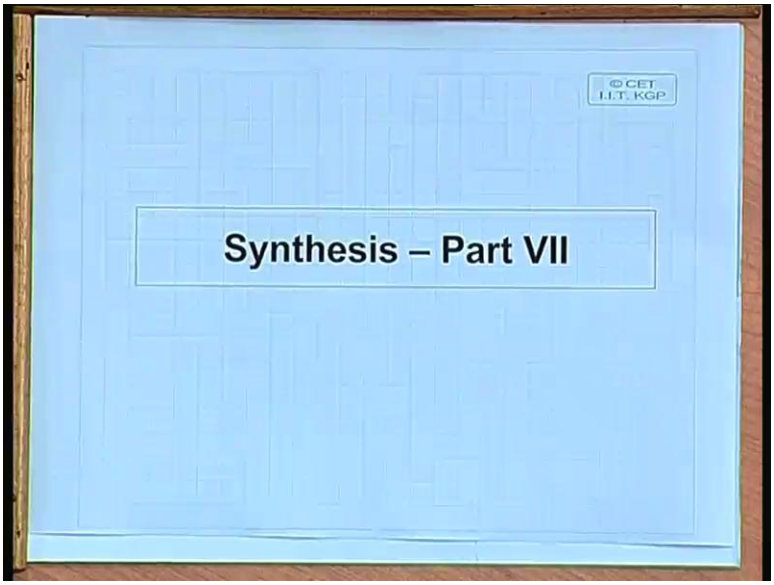
Electronic Design Automation
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No #14
Synthesis: Part VII

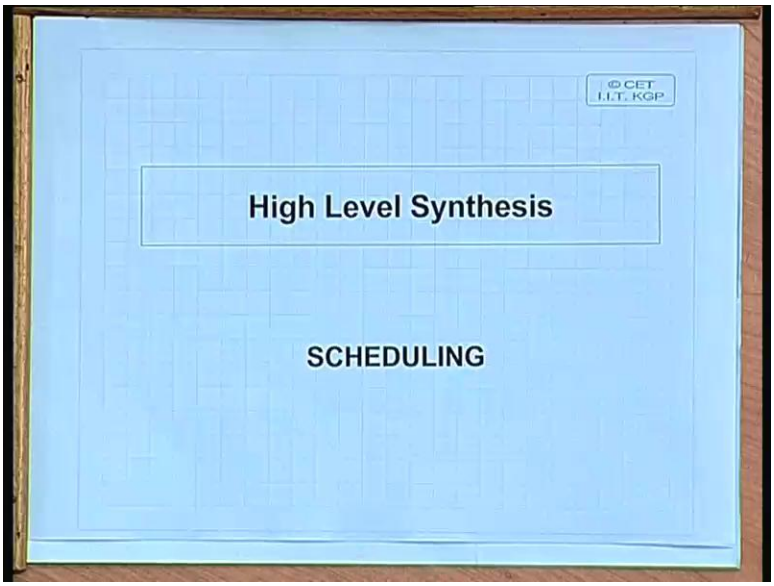
And so continuing our discussion on high level synthesis.

(Refer Slide Time: 01:09)



