we are saying is this is so simple that why do not I just create that data path element that mux there. Right now even though we have not gone to the later resource allocation and sharing kind of a steps yet. And I just translate this into a data flow graph by essentially throwing out the if statement completely, but transforming that into a mux component or a mux functionality. So, I it actually becomes a node in the data flow graph with the functionality being that of a mux.

Student: Sir.

Yeah.

Student: So, we now would not it be preferable to have an additional work mux for the fork step. So, that you do not end up computing both plus and minus simultaneously. You compute only on plus or minus as necessary. Basically we will I do not know how to do it in the FG and CFG that is what we are doing we were computing only one, but here this kind of a.

This kind of a transformation leads to some redundancy in the evaluation, whether it is needed or not, whether hardware is duplicated or not is not clear.

Student: Yes.

This is only a DFG as of now, you could schedule the DFG in the following way that could be scheduled in one this for example, it could be scheduled in a different cycle. If we wanted to remember the hardware from this is not obvious yet it is possible that we can share both of these on the same function unit like an ALU if we.

Student: Yes Sir.

Wanted to yeah that kind of a hardware is of course, meaningful it does mean a little bit of an extension of our abstraction now I should have a multi-function unit that is capable of doing both the plus and the minus. If you have an adder and sub tractor anyway and you want only the addition to be performed or the subtraction to be performed, then you could still do that, but then well you need some control signal there that tells whether to activate or not. Normally we assume that the error is just a simple combinational adder that does not take any such control signal.

Student: The average column may not be (Refer Time: 12:33), but still there will be additional power spent.

Yeah since you are doing some redundant work there is additional energy spent, average power is impacted not in this schedule, but if our schedule was this.

Student: Yes then.

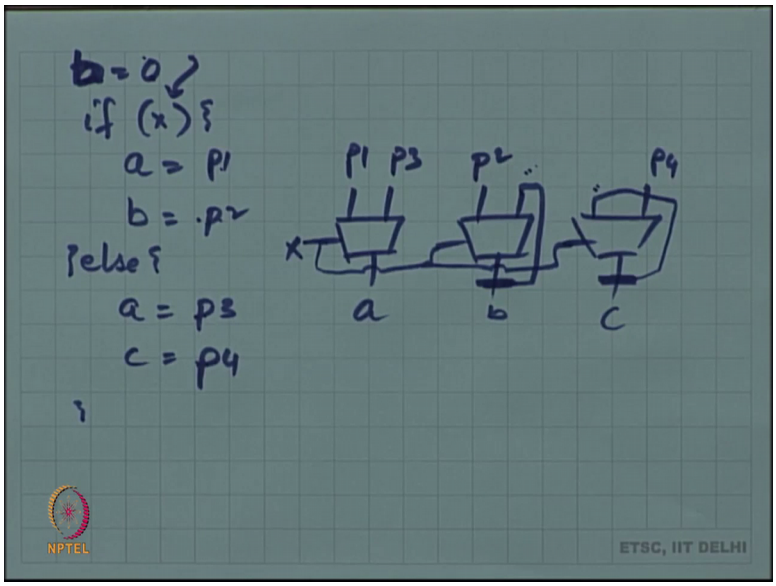
Right then it is impacted because we are doing more work in the same cycle.

Student: (Refer Time: 12:55) we need to have.

Total energy is impacted because you are doing more work and therefore, that must show up in more energy ultimately yeah, but this does avoid the need for splitting the FSM into multiple parts and joining particularly when the statements are. So, simple if they are complex then it is usually worth it, but if it is something like just one states each then it may be worth considering transformations like that. This should also illustrate the general mechanism for how to handle if statements in synthesis.

If I have an if statement that is my condition.

(Refer Slide Time: 13:37)



And I have a bunch of assignments in one branch of the, if and in the other branch of the if I have another bunch of statements and so on. What does the synthesized hardware look like just translate this into hardware without necessarily going through the behavioral synthesis steps? So, the presence of an if leads to the creation of a mux.

Now, what kind of mux it is and what do I we have at the inputs and outputs of the mux. Let us call these let us say P 1 P 2 P 3 P 4. So, that they represent some expressions that are being evaluated, but we want to say the impact of these assignments how do we translate these assignments into.

Student: Multiple muxes will be there for a.

Multiple muxes we need a mux for.

Student: Every assignment every assignment

Every assignment.

Student: like if for if a (Refer Time: 14:53) b and c 3 are there.

Yeah for every variable that is being assigned.

Student: (Refer Time: 14:58).

Just think variables that is being assigned we need a mux yeah. So, that is what a I would need. So, for a I would structure it like this. So, these are muxes and P 1 and P 3 are computed separately and one or the other goes into a. There is the question that what do we do about the fact that there is redundancy here we should be evaluating either P 1 or P 3.

But if you create hardware like that then you are into valuating both P 1 and P 3, but this is a standard delay versus maybe power trade off this does mean more power, because you are evaluating both of them from an a delay point of view it is you are finally, selecting it. So, there select signal is the same right in in both the case all the cases next what about the b?

Student: (Refer Time: 16:00).

B is assigned.

Student: (Refer Time: 16:04).

The other 1.

Student: (Refer Time: 16:08).

The assumption of course, if you leave out in one of the branches leave the assignment to b out in one of the branches, then it should retain it is old value right. So, that naturally leads to a memory element being influenced.

Student: (Refer Time: 16:26).

Right you can not directly connect this, but essentially it may be that in our context we create a memory element register latch whatever it is.

Student: (Refer Time: 16:37).

And connect that back for c it would be similar.

Student: (Refer Time: 16:50).

That depends on the methodology. In fact, a latch implementation may be like this only right. So, if there is no clock then that is how the latch would be.

Student: (Refer Time: 17:04).

Implement no the moment you put that feedback it is no longer a combinational it is yeah, you can see these are some of the early errors that you might identify when you write some VHDL and you try to synthesize, it the tool reports memory elements latches are inferred whereas, you did not intend any. So, this is just a if statement where did I create the memory element it is influenced, because of this imbalance in the assignment statement in one branch, I have an assignment statement in the other branch I do not have an assignment to that same variable then I am implying, that you remember what the value of the variable was earlier that leads to a latch a kind of functionality right.

Same happens if you have a case statement in which.

Student: It lead in the else law if you (Refer Time: 18:01) in the if law we are saying a equal to let say b.

Yeah.

Student: And in the else law we say a equal to then we will also (Refer Time: 18:10).

Else if you say a equal to a that is fine but.

Student: (Refer Time: 18:14).

The way to get around this the way to still realize a combinational circuit from here is of course, to make sure that a is assigned something earlier, let us says is initialized to a constant right. If you initialize it to some something then it does not matter, whatever that value was that is the one that would go here. That is the one that you will connect to this, because in one path you have overridden that value in the other path you have not overridden that value to b you have make it b.

So, in this path you want b to take the value of P 2 in the other path you want b to take the value of 0. So, the mux structure is actually fine.

Student: (Refer Time: 19:01) constant just before ok.

