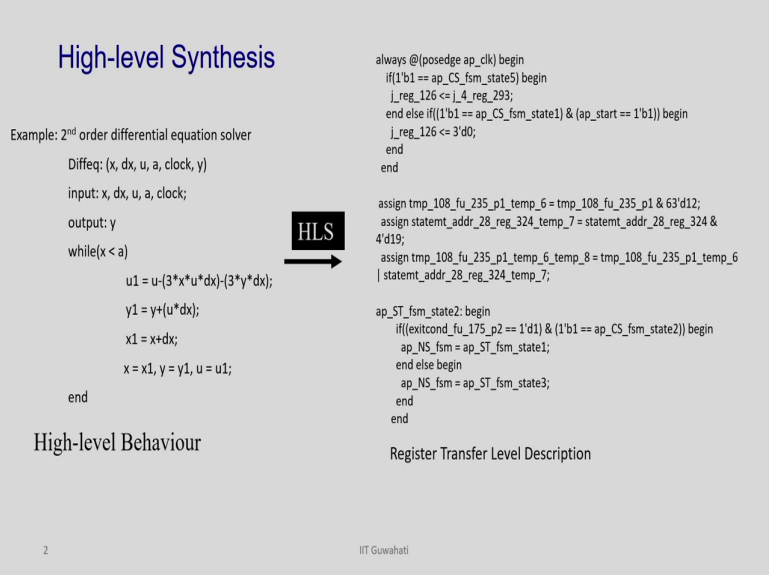
C-Based VLSI Design
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 02
C-Based VLSI Design: Scheduling
Lecture - 05
Scheduling Problem Formulation and ASAP and ALAP Algorithm

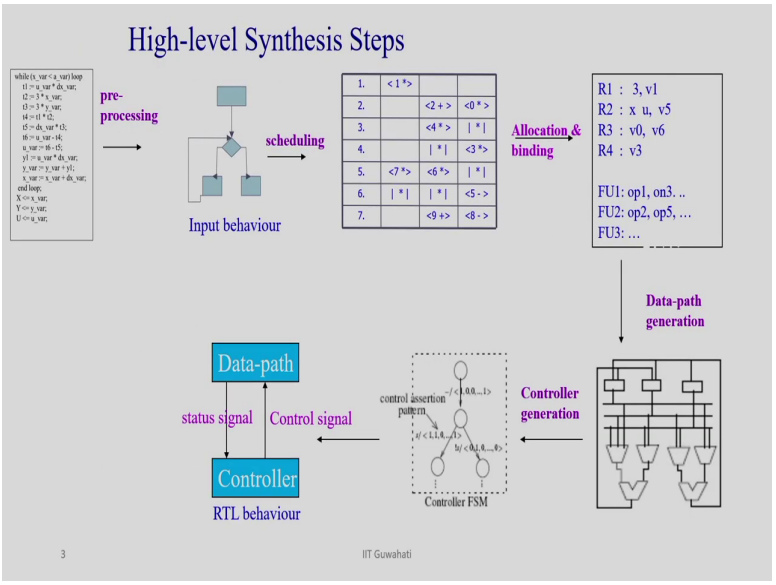
Welcome everyone. In today's class, we will try to introduce the Scheduling Problem.

(Refer Slide Time: 01:01)



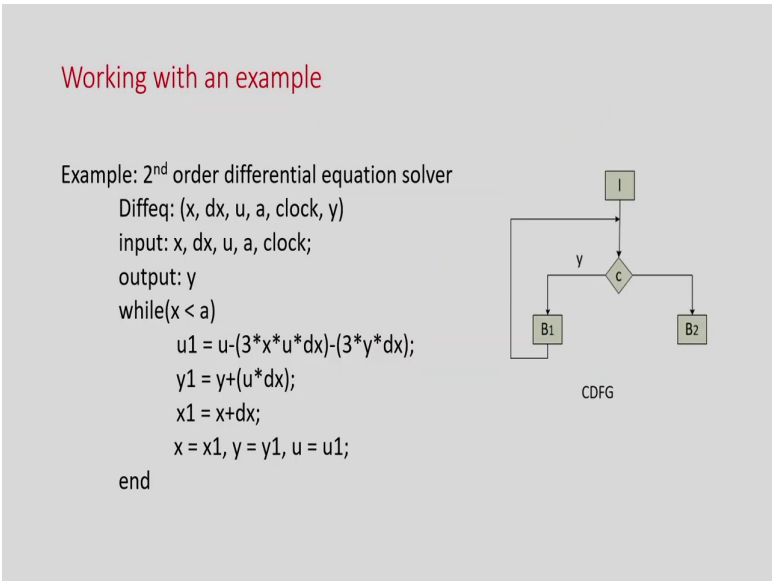
So, we have already seen that high-level synthesis is a process of converting a high-level behaviour detail in C or C plus plus into an equivalent design in RTL level.

(Refer Slide Time: 01:12)



And we have also seen that this high-level synthesis is actually consist of several sub steps and the sub steps are basically pre-processing, scheduling, allocation, binding, data path controller generation and then finally, we will get the RTL. So, what we are going to do is today we are going to take this schedule steps and then, we try to formally define what is the problem here and what are the variations of the problem and that we are going to discuss in today's class.

(Refer Slide Time: 01:38)



So, we have already discussed that once we have given a behaviour, it is basically it has a control flow and data flow graph will be extracted from the C code and then, it is basically you have set of basic blocks in your program and there is a control flow among the basic blocks. So, for the loop there will be a backage, for if else, there will be a branch and so on and basic block is basically a sequence of operations whether there is no control branch, there is no control flow it is a sequence of operations.

And the way we planned the whole high-level synthesis is that we will take one basic block at a time and try to schedule the operations there, we will try to identify the resource required for that particular block and then, we will try to find out the interconnections and so on and then, what we can actually do is basically you try to find out the operations that are across basic block, you can also share the resource.

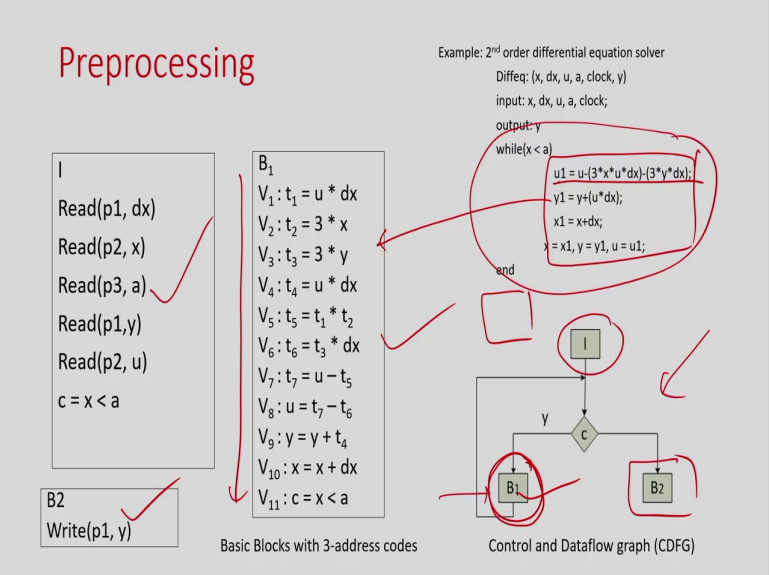
But scheduling, we can do independently of these blocks because they are non-overlapping. So, what we can do? We can take a basic block at a time, we will schedule the operations there and I can take one basic block at a time and schedule all those things.

And during allocation and binding, when you are going to find out the number of FUs are needed, how many number of register needed, what we can do is we can actually not only find out the resource we got from one block, but also we can share this resource across basic block also because operations into basic block if they are in series, we can share the registers even if they are in branch.

So, although they are looks parallel, but they are in mutual exclusive branch, then also I can say use the same resource for both the branches because I mean at a given time, only one of the branch will execute, so, not both the branches will be executed.

So, these are all detail we are going to discuss in the classes, but the bottom line here is try to say that we can take one basic block at a time and then, within the basic block, the operations are there and we want to schedule them.