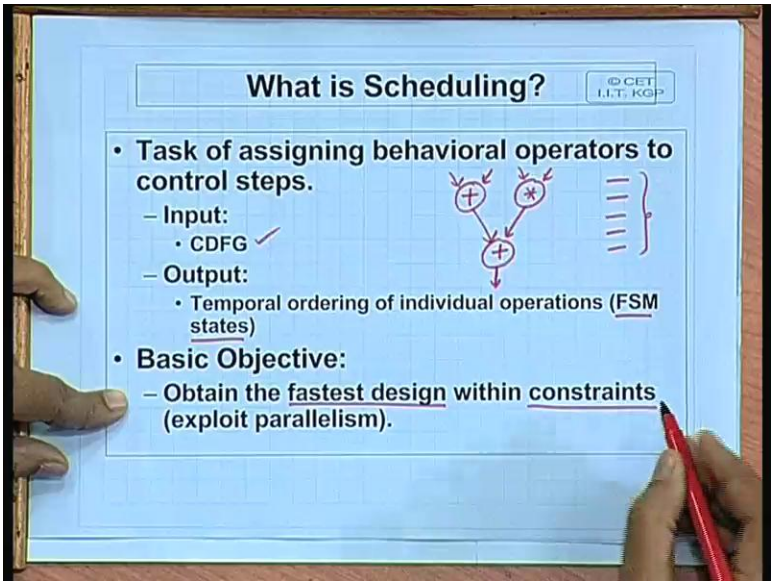
If you recall we had talked about the c d f g, the way we represent behavior specification in the form of a graph. We talked about partitioning we need to partition a large c d f g into a smaller pieces. So that each piece becomes manageable. So continuing with our discussion we talked today about some other operations we carry out on the c d f g in order to achieve or fulfill our purpose of high level synthesis. They are called scheduling allocation and binding.

(Refer Slide Time: 01:44)



So we talk about scheduling to start with okay so here our assumption is that we have a, c d f g available to us to start with and from there we are proceeding in the beginning in the first step to process of scheduling.

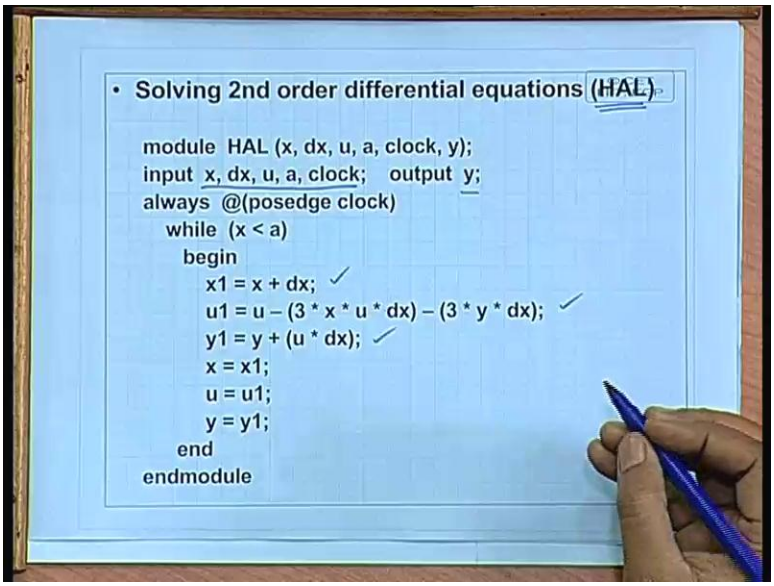
(Refer Slide Time: 02:04)



So first let us try to understand what we mean by scheduling. Scheduling is essentially the task of assigning behavioral operators to control step so what do you mean by behavioral operators. Well you look or you think of a typical portion of the c d f g for example it may look like this an adder a multiplication and another addition. So some data inputs are coming like this so the outputs of this addition will come here the output of the multiplier will come to this adder like this. Now when I say control step this control step may refer to the clock cycles with respect to the control unit the control unit will be generating the different signals to control the hardware in synchronism with a clock pulse. Now when I say you are assigning behavioral operators behavioral operators are these operations addition multiplication.