Yeah. So, that is the value that reaches this if statement, if that is clear. So, then we could we could just have, whatever that value was we could add it to the mux and that may help us avoid the latch in the general case.

Student: Sir generally the tool store the (Refer Time: 19:23) this without a process for example.

Without a process yeah if these are statements that would go inside a process yes.

Student: (Refer Time: 19:32).

Yeah yeah yeah if statement is there only inside versus yeah.

Student: (Refer Time: 19:36) it should be assigned outside that process.

Outside the if within the process, but outside the if.

Student: outside the if.

Right.

Student: So, wherever.

So, these multiple paths here through the then and through the else is what caused the inference of the mux, but the absence of an assignment in one of the paths caused the memory to be inferenced, but if you are sure that it is a combinational circuit that you have in mind, then whatever the assignment is it could be a more complex function, but that function goes as the other input to the mux and there is no need for the feedback to be introduced in the mux.

This is an important thing to remember as you try to work with some synthesis tool RTL synthesis tool. The way to write our RTL such that combinational logic is inference, even in the presence of conditional structures like this is to make sure that every path that assigns something to this is a known path, you do not rely on the on remembering the previous value right.

So, this whole thing could be inside a process and it is fine that if we do not rely on remembering the value of b from the previous triggering of the process. Everything is

still determined from this triggering, which is this function is computed right is it assignment or it is a more complex function.

It is computed the right there that would lead to a combinational logic if you rely on remembering the value of b from the last time the process was invoked, then you need a memory of it.

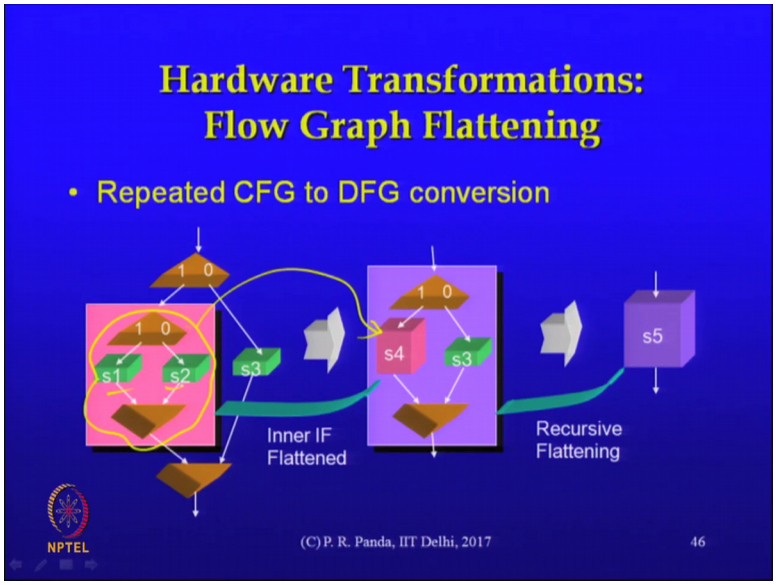
Student: Sir if we have some similar assignments in the if clock and in the s clock we simply say wait until clock pulse then latches will be created.

Wait until if you have the weight until at random places then memory is inferenced there is no way around it. In fact, RTL synthesis tools will not allow you to put weight at random places they will take the weight, but then you they will restrict where you can place the weight maybe it is at the top or at the bottom or you do not have a weight, but have only a sensitivity list then it is ok.

Student: (Refer Time: 21:51) say that ignoring this statement.

Yeah ignoring is that your own risk right it is right, because they do not have the flexibility to insert additional cycles right.

(Refer Slide Time: 22:11)



Flow graph flattening is a more general mechanism having flattened that if statement having converted that representation from a control flow into a data flow. So, essentially this part got transformed to this right.

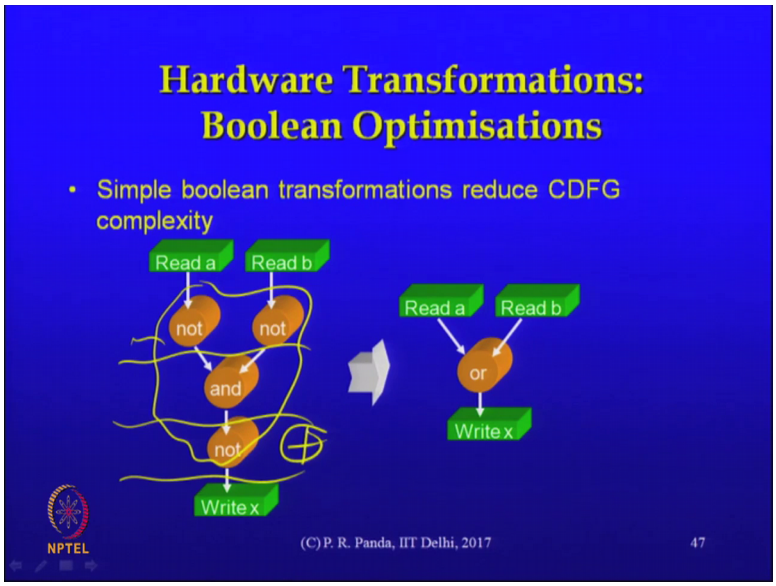
Now, this could be a nested if statement you can recursively do this flattening at higher levels also. Of course, this should be carefully done normally we would do it if those basic blocks s one's 2 let us say they were simple enough, that the logic that is resulting in the CFG to DFG transformation is also simple. At some point if you arbitrarily do it irrespective of the nesting level of the conditions, the graph might just blow up in complexity and you might not want to so.

Student: What kind of a thresholds are used to de market this decision?

Well the simple threshold could be that you constrain the width of the mux up to 2 inputs is, but if it becomes 4 or large then you do not allow that. So, some such simple heuristics could be used, other thing is that this leads to a larger number of evaluations larger number of operations being evaluated like we had identified earlier. More and more redundancy is introduced, because you if we have 8 paths only one of them is being taken you would be evaluating all the 8 operations.

So, you could actually restrict this to the number of levels that you do this over indirectly to it would also be the number of inputs to the muxes right ok.

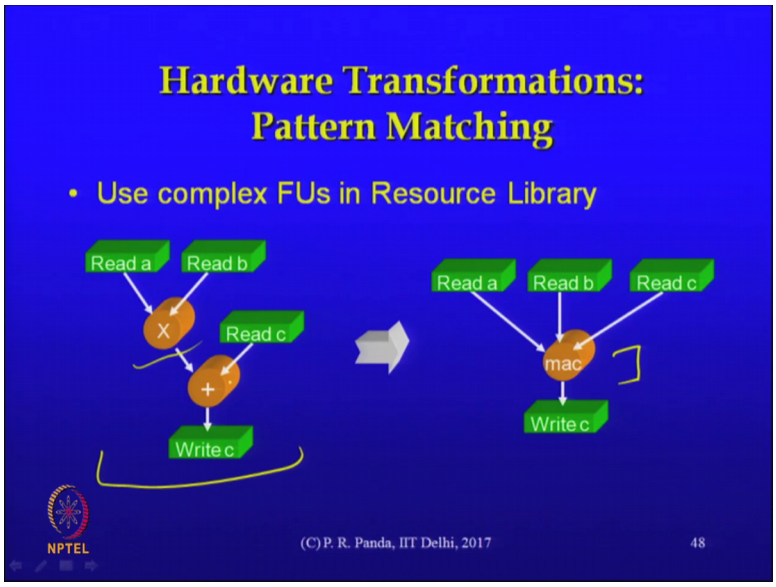
(Refer Slide Time: 24:00)



There are rules from Boolean logic that could be used to simplify parts of our expressions. So, if there is some function that I have here which may be applied to bit vectors as opposed to bits if it is simple bits then, it is alright, it will get optimized at some stage when you do logic synthesis it might optimize it out.