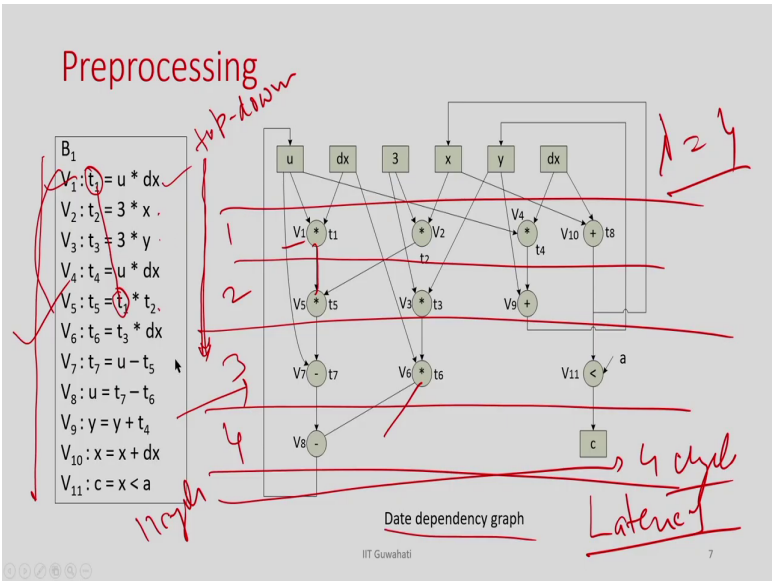
(Refer Slide Time: 03:36)



In the example that we have taken here is the Differential equation example and we have seen that for this example, we have the control flow graph is this and this basic block is basically the reading the inputs, this basic block is the loop body, this is the loop body where these are the operations are there and these basic block is just writing the output. So, which will happen here.

So, this is how we extract this, and the control flow is known and within the basic block also, we have discussed in previous classes that these operations are too big expressions executing them in a single clock is something is not a good idea so, better to break this operation into three address form or unit operations and then, we try to schedule that.

(Refer Slide Time: 04:25)



So, let us try to go more into that these scheduling steps. So, in C, if you have said this operations with you. So, what I am going to do is here, I am going to take these basic blocks and then, I am going to discuss the scheduling on the context of the operation given in this basic block, the similar thing can be done for other basic blocks of the program.

So, now, let us try to understand what is scheduling and what is the objective should be here. So, you have set of operations, there are 11 operations here and if you give this into, I mean if you run this program, say only these basic block using your processor, what is going to happen?

It will execute these operations, and then it will execute this and this and this. So, this is a top-down manner, it will be executed in top-down manner. So, what is the advantage of or disadvantage of this? Advantage is that because we are executing in the top-down manner, the dependency is already which taken care of.

So, for example, we have already discussed this part that operation t1 is getting defined here the variable and it is getting used here so, you cannot execute this operation before this operation. So, but if you were execute them in this top-down manner, automatically the dependencies, read after write dependencies or write after read dependencies are automatically taken care of. So, that is something is advantage.

So, now in scheduling, what we try to do? We have 11 operations, I want to schedule them. Obvious solution that we can think of, you use 11 clocks to do this. So, in first clock you do this V1 exactly in the top-down manner you execute it, in first clock you do the operation V1 and then, second clock operation V2 and so on.

Obviously, this is the correct solution because this is how these things are going to happen in the processor, I am going to do this sequentially, but in hardware, we try to improve the performance. So, the question that I am going to ask now myself that, is it 11 steps is necessary here? Can I have fewer steps to execute this? The answer is yes that the answer is yes because yes, I can do by parallelizing such certain operations in the execution in the hardware.

So, what does it mean? It means that once we execute the operations, it is not necessary that you execute them in one by one rather you try to parallelize the execution of the operation. So, that means, you try to execute more than one operations at a time. Obviously, what are the operations can be executed in parallel, the operations that have no dependencies?

So, if there are two operations which is independent to each other, I can schedule them in parallel and that is what the scheduling objective, you try to find out the maximum parallelism within the behaviour because the behaviour is a straight line of code. I mean it is without further analysis, it is not possible for you to identify the parallelism and for that, we discuss in previous classes that you draw the data flow graph, data dependency graph.

So, in the data dependency graph, you identify the dependencies, and you draw the graph. So, in the data dependency graph, each node represents one operation. So, this V1 is representing these operations and so on and then, if there is a dependency between two operations just like this V1 and V5, you add an edge so, that V1 to V5 there is an edge. So, wherever there is a dependency between two operations, you add an edge in this diagram and then, this gives the dependencies.

And once you try to parallelize or you try to schedule the operations, you try to schedule the operation in such a way that it does not violate this dependency. So, example you cannot

execute this V1 and V5 in the same time step ideally because then, there is a you cannot store this value t1 intermittently.

So, there is a option of operation changing, I am going to discuss later, but in general, if you want to execute one operation at a clock, this V1 and V5 should not run in parallel because V5 depends on operation V1 and so on. So, in schedule, now, I will try to find out maximum parallel number of operation and I will try to schedule them.

So, if you take this behaviour I mean from the diagram that I have shown, even one possibility is that you execute this V1, V4, V5, V1, V2, V4 and V10 in time step 1 and then this V5, V3 and V9 in time step 2 and this in time step 3 and this is time step 4.

So, we can understand that for this behaviour, instead of 11 cycles, there is a solution which can take 4 cycle. So, I can understand that there is a huge improvement. From 11 cycles to execute this body, I can execute the things in 4 cycles and that is what scheduling does.

So, given a basic block or series of program, it try to identify that operations, the number of time or the cycle you need to execute this behaviour and that is called latency. So, it try to identify the latency of the program which is the number of time step. So, for this example my lambda or the latency is equal to 4. So, to execute this behaviour by this schedule, I need 4 cycle. So, the latency of this schedule is 4 ok.

And also, the important thing is it is not about that identifying only the number 4, you have to say this operation V1 is happening in time step 2, operation V6 is happening in time stamp 3 and so that means, that start time of the operations also is get determined so, where this operation is getting executed, starting is execution ok. So, that is something is also determined.

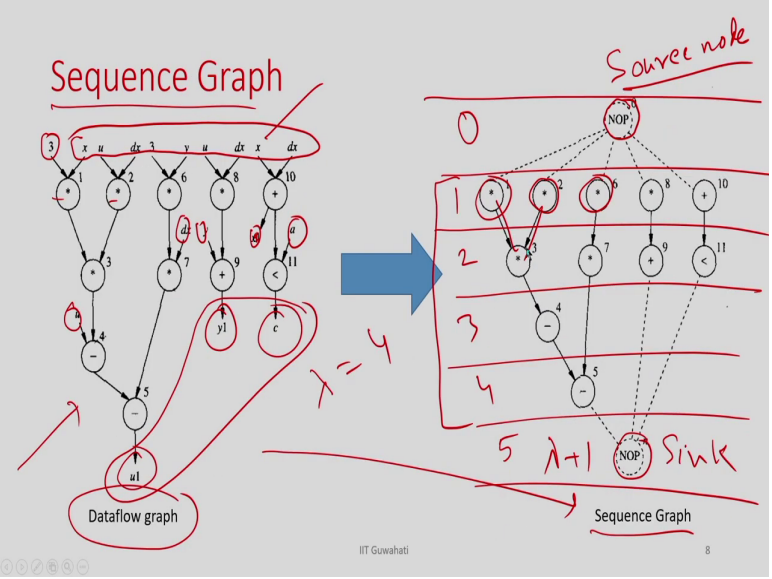
So, schedule given a program or given a sequence basic block or sequence of operations, it determines the total latency of the design that means how many clock cycles I need to execute the behaviour and also, the start time of each program, each operations using the basic block.

So, that is what is scheduling. So, that is the overall thing, but there are maybe very many variations of scheduling, it is not that its shows as forward, you do it in always in 4 cycle,

there may be many constraint for which it may not able to do it in 4 cycles, you probably you need say 7 cycles or 6 cycles and so on instead of 11 cycles, but that is something the variations of scheduling, but the basic intuition is just to identify the total latency and the start time of each operations within that latency limit.

So, that is what scheduling does. So, this is something is the overall scheduling problem.

(Refer Slide Time: 11:31)



So, let me try to understand now, what is the input, exactly input for the scheduler. So, the first input is the sequence graph. So, we have already discussed the data flow graph, which represent the data dependency. So, for scheduling, what we need? The internal dependencies. So, the inputs so, you can assume that these are the inputs so, these are the all inputs and these are also inputs. So, these are all inputs not this one, this one. So, these are all inputs.

So, when I am going to schedule, I am not bother about what is exactly the value of the inputs, we only know that this is an input and input is always available you can assume because at the start of the this basic block, the inputs are available. So, what I can do? I can just simplify this sequence graph into sequence graph.

So, this is that this data flow graph, I can simplify this data flow graph with sequence graph by just removing all these inputs and I putting a single node. That is called source node. So, source node is something is the node which represents all the inputs.

So, I can just merge all these inputs into a single node, and I will just made the single node. Similarly, a program may have multiple outputs and there are multiple outputs. So, again I do not bother about what is the output and what is going for during the scheduling.