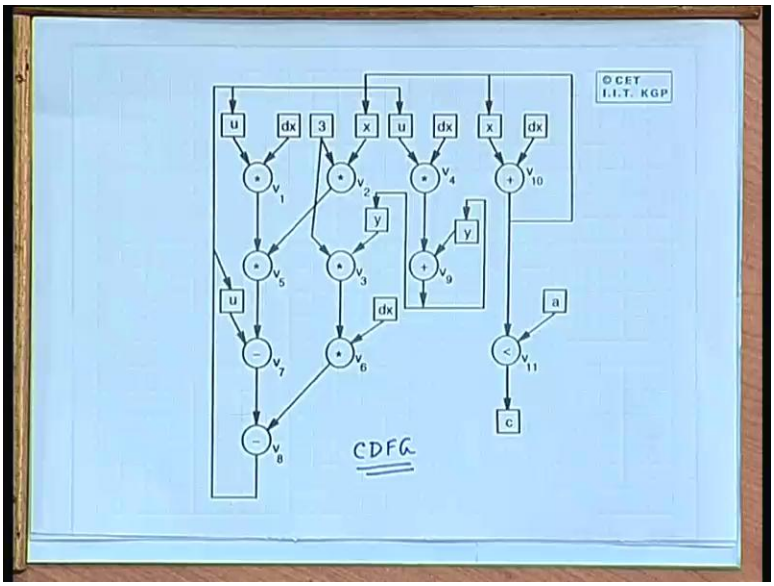
So we will be assigning or allocating these operations with respect to some clock cycle these are the so called control step. So we will see that there are several algorithm for scheduling where by we can put this operators in different clock cycles well in order to fulfill certain constraints. So for scheduling input will be a, c d f g of course and output will be some kind of ordering of the individual operators in terms of time it is a temporal ordering. Now this ordering of time will immediately give us information about the states of the finite state machine which represents the controller okay. So this is our objective to obtain a specification from where the fsm can be designed or synthesized directly. So obviously our objective is to obtain the fastest design which will take the minimum number of clock cycles. Of course there can be some constraints like the number of available resources or means if you want maximum amount parallelism that's a different story okay. So our input as I had said will be a c d f g I am giving a typical example to show how our input to this type of scheduling may look like.

(Refer Slide Time: 04:24)



Well this is a classical example which many people take for the purpose of illustrating their scheme this is a Verilog code for solving the second order differential equation and the name of this code is called HAL this came from the source actually. Now you see this module description has inputs as x dx u a and clock and y is the output now its an iterative loop in synchronism with the clock. Well while x is less than a, you do these 3 steps of computation and at the end of which whatever values of x one u one and y one you calculate you assign them to x u y as the values for the next iteration. So these steps will continue you see that there are 1 2 3 4 5 and 6 multiplication steps 1 and 2 addition steps and 2 subtraction steps. So from a description like this you can straight away come to your c d f g description which for this particular example looks like this.

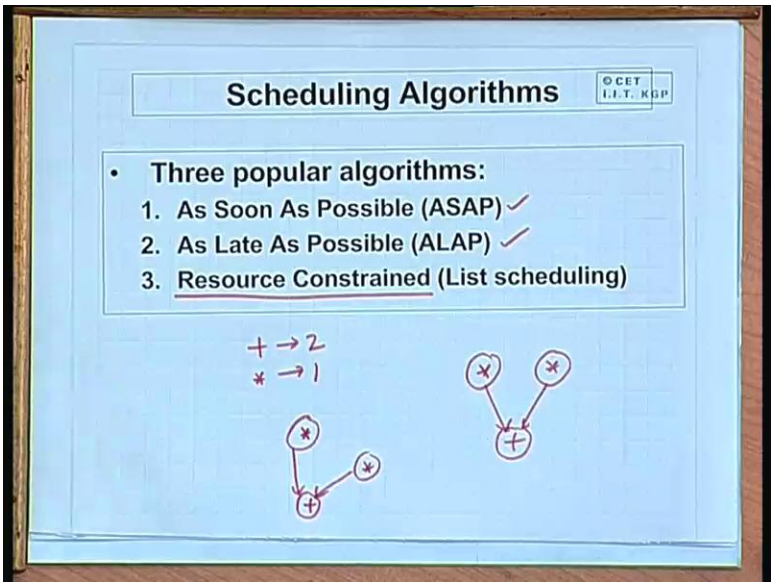
(Refer Slide Time: 05:46)



So in this example these rectangular boxes are used mainly for convenience they actually give the names of the variables which are coming okay. But the circles are the actual behavioral operators multiplication addition etcetera this is a comparator now given a diagram like this you can easily understand that in terms of time you have some flexibility. For example it is not necessary to do this addition in the first step you can even refer it down to the second step if the need arises. Because in this example already there is one path this can represent the critical path which takes 4 cycles to complete or 4 operations to complete.