But some of those rules of Boolean logic particularly the ones that extend to bit vectors as opposed to just bits those one can run some paths to perform some simplifications, idea being that these are very simple operations, even though they may be operating on large bit with data. Since these are bitwise operations the time taken will be small and there may not be a need to explicitly schedule these kind of operations; considering that you also have those kind of operations being scheduled that are going to dominate the clock period anyway.

(Refer Slide Time: 25:02)



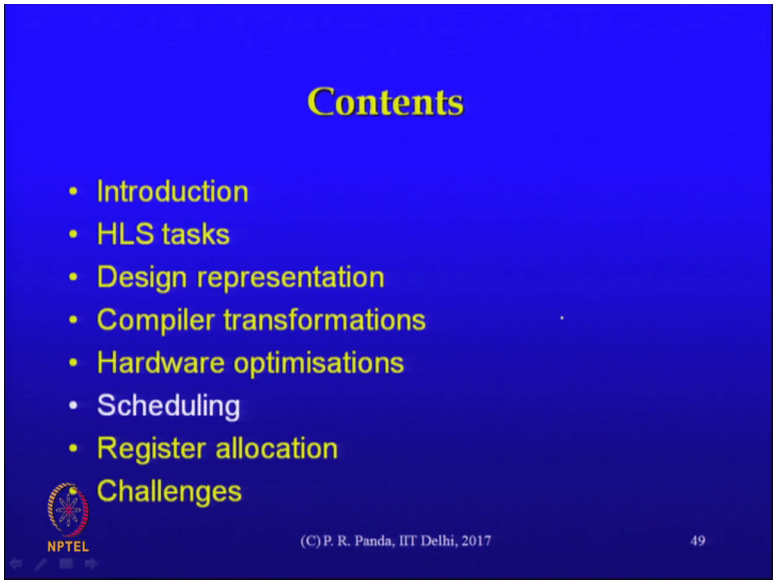
The some kind of pattern matching can be done based on what kind of resources we have in our library, those would be like this say you have c equals a b plus c this is an accumulate multiply accumulate function. One could actually simplify this into a mac operation this would make sense if such a mac unit was there. So, that unit if it is there explicitly in the library is likely to be more efficient than a multiplication followed by an addition it is hand optimized to do that.

So, such a substitution if we would like to exploit that a particular unit so essentially pointing to the fact that the same graph may be translated into many different kinds of

hardware. So, alternatives are there multiplier we do have adders we do have and a mac unit we also have and the knowledge that the mac area for example, is lower than the multipliers area plus the additional area or the same thing with the delays.

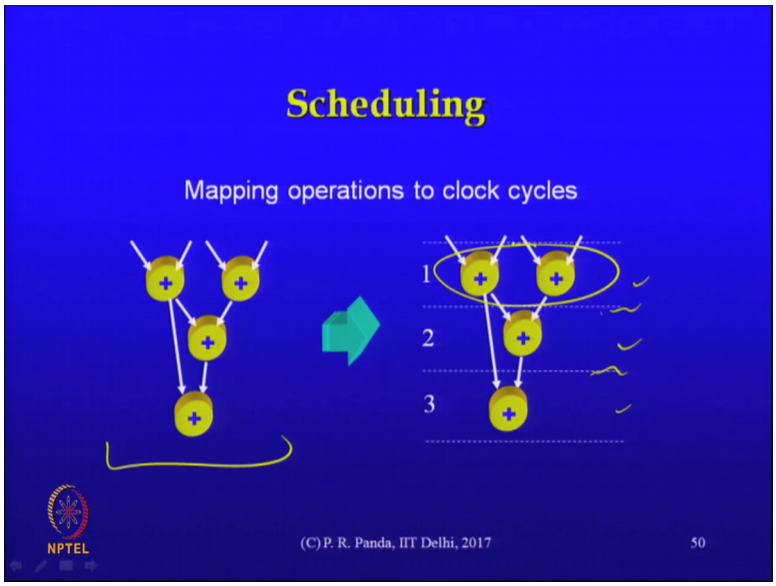
We could use that during this preprocessing step later on if I do the scheduling on that graph there then it is hard to exploit the presence of that operator in any way, but since they are individual operators I have my multiplication I have my adder, but early on it might actually be an opportunity for me to replace a sub graph a more complex sub graph by a single node knowing the functionality of that more complex library operator.

(Refer Slide Time: 26:44)



Those preprocessing steps been done let us go ahead and discuss some of the core high level synthesis algorithms. So, the first step of course, we had identified was scheduling.

(Refer Slide Time: 26:58)

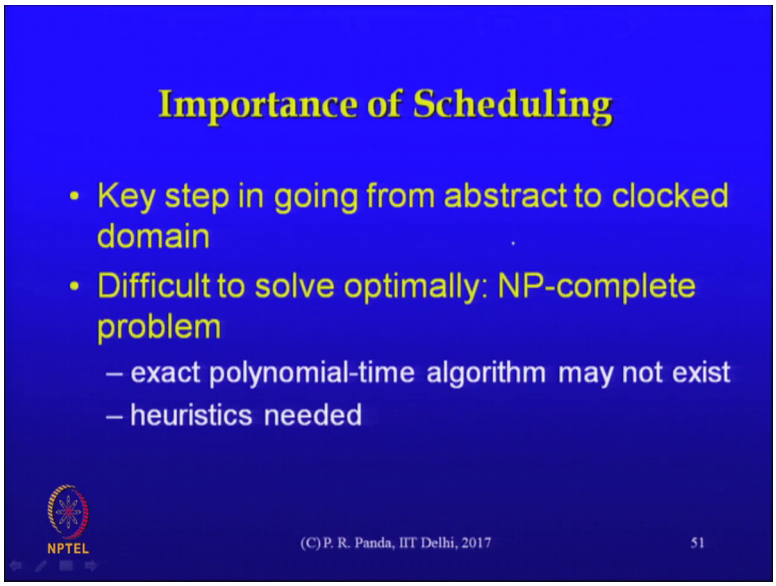


The point of scheduling is to take us a little closer to the hardware domain as opposed to that CDFG which of course, is purely functional in nature there is no timing related information there. Closer to hardware means that first let us introduce some timing partitions and the result of scheduling visually is just this these partitions are identified.