So, what I can do? I can merge these entire output nodes into a single node and that is called sink node, this is just to simplifying the graph nothing else, this is still representing your inputs. So, this data flow graph representing your input program. And what I am just doing?

I am just simplifying the graph because during schedule, my objective is to identify the start time of these nodes not the inputs and also, the total latency, for that what I can do is I can abstract out the exact inputs and all outputs and I can use a dummy node, source node and a sink node to represent the inputs and the outputs respectively. So, this is what I have done. So, this is just simplified the things that is all nothing else.

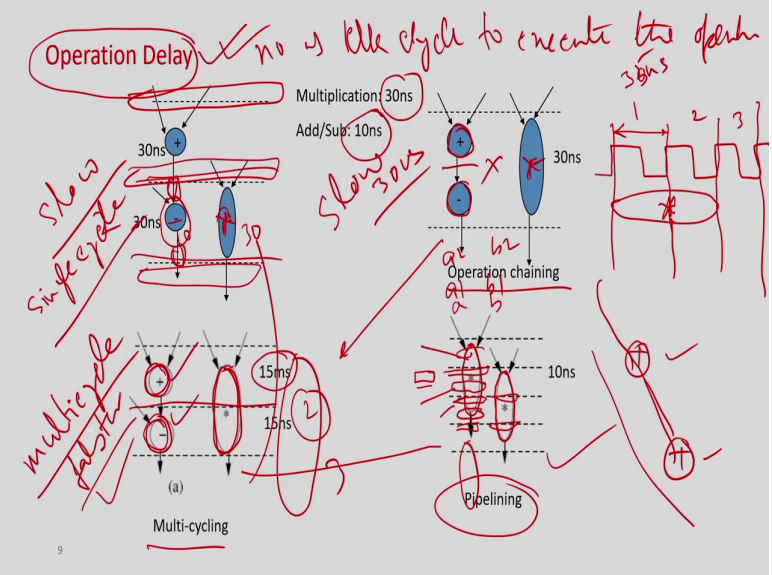
And the way we have discussed that this is my start time so, these are the actual operations, these are the actual operations so, what I can do? I can assume that the source node is always schedule in time 0 because I my times always start from 1, 2, 3, 4 and this sink node will be always schedule in the lambda plus 1 because that is not something is important for other so, my actual schedule is for this.

So, once I do this, I will assume that my lambda equal to 4 and the sink node will be always scheduled in lambda + 1 time step and this is always be schedule the source node will be always schedule in the 0^{th} time stamp. So, the actual schedule is from 1 to 4. So, that is the idea.

So, once we try to take this program and try to schedule, I will always schedule this node at 0 so, it has no dependencies and sink node is always be scheduled once the my schedule is when my essence of the time step of the actual operations are done, then I am going to schedule in the last time step because this is just the representative purpose.

It is just for an abstraction of this data flow graph to a sequence graph for a better representation purpose and deleting the unnecessary stuff which is not needed for my scheduling. So, my input behaviour which is a sequence of operations will be represented by a sequence graph that is my input 1.

(Refer Slide Time: 15:02)



The next input is the operation delay which I have not talked about yet. So, let me understand or let me try to explain this operation delay. So, in hardware, once we try to execute an operation, you need certain time because you try to do these operations in the hardware so, in that hardware, it has the set of transistors and all. So, and now, for each operation, delay is not unique.

So, for example, you will always understand the multiplier is a big operation as compared to adder because you can understand the multiply is lot of addition, lot of multiplication plus addition operations. So, obviously, an addition and a multiplier is not the same operation and I hope you have already tried to implement an adder which is in sometimes full adders and a multiplier at the gate level which is more complex operations.

So, the delay to execute the multiplier will be much higher than the adder, this is just an example. So, you have say shift operations, plus, minus, division, modulus, any operation is

possible so, the delay of these operations are not the same. Now, the question is how do you execute the various operations in the hardware? What are the possibilities?

So, you obviously, understand that in hardware, you have a clock so, you have a clock, it is coming continuously so, you know that this is one clock period and whatever the operations get, this is another clock period. So, this is clock period 1, this is clock period 2 and so on so, this is my clock period 3.

So, whatever the operations are schedule in this say in clock 1 that must be completed before the next clock comes so, that this is the clock period. The delay of that multiplier must be within this limit so, it should not go beyond this, then it, and so, your result will not be stable. So, you cannot get your correct results.

So, now if assume that the addition operation is 10 nanoseconds and multiplier is 30 nanoseconds operation so, to execute them in a single cycle. So, suppose you schedule a minus operations and a multiplier operation in a time step say time stamp 2 so, what and since the my multiplier is 30 nanoseconds, this clock period will be at least 30 nanoseconds, it should be little bit more so, say I 35 nanoseconds or so.

Because there are some setup time and hold time, but it should be at least 30 nanoseconds because otherwise, this multiplier would not complete its execution within the given clock ok. And, but the point here you try to understand that this is 30 nanoseconds, and this is 10 nanoseconds, but my clock is 30 nanoseconds.

So, what is going to happen? This adder is remaining idle for 20 nanoseconds. So, it just takes 10 nanoseconds to execute and then, it just remains idle because your clock is higher. So, it just waits for the next clock to come to do the next set of operations. So, that means, if you the possibility one is that you identify the operation which has the maximum delay and you set up your target clock like that.

If that is the solution 1 that is that is your objective, then for faster operation, it is a waste of clock you could have done three addition operations here instead of one addition operation. So, your clock becomes slow. So, in this option, your clock become very slow and it is the

and the clock period is determined by the delay of the maximum time taking operation and your all operations are single cycle.

So, single cycle means that multiplier is also complete in one cycle; adder, subtractor, safety shift safety, every left shift, everything will complete their execution in single clock. So, this is one possibility, but we already understand that my clock will be very slow and for the faster operator, it will remain idle for many times.

So, what is the alternative? The alternative is you make the slower things into multi-cycle operation. So, that means, now I have decided to do this multiply in two cycles. That means, the multiplier operation will take these many cycles, two cycles and my clock period is double, I make my clock is 15 milliseconds.

So, my clock is fast now, my clock is fasten now and now, because there is two clock, I can use the same adder to do two different operation within the timeframe.

So, this is where I just did this my multiplication operation or that operation, which is slower, I want to execute in multiple clock cycle ok. How to do that and the detailed architecture, we can go into digital design, but let us assume that this is possible. So, to execute the things in two cycles. So, this kind of design is available.

So, what I can do aside that the delay of the multiplier is now 2. That means, there are 2 cycle delay, it is not a single cycle delay. Earlier, it has all operations are single cycle, but here, operation become some operation which is slower, it is become 2 cycles operation.

So, with this, my I can make my clock much faster, and I can execute the operation which is parallel to I mean in parallel to a operation which is happening per multi-second. So, what is determine the operation delay. So, in practical circuit, your delay of all operation may not be same, and delay is the number of clock cycle to execute the operation.

So, for each operation now, we have a delay associated. If the operation is single cycle like adder or subtractor, it is a one, delay is 1 and in the operation is basically multi-cycle, and it will be two or three. So, if it is two cycle, it is a 2, if it is a three cycle, it will be 3 and so on.

So, now, for each node, we have to determine based on your target architecture, you might decide that I want to do my multiplication operation in 2 cycles or 3 cycles. So, you have to specify not only this sequence graph, but for each node in the graph, you have to specify the delay of these nodes ok. So, that you have to specify.

So, just to conclude complete this discussion, there are three, two other possible way you can execute this kind of mismatch of delay. So, this is the multi-cycling, the other operation is the option is the operation chaining. So, what I am going to do? I am I will keep my clock 30 nanoseconds slower clock.

I am going to execute this multiplier for 1 cycle, but I am going to make these two things in chain. That means, there will be one adder followed by a subtraction will be connected in a chain and there is no register in between.

So, the way these are all discussed that there will be a set of register here and these data will be again there is a set of register here and there is a set of register here. So, it is basically after every cycle, the data will be stored in some sequential element.

But here, there is no sequential element at this between so, this two add; two FU are connected in a chain so, this is not there. So, this is called operation chaining that means, still my clock is slow, but I am doing more operations in parallel so, it will give you the same benefit like multi-cycle, but only the architecture will be different.

So, it is you have to decide whether you go for multi-cycle or operation chaining because operation chaining, clock will be slow here, clock is faster and here, architecture is simpler because you do not have to make a chain of; chain of operator, it is basically utilizes the same addition operation to do this both operation, but here I need two operator.