This cycle requires any 2 operations so if my whole means iteration one iteration takes 4 steps to execute then if there is one thread of the iteration which takes 2. Then I have some flexibility in pushing or pulling up the operations means across the time axis. So this is essentially the flexibility that we try to exploit in the step of scheduling in order to well in order to have some kind of desirable characteristic characteristics in the schedule okay. So now talking about the scheduling algorithms there are 3 classical algorithms which we will talk about.

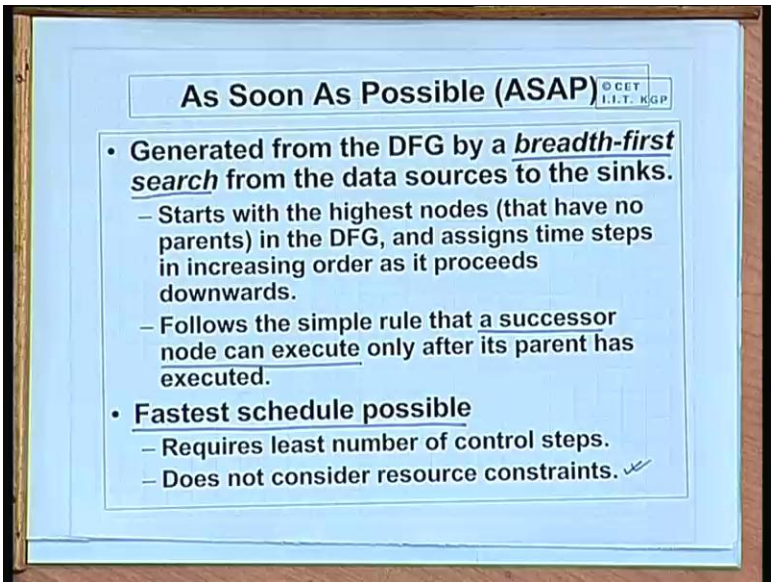
(Refer Slide Time: 07:12)



Of course there are others also we will see. The 3 classical algorithms are first 2 are absolutely greedy as soon as possible as late as possible we will explain what these are. The third one is well this is a schedule which works under certain resource constraints like you can say that I have only 2 adders I have only one multiplier. So now you obtain the schedule okay. So if you have more number of resources possibly you can allocate some or means 2 or more means operations in means in the one I means in the same clock cycle parallel.

Suppose you have 2 multiplications to be done followed by an addition of the result. So if you have 2 multiplier available then you can possibly do them in the same step. Or else what you can do you can do this multiplication in one step the other multiplication in the second step followed by the addition in the third step okay fine. So the resource constraint scheduling is also called list scheduling because it uses some kind of a list. So let us start by talking about the as soon as possible or the ASAP scheduling which is called more popularly.

(Refer Slide Time: 08:37)