This is an important change in abstraction of that representation, because we are saying that not only is this the functionality, but also we are saying that these 2 operations will be done in the cycle; this one will be done in the second cycle this will be done in the third cycle right. That is essentially what takes us down from the behavioral level of abstraction into a more concrete register transfer level of abstraction.

Because we are explicitly saying what should happen? When everything is still not decided specifically, which hardware unit is going to perform, which operation that is not clear? But we have at least made the decision that these 2 operations will be performed in the same track cycle it is still a major step taken towards the implementation.

(Refer Slide Time: 28:14)



This of course, is a key step in going from an abstract to the clock domain.

The idea of a clock was introduced only here until then there was no idea of a clock. This is a classical problem scheduling as you can imagine this is a synthesis manifestation, but overall the scheduling is a general problem that applies in many different contexts. In all of those let us say the application could be airline scheduling or job scheduling on a multiprocessor and so, on a very large number of applications. All of them would have something in common with respect to the representation of that problem in the form of a graph that is generally how it is represented.

In the form of dependencies tasks dependencies between the tasks and any constraint or deadlines that you might impose within which the scheduling algorithm should work, let us introduce the formalism here by capturing our design in the form of a graph and ask questions regarding what the schedulers should do subject to what constraints we are imposing on it.

So, this is a known NP-complete problem, it is difficult to solve optimally, it just means that an exact polynomial time algorithm might not exist.

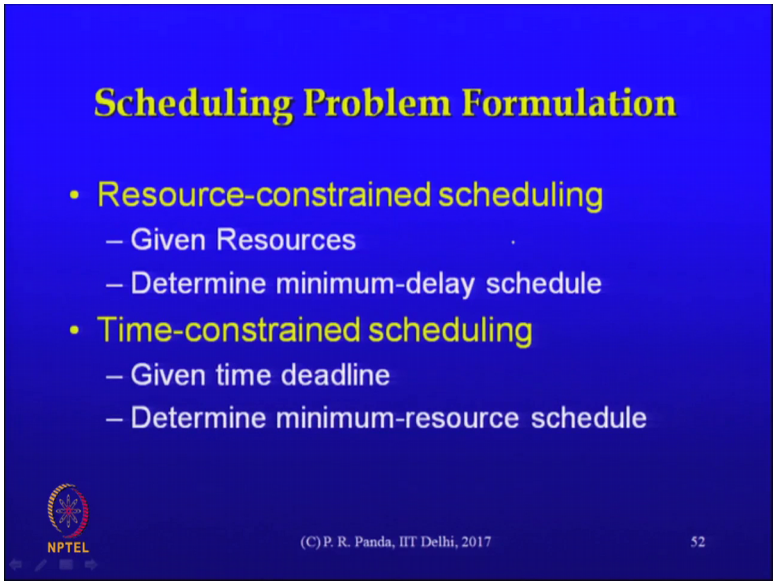
Student: (Refer Time: 29:36)

However since it is a practical problem it means that we still need a solution even though we are not making guarantees about it is properties. So, that brings us to the domain of a

heuristics from as an engineering simplicity point of view, one can try out several different heuristics they may have different complexities with respect to time. And an appropriate one has to be chosen based on several different situations like the size of the graph and so on.

But it is an important problem.

(Refer Slide Time: 30:14)



Generically one can formulate the scheduling problem in many different ways; one could be a resource constraint scheduling. Which means that resources are given, but you have to generate a design within those resources of course, that CDFG is given what we are scheduling here is actually just a data flow graph? Not even the overall CDFG. We did say that the way we will handle the CDF G is we will schedule the data flow graphs individually one at a time.

So, it is just a data flow graph that is given it is a dag no cycles even, but that does not make the problem any easier. So, resources are given, you determine the minimum delay schedule considering the DFG is given, the resources are given, the latencies are given of all of those operations right.

So, those operations have to be mapped to the resources. So, the resource library consists of elements whose delays we should know. Whose area also we should know, but we are

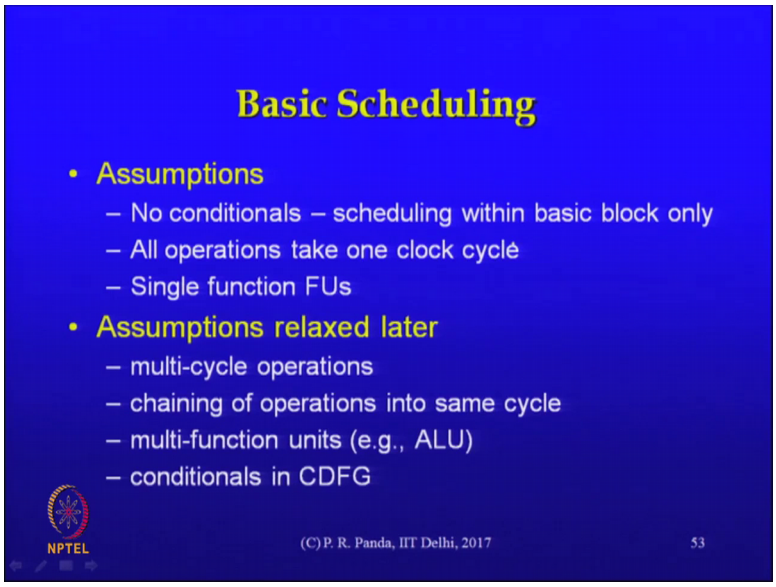
saying that in this form we will define the resources, fix the resources and you say that use these resources and give me the minimum possible delay schedule.

That is one way to formulate the problem you could also go the other way around, you do not fix the resources, but you fix the time in one you fix the resources and ask it to generate the minimum delay schedule, you could also say that the delay is fixed the deadline is provided and within that you just generate the minimum resource schedule, which one to take that depends on the design scenario.

It depends on what the constraints are and what you are trying to optimize? If the constraint is that you need a solution that must fit within a certain area, then you might go for the first version, where resources are constrained and the objective of a scheduling is to just minimize the delay to the extent possible. It could also be that your application is such that the result must be generated within a certain amount of time, that is part of the functionality we might consider it.

So, then we will just fix the time and the objective of the optimization then is to minimize the resources. Both are difficult problems it is just which one is applicable depends on the design scenario. So, we will first look at the version, where we make a lot of simplifying assumptions.

(Refer Slide Time: 32:48)



First of all there are no conditionals we are scheduling only within the basic block right, but of course, we know how to extend that solution into multiple basic blocks. So, at least a simple way to do it is you schedule each of the basic blocks separately. So, that is fine there are no conditionals and therefore, we are not talking about a CDFG to be scheduled. Although we will relax that later on, but that is the assumption that first we start with just a basic block which means just one DFG needs to be processed.