So, it is the trade-off between these two, but this is the possibility by operation chaining also, I can solve this problem of having two operations which have different kind of delay. The other option is the pipelining which is very useful that once you have this my operation is multi-cycle, I will still make the operation multi-cycle say 3 cycles, but I do not wait to complete this operation and then, I will start a new operation, what I can do?

I can start off the same multiplier for the different set of data. So, this is called pipelining. That means I am just adding some pipelining stages in between which is basically set of registers and then, what I can do is I can just take a first set of data say is a and b and I will just do the first part of the multiplier.

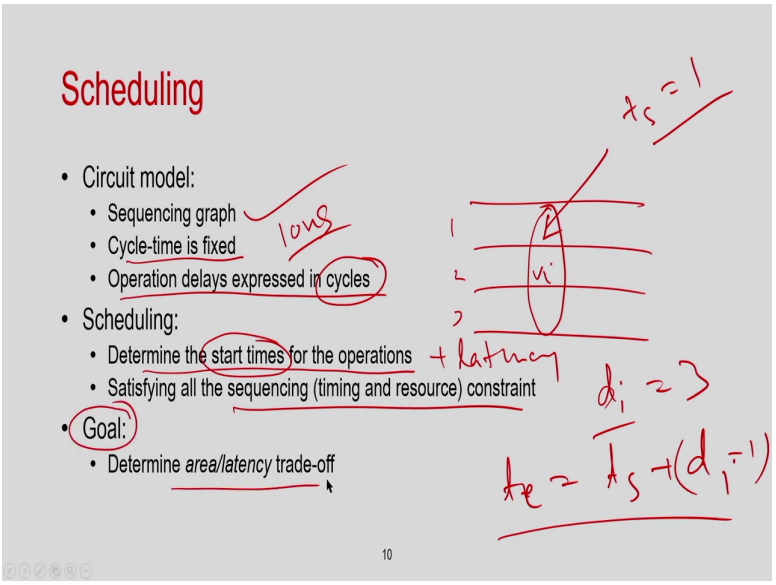
So, I have to now develop my multiplier such that there are three stages and it will do some processing or maybe and the intermediates result will be stored here and in the next clock, I am going to give a1 and b1, then what is going to happen? This part of the multiplier will execute on a, b and this part of the multiplier will execute on a1, b1 and the next clock, I am going to give a2 and b2.

And so, now, the last part of the multiplier will execute on a, b, middle part will execute on a1, b1 and the first part will execute on a2, b2. Here, although I make my multiplier in multicycle, but I just making this kind of pipelining of the multiple tasks on the multiplier. So, I can this multiplayer is equipped to execute on multiple data in parallel and it will give you much faster performance.

So, all these detail so, obviously, the implementation of this pipeline will be different from this and this pipe multi-cycle will be different from this and so on, but these are all internal digital design things which is kind of abstraction to us in this course. So, suppose we have this kind of multiplier available in my database and then, I am going to do the scheduling.

So, my objective is not to develop this circuit because it is something is given, it is a library of resource is given to me and high-level synthesis is just given that schedule, it when given this kind of resource, you just try to schedule the operations.

(Refer Slide Time: 25:32)



So, these are the variations. So, the bottom-line here is that you are you have the sequence graph and also, you have given the delay of the operations in terms of cycle so, how many clock is needed to execute this operation is given, but the clock time is fixed so, you mention that my clock is 10 nanoseconds and so on. So, this is what is the given to us and what schedule does? It determine the start time of the operation.

Now, this start time will had a make sense because if your operation which is say multi-cycle, you have a operation multi-cycle, you got to understand where it starts. So, if this is 1, 2 and 3, this operation starts at time step 1 so, t start of this operation will be 1, start state and it will end at time stamp 3.

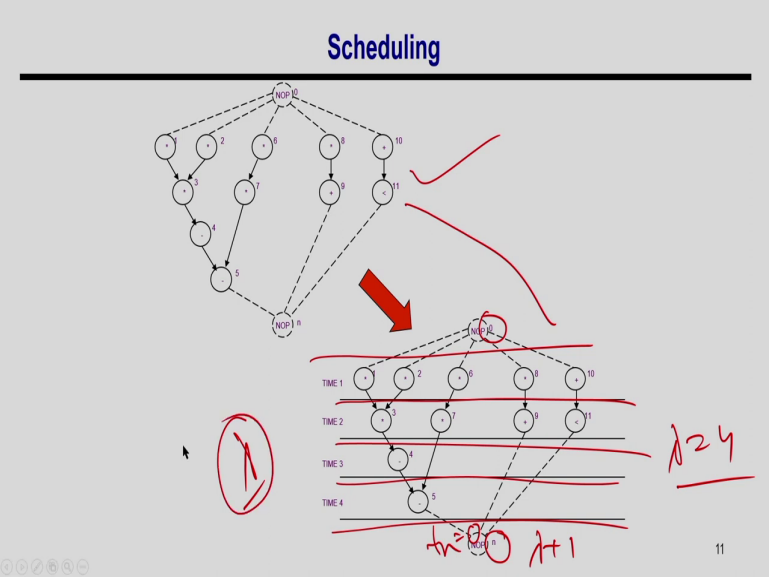
But if I know the start state and the delay of the operation, say the delay of the operation suppose this is the node Vi and the delay of the node is di and then, my te = ts + di because it start from 1 so, 1 + 3 it is basically is di - 1 basically because so, it start from 1, 2 and 3 and my di = 3 in this case.

So, schedule give you the start time where the operation starts and if the operation is multi-cycle operation, I can understand what are the time step, where this operation will be live or executing and it is not that it is give this, but you have to make sure that the

constraints are satisfied. So, the so, far the only one constraint we know which is the data dependency.

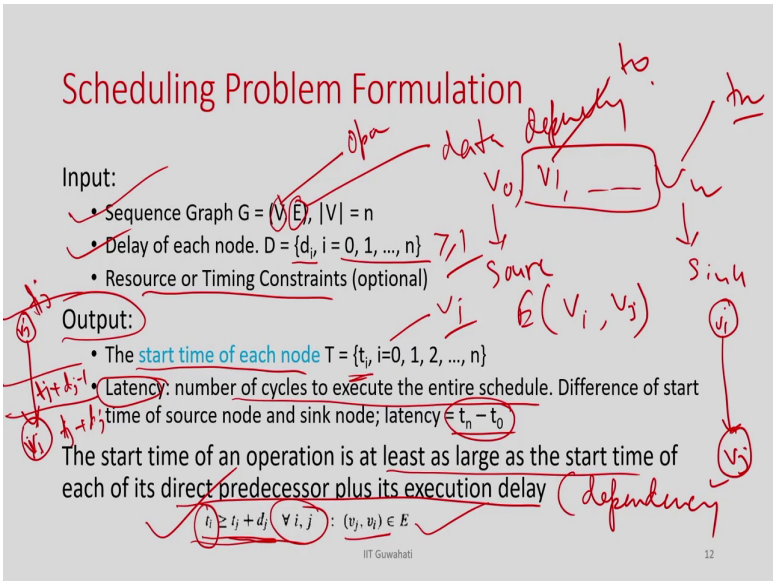
You identify the start time plus latency, but you make sure that so far the constraint is only the data dependency so, the data dependency constraint is satisfied, but your overall objective of the scheduling is just not the satisfying the scheduling, you might have different other goals which is latency or area trade-off that I am going to discuss.

(Refer Slide Time: 27:29)



So, if as I mentioned that given the sequence graph and the delay of this node, you are scheduling just assigning the time step, you just determine the operation that is going to execute in each clock and the total latency is equal to 4. So, you can see here this is always 0 and this is always the sink node which is lambda plus 1. So, this is what scheduling does.

(Refer Slide Time: 27:52)



So, if I just try to formally now place the scheduling problem that you have a given a sequence graph where the each node represent a operation and this edge represent the data dependency. So, this is my operation and this is the data dependency and you might you can assume that I have node say V0, V1 to Vn so, this is my sink node always and this is my always source node and these are the actual operations.

And if there is a edge which is from say Vi to Vj that means, you have a dependency from Vi node to Vj node. So, you have a dependency that means, you cannot execute these operations before that and in this particular discussion, operation chaining I am not discussing, but you can always extend the whole scheduling to operation chaining as well.

And what is the output? The output ok let me complete. So, this is the sequence graph and you have also given the delay so, that means, the delay of each node di for each node so, V0 to Vn and delay is basically is greater than equal to 1, you need at least 1 cycle to execute so, either it is 1 or more and also, you might have given some constraint on resource or timing that I am going to discuss next and your output of this schedule is the start time of each node.