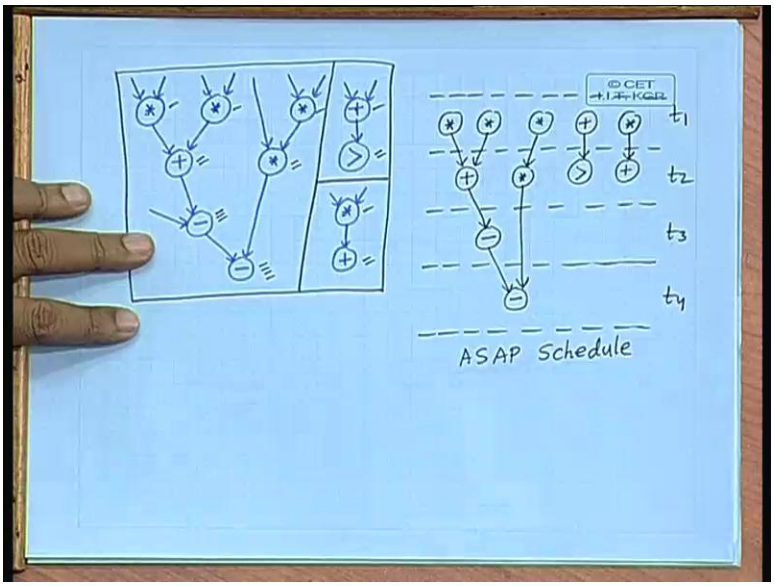


So ASAP is very simple in concept so here given the dfg you carry out a breadth first search from the leaf nodes of the tree and you carry out the breadth first search from the leaf node which represents the sources of the data down to the sink. And you assign time step in that particular order so you try to assign a particular operation to the earliest time step possible you start from the beginning go down the tree and ofcourse the obvious constraints have to be satisfied which is a successor node can execute only if the parents have completed their execution okay. So the parents have to be allocated a time step which is before that of a child. The ASAP schedule as is defined here this works without any constraints. So it is assumed that unlimited or infinite amount of resources are available so that whichever operators can be allocated you allocate it.

And as a result what you obtain is the fastest schedule possible because starting from the source you are trying to allocate an operation to a time step which is the earliest possible that is the earliest you can assign. So although these are greedy algorithm the way you are doing it the final schedule you get will be the smallest in terms of the total time required. Okay, but [Student Noise Time: 10:22] I will take an example. But as I told you this classical version of ASAP does not contain any reso does not consider any resources constraints we assume that infinite amount

of hardware are available to you okay. So let us take an example to illustrate what this ASAP scheduling means looks like.

(Refer Slide Time: 10:48)



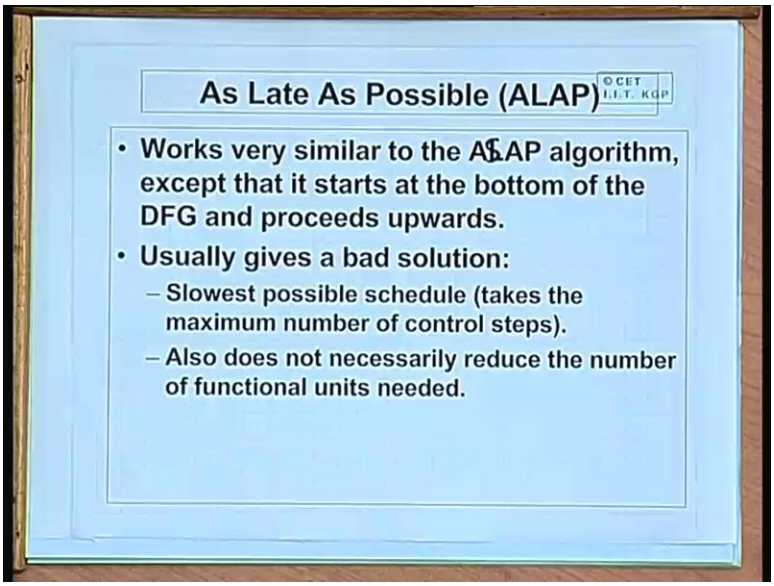
So we are not taking the hal example. Let us take a simpler example which will be easier for us to illustrate. So let us take an example like this. There are 2 multiplication steps followed by an addition there is another multiplication step here the data coming from here. There is a subtraction out here. These are the sources of the data and a final subtraction here this is one thread of the dfg and there is another thread of dfg assume that there is an addition followed by a comparison this is all. And there is another thread but there is a multiplication where the inputs are coming again from the top followed by an addition. So this is our given data flow graph this as you can see this data flow graph is actually a union of 3 graphs one is this other is this. So we apply this as soon as possible concept to each of these 3 graphs. But before doing this what we do we divide the time into several steps. These are the different time steps we are considering so in fact this example will take 4 time steps only.