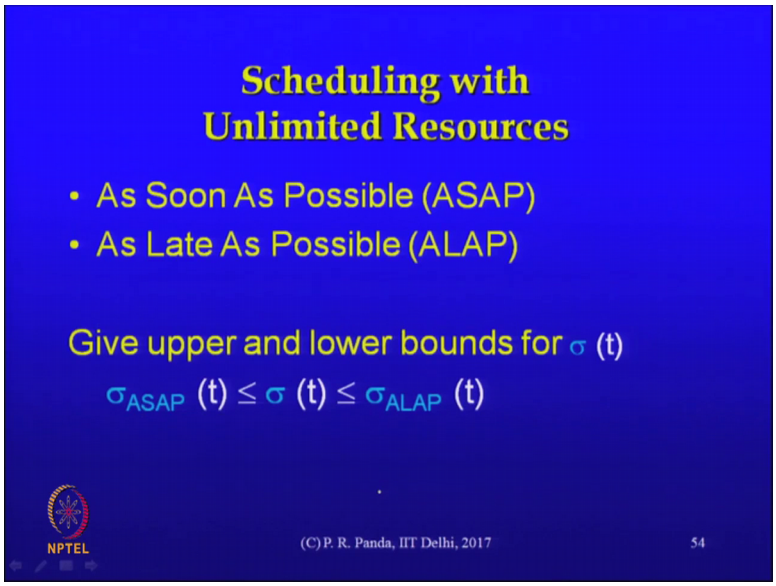
Let us assume that all the operations take one cycle, that is just to simplify the presentation of the algorithm it does not make any difference. The problem is still difficult even if you assume that all operations take one cycle. And let us assume that there are single function effuse meaning that you do not have ALU kind of functions that are capable of performing both an addition and subtraction and a bunch of operations.

We can easily relax all of these constraints later on. Once we define the basic strategy, we can relax this one certainly into multi cycle operations. It is a simple extension we can have chaining of operations into the same cycle means that, there is one operation dependent on another operation right.

Both of them could be scheduled into the same clock cycle if enough time was there you could have multifunction units. You could have ALUs that are capable of performing more than one operation type you could have conditionals and so on all of these are generalizations that take us in different directions.

But just to simplify the algorithm let us just make these assumptions for now.

(Refer Slide Time: 34:36)



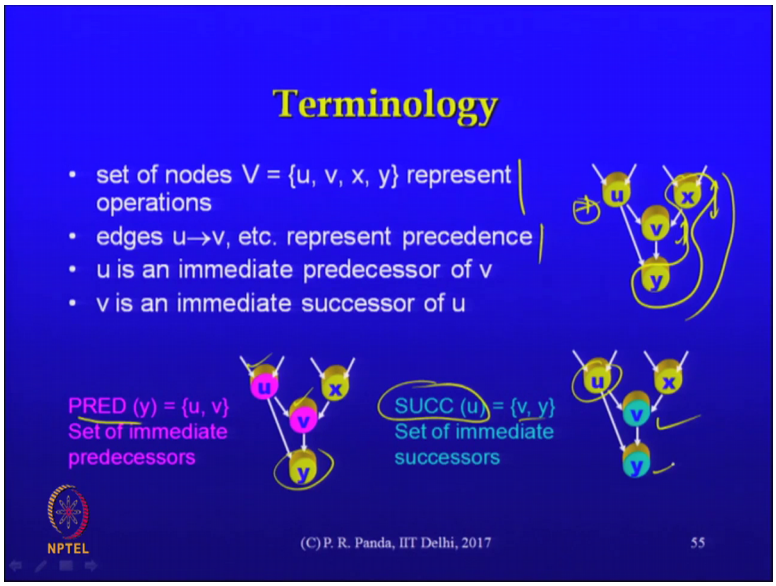
There are a large number of possible heuristics to solve scheduling; knowing that an optimal solution is not easy. You do need a solution soon it is the kind of problem where you can not afford to spend too much time, because the algorithm is applied possibly on a graph that has a very large number of operations.

So, the solution should be scalable in the sense that if you have thousands or tens of thousands of operations in your process, then we should be able to come up with a reasonable quality solution. So, the algorithm that we will look at is one variation of a number of different strategies you could take to perform scheduling, but this is chosen because it illustrates several principles along the way.

That leads to first a computation of an as soon as possible scheduling and as late as possible schedule that is the ASAP and the ALAP, this is in the final schedule in strategy.

But they are useful nevertheless because they result in some bounds and upper bound and the lower bounds for what is the earliest, that a particular operation can be scheduled at and what is the latest particular operation can be scheduled at without violating deadlines and so on.

(Refer Slide Time: 36:12)



So, let us define the terminology in terms of which we will present the solution. So, that is my DFG as you can see the application here is the scheduling context of high level synthesis where these operations are our arithmetic kind of a operations plus minus and so, on.

And what do these edges represent in the graph they represent precedence or dependency right the V node can be scheduled only after the u node is completed, because it is using the result that is computed in u, you can see that this is a very generic representation of this problem statement.

The representation would be the same even if you change that level of abstraction completely and because of which the problem statement might completely be different let us say this becomes a task graph. In a multiprocessor context u,v these are not simple operations, but each is a program this is a task to be scheduled. And the program u is producing some data that v is dependent upon. And scheduling's representation still does not change this graph is still the input to the scheduler, it is just that each is associated with a delay right x has a delay v has a delay and so, on there is a latency that is associated with each of those.

Those would change and of course, what u is doing? What x is doing? These are not the business of the schedulers' strategy. The schedulers just needs to know the dependency structure which is captured in the graph it also needs to know the latency of these

operations u v how long each of them takes so, that the appropriate schedule can be generated.

You can see that if this were multiprocessor task scheduling the formulation would still be the same. And your solution might actually still apply there of course, the meaning of the tasks is different the meaning of the resources would also be different here our resources are alus and multipliers and so, on. There what would the resources be in a multiprocessor task scheduling the task would be a program resource would be what?

Student: (Refer Time: 38:31)

The processor well it could be lots of things, but essentially you have a set of CPUS and their associated memory hierarchy and so on, but you have to take a decision of out of all the tasks that are there in this flow graph, which ones should be mapped to which processor at which time.

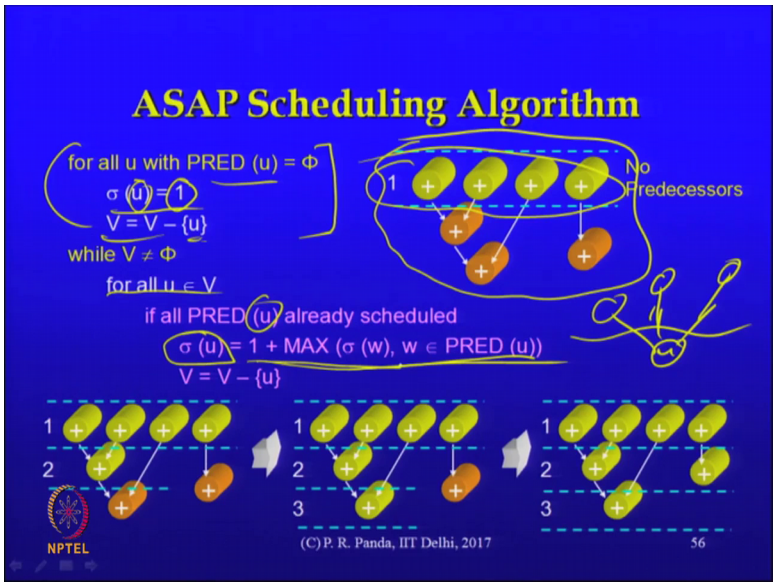
So, it is a this similar problem, you could cast many other scheduling kind of problems according to the formalism like this, where you just capture it in the form of a graph where a node represents the computation.