So, that means, you identify the ti for each Vi so, you just for each node Vi, you just identify the ti where the operation it start its execution and the overall latency, it is the number of cycle to execute the entire schedule. To interestingly that can understand that start time is

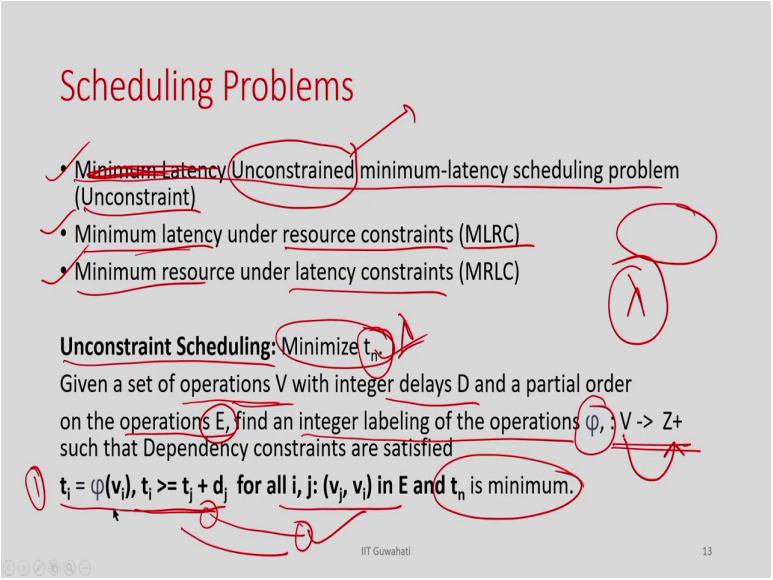
basically tn for Vn and this is t0. So, the latency will be this minus this. So, tn minus t0. So, this is how I can identify this.

And obviously, the schedule should satisfy the dependency constraint, this is what is called dependency constraint ok that means, if there is an edge from say Vj to Vi. So, the way it is represented if it is say Vj and there is a Vi that means, this is my Vi and there is an edge here and it start at tj and the delay is d so, that mean it will execute completion of the execution is basically tj + dj - 1. So, it will complete here.

So, my Vi can only start after this. So, this is what is given by this that my t i, the start time of this node will be at least tj plus dj. So, so that is what is the start time of this ti must be greater than equal to tj + dj for all such edges in my edge list. So, it is not that you execute this you give the time step, but the time step must satisfy the dependency constraint.

Here, because the operation it can be multi-cycle so, the constraint is basically ti is greater than equal to tj + dj where there is an edge from Vj to Vi. So, what is my scheduling problem.

(Refer Slide Time: 31:11)



So, based on your optimization goal, you might have different variation of the scheduling. There are three primary variations of the scheduling problem. The first one is the minimum

latency unconstrained scheduling. So, basically, it is unconstrained you just say unconstrained scheduling.

So, you can just remove this part, unconstrained minimum latency scheduling problem that means, you do not have any other constraint other than your data dependency. Data dependency is by default, you cannot ignore that, it is must there. So, only thing there is no other constraint.

So, that is why it is unconstrained, there is no resource constraint or latency constraint your you have just given the sequence graph and your objective is to execute everything in minimum number of time step. That is why this is unconstrained and minimum latency.

The other variations is now think about that so, doing this, you might need lot of resources, but that much of resource may not available. So, you might say that I have this much of resource, and you schedule the operations within that using that resource and you minimize the latency.

So, you try to schedule the whole behaviour using a given set of resource that means, you have given a set of resource constraint, but your objective is to schedule the everything within minimum number of time step. So, that variation is called minimum latency under resource constraint, this is another variation of the schedule which is called MLRC.

The alternative variation is the just do all of this that you have given a latency constraint that I have given a sequence graph and I want to execute this it is a 10 clock cycle and you identify the minimum number of resource needed to execute in 10 clock that is what is the other problem.

So, you can have three variation of problem, the first one is that there is no constraint, this is a unconstrained problem and your objective is to schedule everything in minimum number of time step so, that means, you want to minimize your latency under no constraint and the unconstraint is implied, there is a data dependency constraint is already there because you cannot deny that.

And for the other two variations where you have actually additional constraint that you want to minimize the latency under a given set of resources which is called MLRC and other

alternative variation or the dual problem is that you have given a latency bound there and you want to minimize the resource to achieve that latency that I have given say this much of this latency.

Say, I want to execute in say 10 cycles and I want to identify the minimum number of resource needed to execute that sequence graph in 10 cycles with minimum number of resource. So, these are the three variations of the problem, and we will see how to solve this problem, what are the algorithms in the problem in subsequent classes.

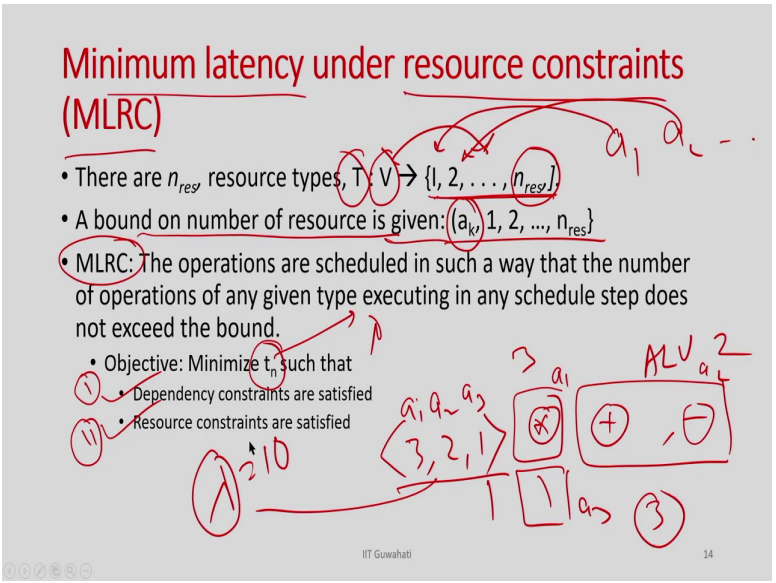
So, let us try to define this formally again, this unconstraint scheduling your objective is to minimize the time step, the latency you can assume this tn. So, the problem here is that the identifying the minimum value of this lambda or the latency is same as minimize the t n because so, tn is the time start time of this node.

If this is minimum that means, all things are done very early, my latent lambda is also minimum. So, that means, I can always say that minimize latency means you minimize the t n or the minimize the start time of the sink node ok. So, here, the way you should define the given this V and the set of operation V and their corresponding delay and their data dependency E.

So, this is the input, you just identify your labelling of the operation phi which assigns so, V to some integer number which is basically the start time. So, this is the assignment of the nodes Vi to some start time, some integer. So, that was the start time of the node such that your dependency constant is satisfied.

So, what I am doing here is that my ti that is already discussed that if there is an edge from Vj to Vi, then this will only start once this is completed and that is given by this constraint, and this must satisfy for all E and your tn is minimum. So, that is will minimize the latency. So, this is what is the definition of the unconstraint scheduling which minimize the latency.

(Refer Slide Time: 35:25)



The other two variations I will just quickly now define. The first one is the MLRC. So, you have given a resource constraint. So, how can I give of the resource constraint? First of all, you have to understand what are the resources available. So, the what kind of resource is available? The function you need. So, you have multiply, you have add, you have subtract, you have div and so on.

So, I what I can do? So, usually in hardware adder and subtractor is same so, I can just assume that these are ALU which can do this addition, subtraction, this kind of operations and I have a mult, I might have a div and so on. So, these are the type of resource and say there are n, these are the types are n res. So, these are the number of type of resource available. So, for this example, there are three, one is multiplier type of resource, one is ALU type of resource and one is a div type of resource.

So, for each node, I know what is the resource type so, that is already given. So, for that the node is the number of operations and the for each operation, what which is the type of operation? Whether is a multiplier type or whether there is an adder ALU type or a div type is known are known to us, it is obvious from the program and important is the resource bound.