So we show 4 steps t 1, t 2, t 3 and t 4 you look at this dfg and try to find out what are the what are the operations you can you can allocate to time step t 1. You can say I can allocate one 2 3

this addition 4 and this multiplication five. So all this 5 are assigned to t 1 because they do not have any other dependencies from the top so assign them. Well I am not showing the arrows from the top so these are the 5 steps or the 5 operations. We have already allocated then once these are completed in time t 2 what we can do. We can allocate this addition this multiplication this comparison and this addition. So let us do that so this addition is done whose inputs depend on these we can do this multiplication which other than the input it depends on the output of this operation. This comparison depends on addition and this addition depends on this multiplication. So after this is done in time step t 3 we can we have only one choice the subtraction. So you do the subtraction whose input is coming from this and in time t 4 we have the subtraction again. So the other input is coming from this multiplication. So here what we have obtained this is called As Soon As Possible or an ASAP schedule.

So what I have said ASAP is a greedy approach it always tries to assign a time step to the earliest time. A step of computation to the earliest time possible and the resultant schedule you obtain is the smallest in terms of the total time steps have taken. But you see it does not contain or consider any say means information regarding the number of resources actually available. Say this schedule for example requires 4 multipliers in the step t 1 because there are 4 multiplication you are trying to do in parallel okay fine. So this is one extreme as soon as possible the other extreme is that as late as possible as late as possible means you try to push the operations as late in the schedule as possible. So let us also look at this this ALAP.

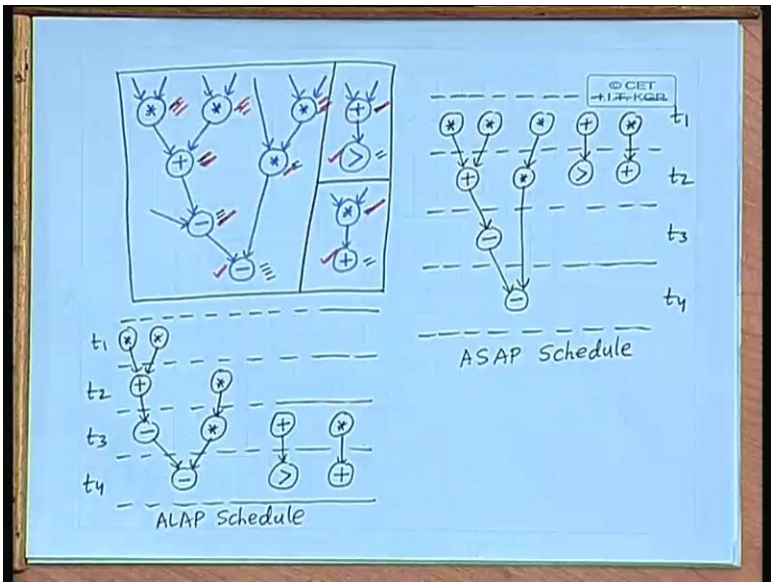
(Refer Slide Time: 15:52)



It should be s. So this As Late As Possible this works very similar to ASAP but the difference is that instead of starting from the top and going towards the bottom you start in the reverse order. You start with the root of the tree which in our case is the bottom and you start moving towards the leaves which are the data inputs and you try to allocate an operation to a time step in that order starting from the latest back down to the earliest. Now ALAP usually gives the slowest possible schedule takes the maximum number of control. Of course in this example you cannot just you really cannot appreciate that.