But really we have made no assumption about what kind of computation it is? It could be very simple it could also be a complex, but the edge represents dependency which means that if the reason edge from u to v, then v b is ready to be scheduled only after u is complete.

So, that is our set of nodes in the graph edges represent precedence or dependencies, I say if there is an edge from u to v, then u is an immediate predecessor of v and v is an immediate successor of u. Why immediate this is as opposed to the relationship between x and y where x is the predecessor of y, but not an immediate predecessor because there is no direct edge from x to y.

I can define this set PRED of y to be the set of immediate predecessors of a node y, here that would be these 2 nodes, because they have outgoing edges that landed y. Same thing with successor set that is the set of immediate successors of a node u. So, for that node the set of successors are these 2 immediate successors are these 2.

(Refer Slide Time: 40:34)



With that we should be able to come up with a simple ASAP scheduling algorithm that algorithm might look something like this.

The idea could be the following that is our DFG for ASAP scheduling assume that there are an infinite number of resources. If there is the case what is the earliest that a particular node could be scheduled how do you find this out? Let us answer this question of which nodes could be scheduled in the first clock cycles.

Student: (Refer Time: 41:07).

All of the nodes that do not have any incoming edges, because they do not depend on anything we can collect all of them and schedule them into the first clock cycle. That should lead to an obvious strategy for how to do ASAP? After identifying these nodes how do you continue with S M?

Student: (Refer Time: 41:25).

Yeah just remove those nodes from consideration and apply the same strategy.

Student: (Refer Time: 41:30).

Once more right so, that strategy should be simple we could write variations of these algorithms, but essentially it could be that I first pick up the first cycle in this way for all the nodes with no predecessor no immediate predecessor I just schedule. So, the sigma is

just the schedule number it is the clock cycle value that we are assigning to a particular operation u. So, sigma of u is one means that all of them are going into the first clock cycle.

Then I just. So, having considered u already I remove u from the set of nodes under consideration fine. After that all I need to do is I trade through the remaining nodes, we have already deleted a bunch of nodes from that set v. For all the remaining nodes let me just iterate through and make the following check that I will check whether all it is predecessors' immediate predecessors have been scheduled or not.

How do I know of course, I initialized that schedule to some something else a negative number or something like that. If it is assigned some positive number like one or anything else, then I will say that if all the predecessors of u are scheduled, then the u itself is ready to be scheduled. And I can assign to u a clock cycle, which is what I have said 1 plus the max of the schedule value of all it is predecessors saying that if u is dependent on a number of nodes.

So, that is one that is one each of all of these guys have been scheduled already, then we are just saying that it is one plus whatever is the max what is the reason for this.

Student: (Refer Time: 43:39) the last scheduling third cycle (Refer Time: 43:41) of the third cycle.

Yes.

Student: (Refer Time: 43:43) next cycle.

All it is predecessors might not have been scheduled in the same clock cycle they might have been scheduled in different clock cycles. So, the earliest that you can schedule u is the cycle after whatever is the latest that any of it is predecessors have been scheduled.

Student: Sir why did we have to do a separate block of statement for the first one, then the initial what is say source nodes of the graph? We wrote some specific lines and then we are using this algorithm is where (Refer Time: 44:17) first point itself.

Yeah it is not valid simply because well if you use this computation 1 plus max of the predecessors yeah, what does it mean? If you do not have a predecessor you have to

appropriately define it that is all it is, you could write the whole thing in one loop taking that into the initial condition also as part of that computation.

So, with this you are not necessarily identifying the nodes 1 level at a time, but you would get a result which is the same what we are doing is. So, let us say the first level has been identified then I moved on to that node all it is predecessors have been identified. So, that is all right I can assign a node a cycle to this.

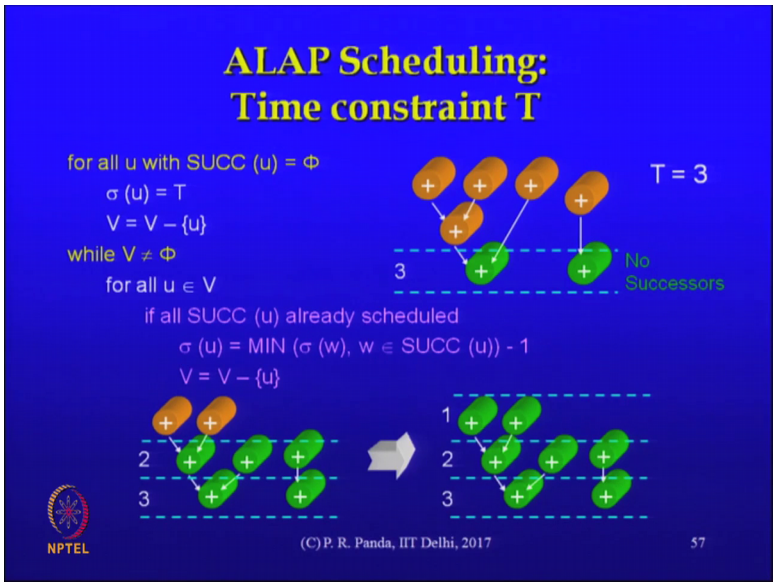
Following that let us say I came here actually I did not say anything about the order in which you traverse that loop I just said for all u belongs to v, and actually you can see there is there could be an efficiency issue in the order in which.

Student: Large (Refer Time: 45:22).

So, yeah you may need to revisit that or refine that according to some of the other logic remember that we have already looked at, but it does mean that there are multiple orders and the number of iterations that you would go through might actually depend on the order in which you are looking at it although the result will not change.

So, if I look at this one next after this then to I am at a situation where all it is predecessors have already been scheduled. So, I can derive the schedule for 3. Even though this one is not yet done this would go into the second cycle, but I pick this node to process next and. So, I should be able to just find the schedule for that. And then that leaves this and that can be handled by just noticing that it is predecessor has been scheduled. And we the earliest that we can schedule this is to that is my ASAP scheduling.

(Refer Slide Time: 46:20)



Correspondingly we can do an ALAP scheduled they all the idea is analogous, but we are trying to find out the latest that you can schedule any node, given the same graph what is the latest that we can schedule any node. The strategy somehow would be similar, but we go in the reverse direction, but how does one go about it the earlier algorithm was that we look for nodes with no incoming edges. If you are looking at as late as possible how do I go about it?

Student: (Refer Time: 46:58).