So, for each type of such resource, a some bound ak is given. So, a1 is the resource bound for the 1st resource type, a2 is the resource bound for the 2nd resource type and so on. So,

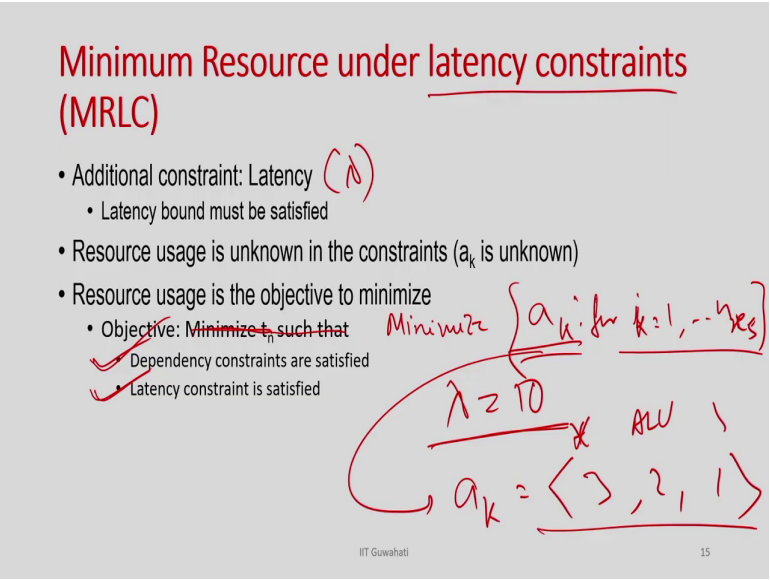
suppose you have given 3 mult, 2 ALU, 1 div so, that was the resource bound that means, I have so, suppose if this is a1, this is a2 and this is a3, my bound is basically <3, 2 and 1>.

So, this is the resource bound a1, this is a2, this is a3 that means, you just specify how many multipliers is available in your target hardware, how many ALU available in your target hardware, how many divider is available in you hardware so, that is the resource bound and I have given the behaviour now. So, your objectives are to identify the minimum latency under given this resource constraint.

So, now, these MLRC can be defined like this given that schedule the sequence graph and the delay that is already given that you have the sequence graph and the delay of the operations, you try to and also given this resource bound, there are three inputs now instead of two, you try to schedule the operation such that you minimize this tn, you minimize the latency and you satisfy both dependency constraint and resource constraint.

So, earlier, it is the same thing, it just that is unconstraint is basically just satisfy the dependency constraint and now, it is satisfying both dependency and resource constraint because that is the things given to us. So, this is what the MLRC problem definition.

(Refer Slide Time: 38:15)



The third variation is MRLC which is already I have defined that your latency given, you have given a lambda, you want to schedule everything and say 10 clock cycle is given and

you have given the sequence graph and delay everything so, you have given this latency, sequence graph, and the delay of each node, what you want to do?

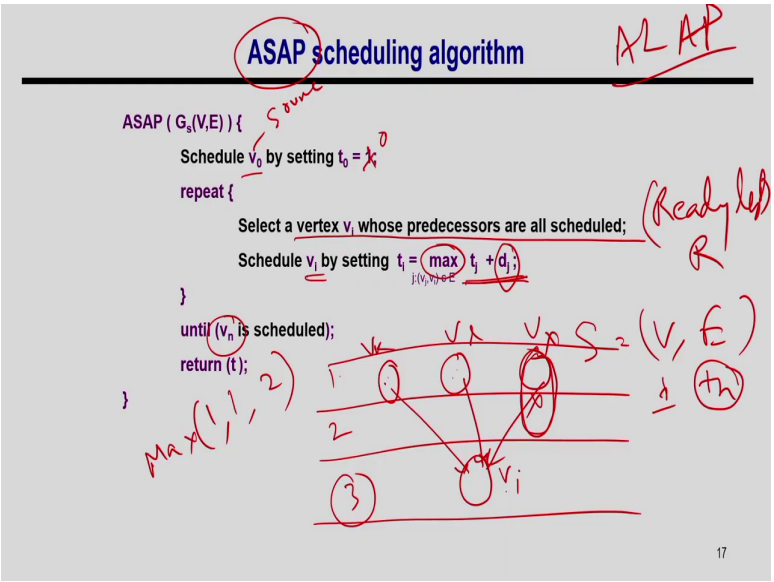
You want to do a schedule which satisfies the dependency constraint and the latency constraint and what is your objective? You objective is basically minimize ak for i; k = 1,2, ..., n resource. So, I already know that this number of resource is 1 to n resource. For each k, I want to find out the minimum value of ek.

So, if you take the same example earlier, say I have given a sequence graph, earlier example what I what is given? I have given these <3, 2, 1> and I identify the latency is lambda which is say 10. Now, I have given this lambda equal to 10, you need to identify the ak for each time and which is basically say 3, 2, 1. So, you need 3 multiplier, 2 ALU and 1 divider.

So, suppose this is what you want to identify, this is the dual problem that you given this latency bound 10, you identify to schedule these operations within 10 cycles which satisfy both dependency and latency constraint, what are the minimum number of resource needed? So, that was the problem.

So, these are the three variations of the problem and this I mean we have different algorithm for all these variations and we are going to discuss them in detail in the subsequent classes.

(Refer Slide Time: 40:15)



In today's class, I am just going to discuss this unconstraint, the first version which you can understand this is much simpler one because there is no other constraint so, for that I am going to take two simplest algorithm, one is the ASAP algorithm and the other one was the ALAP algorithm. These are very simple algorithm. So, I am quickly cover this.

So, the ASAP algorithm is basically as soon as possible. So, you have given the schedule, graph sequence graph which is basically V and E, you have you try to schedule this behaviour and there is no other constraint you try to minimize the latency. So, that means, the tn, you try to enter the minimum value of the tn. So, how I am going to do it? I just schedule this V0 into t0 not 1, this is wrong. So, I just V0 is my source node.

Now, for the rest of the node, what I can do? Since it is a as soon as possible so, whenever a operation is ready to be schedule, let us schedule it. What do you mean by the ready to be scheduled? That means, it is all predecessors are already scheduled.

If its predecessor operation is not scheduled, I cannot schedule this operation now because it will has to wait for that operation to be scheduled and operation is ready to be scheduled that means, the operations which are basically all predecessors are already scheduled. So, you identify all such vertex whose predecessors are scheduled, we can assume this is a ready list, I just maintain this ready list R.

And then, just how can you schedule this? So, you know because schedule Vi because these are the operation that can be scheduled now. So, you are now in time step, you are now tried to schedule this nodes and where I should schedule Vi? Because if this is the node and there are, these are the predecessors so, these are the predecessor this is my node Vi and there is a there it is depend on this say these three Vj. So, this is say Vk, this is Vl and say Vx.

When I can schedule? Only so, all these three must completed its execution. So, suppose this is executing so, this may be different cycle also. So, this might be a single cycle, this might be single cycle, this might be 2 cycle operation.

So, I can only schedule once all these predecessors are done, once when these predecessors are done? The t, so, I can so, if this is my predecessors is say Vk; Vj, I know and if it is

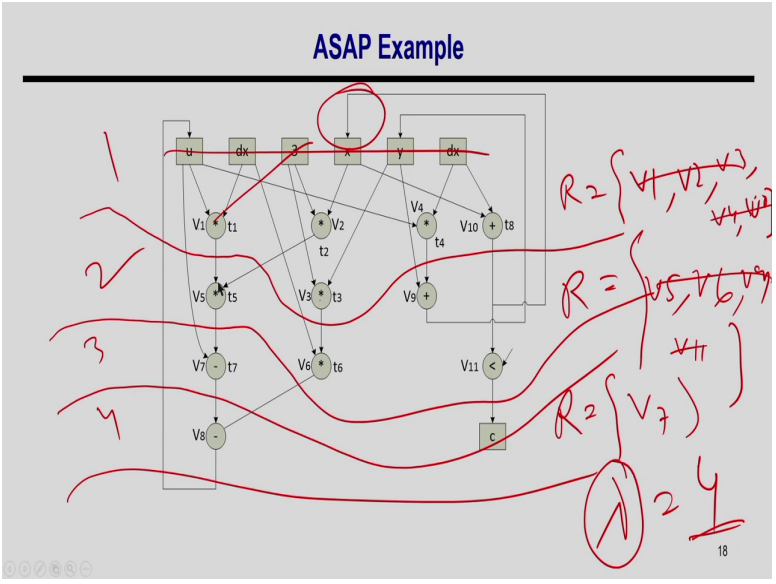
taking dj, then I only can schedule the operation at tj plus dj. So, what I am going to do now? Because it might have multiple such predecessors, I am going to the max of that value.

For example, here if say this is 1 and 2, then since this is it is completing the execution in 1 clock; cycle 1, this will complete its execution in clock cycle 1, but this operation is taking 2 cycle so, I can only start this Vi at time step 3 which is max of 1, 1 and 2. So, this is the tj+dj, I assume this is a single cycle, this is a single cycle and this is a 2 cycle operation.

So, this is how I can just do this. So, I will just identify this and these are all scheduled and once my sink node is scheduled that means, I am done, this is how I can execute this operations. Whenever the operation is ready to be schedule, you schedule it, but where to schedule?

Based on the predecessor are scheduled and whenever they complete. I only know the start time of this and their delay, from that I can identify what is the latest possible time step till what the predecessors are actually executing the behaviour and I am going to schedule it after that.

(Refer Slide Time: 43:57)



So, let me just take the example. So; obviously, this is something there is a sink node, this should you can remove these things and then so, in the initially, you can see that these are all depends on the inputs so, V1, V2 so, my ready list consist of V1, V2, V3 also because it is

also depend on this so, V3, V4 and V10. So, what I can do? I can just schedule because these are all inputs, I can schedule V1 ah, V3 these all five operation into time step 1, then these are all done.

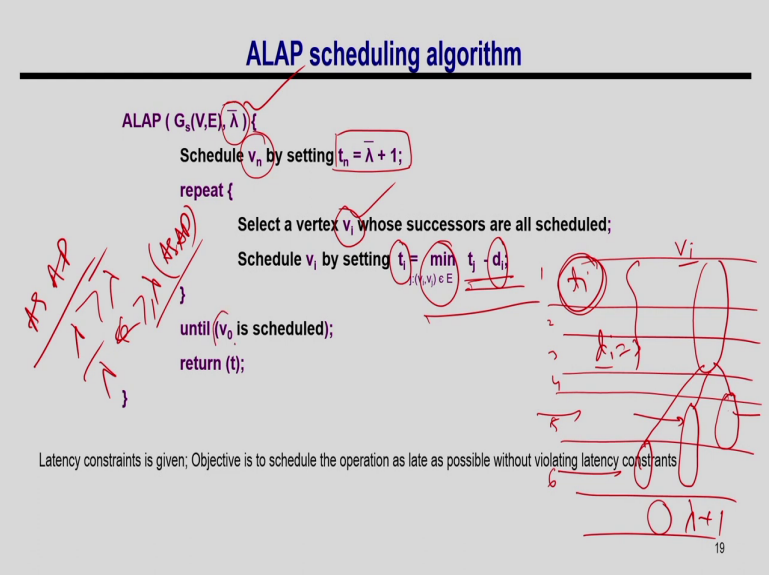
Then, I am again going to calculate my ready list and let us assume all these operation in a single cycle. So, but if it is not single cycle, you have to identify the start time of the subsequent node. What is the now ready list? My ready list is now V5, then V6, then V9 and V11. So, in the time step 2, I can schedule these operations.

So, this is my time step 2. Then, these are all done and I am assuming everything in single cycle otherwise again schedule will be change. So, then, what is the ready list? I have only V7. So, in the time step 3, I am going to schedule V7 and in time step 4, I am going to schedule V8. So, I need 4 cycles.

So, this ASAP tells us the minimum number of time stamp because here you can understand that I am executing everything as early as possible. So, whatever this lambda I got from this equal to 4, I cannot execute this behaviour less than 4 time step because there is no constraint here, I cannot have any schedule which can execute this behaviour less than 4 cycles.

So, ASAP give you this number that the minimum latency needed to execute this behaviour; given behaviour because this is where there is no constraint, and I am executing everything as early as possible so, that always ensure that it gives the minimum latency and if it is a the some more operations are multi-cycle, you I mean that is kind of your homework, you can actually try that what will be the ASAP scheduled for that ok.

(Refer Slide Time: 46:13)



Now, I am going to explain ALAP algorithm. So, as the name suggests, it is as late as possible. So, my objective here is to schedule the operation as late as possible. So, if you remember that in ASAP, what I did? I just start from the search start node or source node because I want to schedule the operations as soon as possible.

So, here, intuitively you can understand if I start from the start node, I do not know what is the latest possible state where I can schedule so, the best idea to start from the sink node or the last node ok. So, how we will start? So, I will start from the sink node so, the V n and I schedule to the last that lambda.

So, I have given this pi, the number of schedule possible, the number of state in which I am going to schedule the operations, so, I have given that, I am just put into the last node. So, I just say put the sink node here so, lambda plus 1. Here one thing is to be noted here is that how do guarantee that this the given lambda bar or the number of time step we need to schedule these operations is sufficient to schedule this. So, this we can check with ASAP. So, if the ASAP time is so, ASAP give you say lambda and this lambda is greater than lambda bar so, the I mean I cannot schedule this operation in lambda bar, then I can have this check at the start and I will say [FL] no, it is not possible to schedule it here, but if my lambda bar,

the given time here is less than sorry greater than equal to lambda which is my ASAP time, then it is schedulable so, that check I have to do first.

And then, what is coming here? So, I will start with the sink node, I schedule it in lambda plus 1, then how I will going to take a node say Vi, how do I know? So, first we have to check when this node can be scheduled is that when it is all successors are scheduled. So, I have to just check all the node whose all successors are scheduled say let us say I have decided a node such Vi. So, say suppose this is my Vi. So, this Vi want to schedule now.

So, when I can schedule it? I can only schedule it when all its successor such a way that in which time step so, rather I the way I should say [FL] I want to schedule it in t i alright so, what will be my ti? Which state I can schedule it? So, the answer is that I should do such a way that if I schedule Vi at start its execution at ti, it should finish say suppose its executions say it is a 3 state cycles.

So, this is my di. So, the my ti will be such that if I schedule this Vi at start its execution from ti, it should able to finish his execution before the start time of the his successors. So, unless its execute execution is complete, the start node I mean the successor node can start, but the successor node is already there scheduled is fixed.

Say suppose for this node, there are 3 successors. So, these successors is 1 cycle ok and say there is another cycle is 2 cycles and there is another is 1 cycle, then there are 3 successors of this node, it can happen. So, this node is nothing but is scheduled like this. So, it has 3 successor node. So, I have to determine the value of t i and how can I determine the value of t i that is what I try to understand.

So, the objective here is that I have to make sure that if I schedule Vi at starting his execution at ti, it should finish before the start of his successor nodes and this tj is already known. So, as I mentioned this is the tj of this node, start time is this state, for this; this is steady state, this is also steady state. So, what is I have to decide?

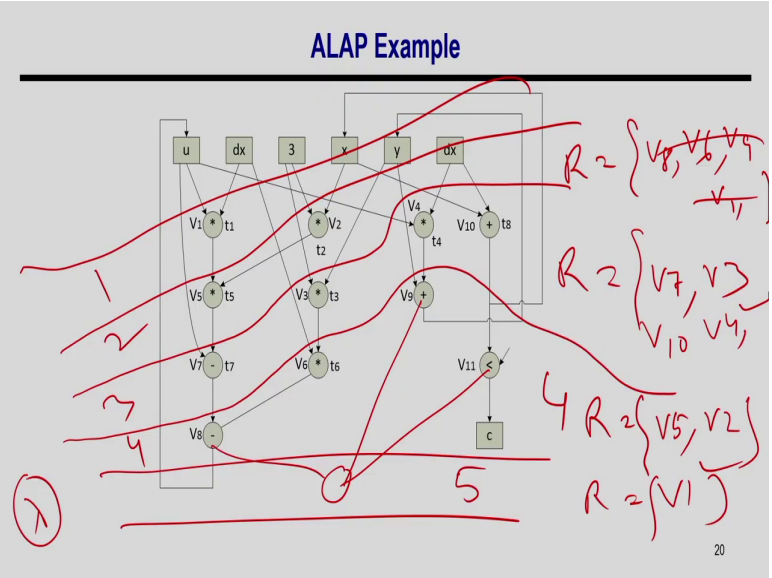
So, these are the tj for all successors so, I am going to take the start time such a way that this tj minus di so, di is the execution time of this. So, if I take this node say di is say 3 cycles so, if I so, basically let us try to understand this say this is 1, 2, 3, 4, 5 and this is 6. So, if I say

this is 6 so, 6 minus 3 is 3. So, then it could start from here, but if I start from here, it will overlap with this because this is starting from. So, this is starting for 4 one. So, this is starting for 4 so, 4 minus 3 is 1.

So, what I should take? I take the minimum of all the value of tj minus di. So, basically i it is not dj. So, basically, I have to see what is the start time of the all successor node and from that, I have to see who has the minimum value and from that, I will just subtract d i and that will be my ti.

So, that is how I should decide the start time of the node Vi and this process will go for every iteration, I am going to select a Vi whose all successor is done and then, I am just decide his time by this logic and I will keep doing it until my source node is getting sink I mean scheduled.