You can start with nodes at nodes that have no outgoing edges, but where do you schedule. It ASAP was clear you schedule those nodes that did not have dependencies into the first cycle for ALAP.

Student: So, that they will go ahead that is given.

ALAP means a constraint right. So, otherwise then you could postpone it forever and therefore, it has to operate with a time constraint. So, this time constraint let us just impose the third cycle as the timing considered, which means that the deadline for performing this is 3 where did 3 come from maybe we first did an ASAP and then realized that that the length is 3 ASAP found a schedule with 3.

So, it certainly means that you cannot find a schedule that is shorter; than 3 right are you convinced that if ASAP gives the number for the number of cycles that it took it is not

possible to find a shorter schedule. Of course, that it had better be if we are saying it is the name itself says as soon as possible, you should be able to prove that you cannot find a shorter schedule than that.

We can take that of course, it need not be 3 it cannot be less than 3, but it could be 10, but if it is 10 as opposed to 3 what happens to the resulting schedule. Maybe we answer this question later on let us go ahead and solve this, I identified the nodes with no successors and assign them to the last whatever is the deadline we assign it to them then.

Student: We will delete that.

Those nodes are removed and also the cycle is decremented and then I. So, the deadline is updated and then I again run this actually. So, this is similar. So, I identify which are the nodes that would go here. So, that node has it is dependencies takes. So, these 3 nodes are selected and let us say in this into the second cycle and the other 2 nodes are selected into the first cycle.

We went in reverse order the algorithm would also be very similar that initialization step would now be applied on the nodes that have no immediate successors. And that t is what is specified there as the cycle into which we are scheduling it. Removing those set of nodes from consideration then I run through the same logic, but for all of those remaining nodes, what we are checking is if all these successors are already scheduled. Then that is the logic does not make sense you take the minimum of the clock cycles into which we have scheduled all it is successors and subtract one from it.

You see this is the reverse logic right and you remove that node from consideration and continue in the loop. Both of these are relatively straightforward these are not difficult like this is the extreme, where the assumption was that there is no resource constraint. So, obviously, this is just an ideal schedule it does tell us something though, some valuable information could be inferred by just doing a, an infinite resource scheduling like the ASAP and ALAP where even in the context where resources might be constrained in some way.

What information can be derived from here? These are ideal schedules remember they made the assumptions that we are violating our constraints in terms of availability of resources, but still what information could be derived from here that is useful.

Student: (Refer Time: 51:07).

Yes it bounds for us the range in which the schedules of individual nodes can lie.

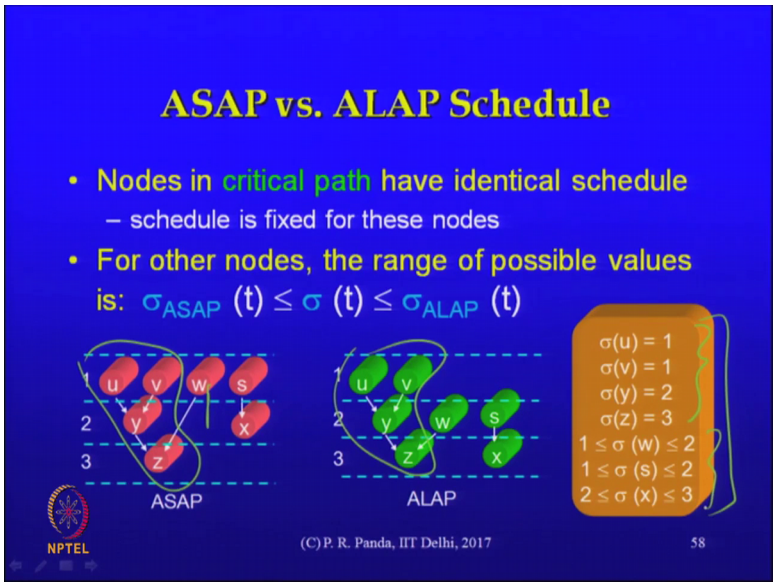
Student: (Refer Time: 51:21) why should I give 3 I can even give 0 I will get negative number then I can offset it back I mean I do not I need not give any number (Refer Time: 51:30).

Yes look at why we are doing it this way there is a reason, if you were to do just ALAP scheduling then you do not need the cycle, but with what we are going to do with this result you do need a constraint like.

Student: If I am taking my entire (Refer Time: 51:47) I can always (Refer Time: 51:49) back to the (Refer Time: 51:50).

Yeah, but why do not we look at what we are doing this information.

(Refer Slide Time: 51:55)



So, let us compare the ASAP and the ALAP schedules ASAP schedule look like that a lap schedule look like this, if you compare you will find that there are some nodes for which both the ASAP and ALAP schedules are the same. For this set of nodes, they are the same, but there are other bunch of nodes for which they are different. And here is one inference you can make that these are nodes for which the schedule is fixed irrespective

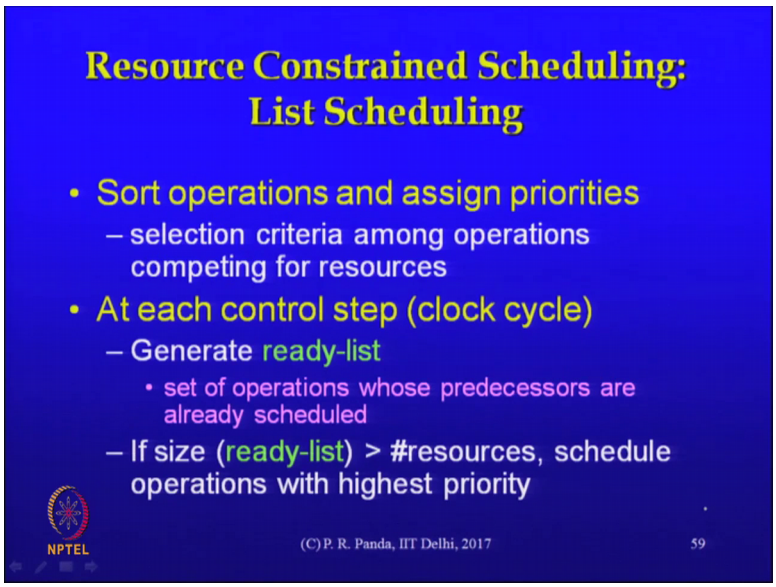
of whether it is a ASAP or ALAP are the nodes that we will say fall in the critical path right.

But the there are these other nodes for which it seems as though there is some flexibility. Even if the deadline was 3 then W I do have the choice of scheduling into the first cycle or the second cycle, x and s are also similar. All of those nodes are not necessarily as critical as the uvy z for example, is that information is valuable information that could be used in a realistic scheduling algorithm is.

So, our bounds are these for u v y and z these are the nodes that were critical and for these. In fact, the schedule is fixed right, but for these other nodes we have defined the ranges as of now we have may not made the decision, I do not know the resource strange yet, but even without the resource constraint. We have classified the nodes into nodes that are in some ways more urgent to schedule or more critical versus those that are less critical, that kind of information is inferred from the ASAP and the ALAP schedules.

Even though they have made assumptions which may be unrealistic, but still these are properties of the graph and that information that is derived from here could be used in some ways by us.

(Refer Slide Time: 54:07)



How we will use it we will just get to, but let us keep in mind that we have already done that kind of analysis that is simple to do both the ASAP and the ALAP how simple is it just get back to that algorithm here.

What would be the complexity of that ALAP scheduling algorithm as also the ASAP scheduling algorithm this is the same it is go to the ASAP instatement right. How about that initialization step how much time does it take?

Student: (Refer Time: 54:44) that is to find out if it is a source or not.

This is a loop right the body is a constant amount of work that is happening in the loop. So, somehow everything is happening here how much work is happening there.

Student: (Refer Time: 54:59).

For all the nodes for which the set of immediate predecessors is null to do that you have to go through all the nodes.

Student: (Refer Time: 55:12).

For each of those nodes how much work do you need to do?

Student: (Refer Time: 55:15).

All the incoming edges.

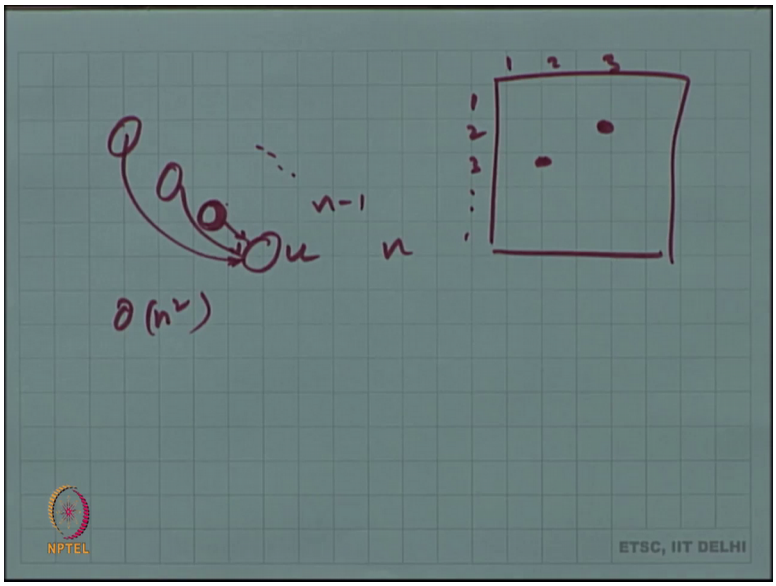
Student: (Refer Time: 55:18).

Have to be identified you can see the, what is the worst case? So, worst case complexity just for that initialization.

Student: v plus e order of v plus.

Order of v plus e why.

(Refer Slide Time: 55:36)



Student: (Refer Time: 55:35).

Yeah let us say I to simplify this, let us just make it on the of the order of if there are n nodes then what is the complexity, worst case for the initialization.

Student: Order of n (Refer Time: 55:54).

Order if n and one node you can have how many predecessors that is my let us say the u was here.

Student: N minus 1 predecessors.

It could have all of these guys can be we are looking at the worst case example right so. In fact, you could have n predecessors in the worst case, you can have what about this node the directed edges of course, that one could have how many predecessors. So, could have. So, what is the complexity just of that initialization is n square, because for this I have n I for this maybe there are n minus 1 and so on now 1 plus, 2 plus, up to n and so on. So, that could be an n square complexity in the worst case of course, realistic graphs may not look that way yeah.

Student: Sir we are not looking for all the intervene edges, we are just looking for the nodes which have no intervene edge we are looking for the one which has no.

(Refer Time: 57:04).

Student: Right. So, we did not go to n minus 1 times the same body (Refer Time: 57:09).

Where this is the good point it depends on how you have represented it? How do you represent the graph?

Student: Sir (Refer Time: 57:17) about that how do I represent the graph as in like we have a simple graph.

Well here is a representation node one node 2 node 3 and so, on 1 2 3.

Student: (Refer Time: 57:29).

Right each element here tells whether there is an edge from 2 to 3 right that element says, whether there is an edge from 3 to 2.

Student: (Refer Time: 57:39).

So, if it is a matrix representation then finding that you have.

Student: (Refer Time: 57:45).

Even one predecessor or not does require a traversal element.

Student: (Refer Time: 57:47).

In most cases if it is a list representation it can be better that is up to you.

Student: (Refer Time: 57:53).

You just need to know whether the list is empty or not.

Student: (Refer Time: 57:56).

Yeah and fine.

Student: (Refer Time: 57:58) so, in that case we have order of n only because then for everyone we just have to find out (Refer Time: 58:02) or not.

Yeah you could do.

Student: (Refer Time: 58:05) order of n c.

Right what about the next part this part let us get to the body itself, how much time for this just the worst case 1 plus. So, this is simple this part is simple whatever the max max of all predecessors, would involve traversing all the predecessors. So, how many predecessors can I have?

Student: (Refer Time: 58:42).

I can have n predecessors. So, this max the loop body itself actually might take or run time.

Student: (Refer Time: 58:53) yes we are doing that for every (Refer Time: 58:56).

Here to you can try constructing the worst case the body takes and I have taken node u and I am checking if all the predecessors have already been scheduled. Actually that even though we have written it in this way the max can be computed simultaneously with the traversal of all the predecessors. Otherwise if you arrive at the loop body and then perform the max by traversing the predecessors again then it seems as though you are doing extra work.

But essentially all of these guys and all of that computation could be done together yeah. So, if you find that one of those nodes is not scheduled then it is already you have to about the process and then you just go to the towards the next node.

Student: (Refer Time: 59:50).

So, let us say order n work is done for this. Now getting here for that loop you are going through all the nodes well maybe all the node not all the nodes, but remember you are removing nodes as and when you are processing it. So, that the number of nodes you are processing could actually reduce as you are going through the loop. So, how much time for this this you could actually be going through n nodes in the worst case and; that means, that this itself is n square as of now.

Because this was n that could be n square, but everything is not resolving in one iteration of that outer while loop depending on your example; your worst case might actually lead to only one node being resolved in every iteration of that inner loop, which leads to an n cube kind of complexity. You can try and construct a data flow graph for which it is actually the worst case, but we should understand where all the complexity comes from.

Student: Sir.

What we have done. So, far this is a preprocessing we just did the easy thing first, which is doing an ASAP schedule and an ALAP schedule and the objective of doing that was that it revealed to us, what was the range we really have or in some ways it prioritize the nodes for us. Some of the nodes are more important to schedule, because if you delay some of the nodes for example, in the critical path as you do the actual schedule.

Then all it is subsequent nodes would get delayed on the other hand if you delay some node that is off the critical path right for which it seems as though some range was there. Then you do not necessarily immediately incur a cost maybe it can be adjusted later on because some flexibility was there, some nodes it is more flexible other nodes are less flexible that is what is the basic information that one derives from the ASAP and the ALAP. If you do both of them and you subtract one from the other if it seems that that number is not 0 then there is some slack if it is 0, then yeah less slack right and let me stop here.