(Refer Slide Time: 51:47)



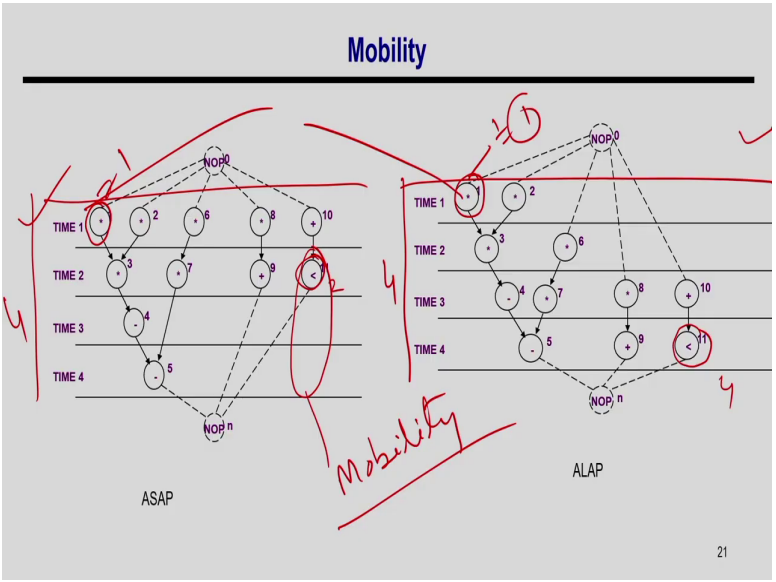
So, if I just try to do it for this example and you assume that there is a sink node here which is I have missed here. So, if this is the case, then you schedule this in time step 5 and let us assume that these are all single cycle operations so, my ready list will be V8, V6, V9 and V11. So, what I am going to do? I am going to execute these operations in time step 4 and these are done.

The next time, what is my ready list? V7, V3 and V4 and V10. So, I can schedule this V7, V3, V4 and V8 in time stamp 3.So, I am going in reverse direction now and then this is done, then my ready list will be V5, V2 only these two. So, I can schedule these two operations in time step 2, then this is done, then my ready list will only contain V1 and I am going to schedule these operations in time step 1.

So, this is exactly like ASAP algorithm, but I am going from the sink node to source node because my lambda is given and I want to schedule the operation within this time frame, but I want to execute the operation as late as possible, I want to delay them as late as possible ok.

So, this is how these two algorithms work ASAP and ALAP which is something quite obvious. Again, you should actually try this considering the multi-cycle. So, if we assume that multiplier is 2 cycles, you identify the schedule that can be a good homework for you.

(Refer Slide Time: 53:31)



So, if you take that example the Diff eq example, this is my ASAP solution and this is my ALAP solution. You can understand this is where every time, whenever there is a operation is available, I am going to schedule it and whenever in this case, whenever I want to schedule it as late as possible, but I want to make sure that everything is done within 4 time step.

Now, the question is what is this algorithm does? What is the use of this algorithm? The first of all the first use of the ASAP it gives you the minimum lambda. So, what is the latency

bound? You can actually calculate the latency bound by ASAP because you cannot have any solution which can give you the solution, which is whatever the ASAP solution gives; you cannot have any solution which is lesser than that time ok. So, that is something is very well understood.

And this ALAP is something as late as possible, but this gives a very nice property. So, let us consider this diagram, this operation say operation 11. So, it is schedule in time step 2. This is the earliest possible schedule, we cannot schedule it before that and what is the time step, where it is schedule in the ALAP? It is schedule in time step 4. So, this is my time step 1. So, what is this schedule? This is the ALAP. So, this is 2, this is 4.

So, then, this is the time which is the last possible time when it can schedule, it cannot schedule after that because then this would not satisfy the lambda and for this both case, my lambda is 4. So, given a latency lambda, if I identify the earliest possible time, where it can schedule? By ASAP and by ALAP, I can identify what is the latest possible time where it can schedule so, that something is the boundary and what is the flexibility? So, it can be scheduled anywhere in between and that is what is called mobility.

What does it mean? So, by ALAP, I determine what is the last possible time it can schedule and the by ASAP, I identify what is the earliest possible time where I can schedule and it means in between that ASAP and ALAP, it can schedule anywhere.

So, that gives the mobility, and this is very important in the context of the constraint scheduling like this was constraint on timing constraint so, when we have to schedule the operation on say on a given resource bound, you have to schedule the operation which is become critical or which become mobility become 0, it has to schedule.

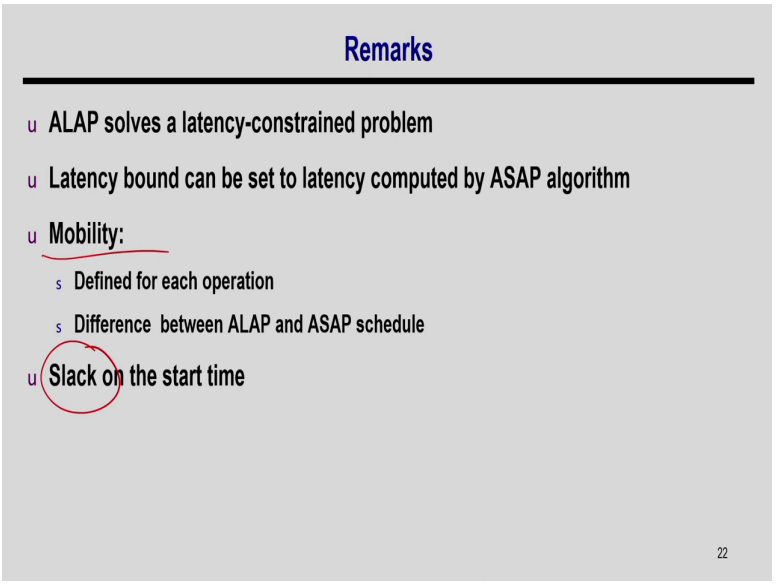
For example, you consider this operation V1 whose earliest time stamp is also 1, ASAP; ALAP also 1 that means, this is a critical operation, and it must be schedule in time step. If you do not schedule it here, your total latency will not meet your target latency. So, this mobility determines the criticality of operation.

So, suppose you are in some time step, what are the operation that are available to be scheduled there, you identify the criticality operation which is the mobility which is nothing,

but ALAP minus ASAP time step and if the mobility is 0 that means, it is now become critical for this time step, you must schedule that first and then, if the operation which has more of mobility, you can defer it.

So, whenever you are going to discuss about the resource constraint scheduling, this mobility is a very important factor which actually determine the if there are say less resource available and more operation to be scheduled, which operation to be selected for the time step.

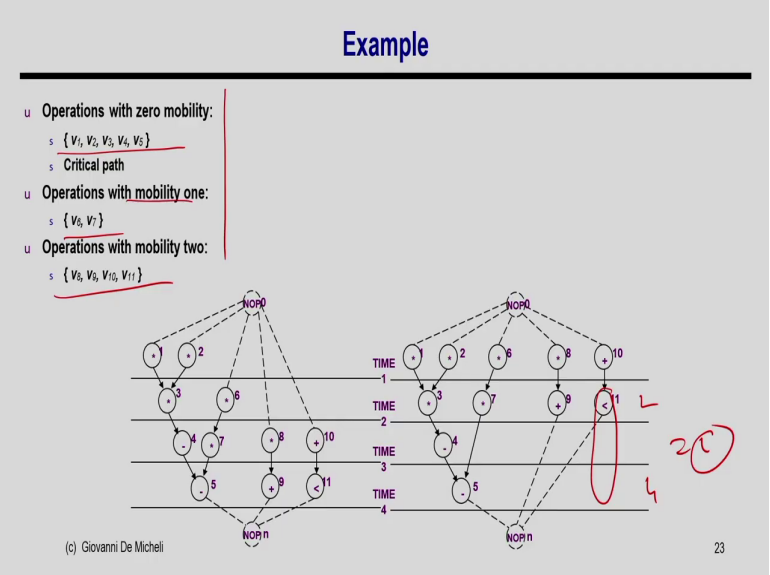
(Refer Slide Time: 57:18)



So, this is why this ASAP and ALAP algorithm is so important although, they never get used practically to implement certain design, but they are always run-in background to help the other efficient algorithm to run. So, specifically, this ASAP gives you the minimum latency bound and this ALAP and ASAP combinely give you the mobility of the operations.

And from this mobility which is the slack, this is the time where it can be scheduled so, I can internally identify the slack of each operation and based on my resource constraint or the latency constraint, I am going to schedule the operation which slack is less.

(Refer Slide Time: 58:00)



So, for this example, you can actually identify the mobility of the operations which is kind of a homework for you and I have already identified the mobility of this operation is 2, basically this is 2 minus 4 is 2. So, operation with mobility is two is this node so, you can actually verify and operation with mobility one is V6 and V7 and mobility zero is this operation.

So, with this, I conclude today's class. I am going to continue this scheduling discussion in the subsequent classes.

Thank you.