But in some other examples you can see that there will be a difference in the times required in ASAP and ALAP and this ALAP while it may increase the time step but it also does not reduce the number of functional units. So this ALAP is not really a good way of making schedule but again let me repeat that these kind of simple schedules are typically used as the starting point of some of the iterative algorithms which people also used. So if this is some kind of an iterative algorithm you can start with this ASAP and ALAP as the input solutions this represent the 2 extremes and then you can play around with it you can make some changes to that to obtain a better solution fine. So let us see how for that particular example the ALAP schedule will look like. So let us take that same example.

(Refer Slide Time: 17:36)



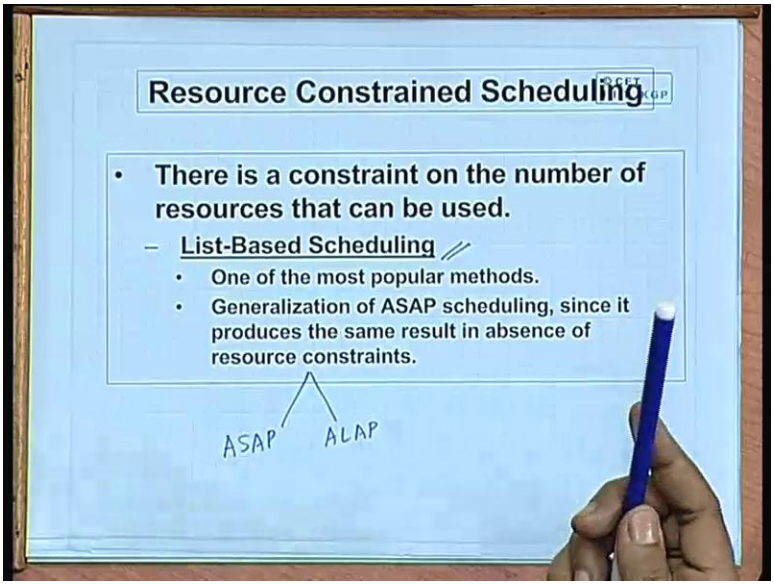
So for this I am showing the ALAP schedule out here. So in fact this in this example also you will be requiring 4 time steps t 1, t 2, t 3 and t 4 okay. So now we start from the bottom of the tree. So the choices we have now here is that we can allocate this, this and this in time step t 4 we start from the bottom. So we allocate this subtraction, we allocate this comparison and this addition. [Student Noise Time: 18:27] 4 steps I know the solution because otherwise you yourself keep this open and start from the bottom and see how much steps you require. But since I know the solution I have drawn 4 lines. So after these are complete you go one step up say that just before that what are the options you have you can do this you can do this this and this.

So all this 4 will be allocated to t 3 step so you have this subtraction this multiplication this addition and this multiplication they will now be pushed down to t 4 you see this multiplication addition and this multiplication they were in t 2 in ASAP. But now they have been pushed down to t3. Now once these are done in the step t2, you have the choice of putting in this addition and this multiplication okay. So this addition will be here and this multiplication will be here. So finally in step t 1 you have these 3 this 2 multiplications to be put in. So you put these in here so what you get here is your as late as possible schedule. So you can see if you just compare this 2

here with respect to the last time steps the operations have been pushed as far down the axis of time as possible.

But in this example both takes the same amount of time but there can be some other examples where we will see. Well if there are no resource constraints the time shall be the same but if there are some resource constraints you will see that this ASAP and ALAP may lead to different times. But you try to understand in most of the practical scenarios you do not have unlimited amount of resources. So we will have to have an algorithm which works or plays with a say some resource constraints and tries to get a solution based on that. So what we consider now is something called resource constraints scheduling.

(Refer Slide Time: 21:08)



And one of the most popular methods which people use under this called list based scheduling. This method is popular because this is simple and it can also take care of resource constraints now you will see that in this list based scheduling algorithm you use the ASAP and the ALAP schedules for obtaining some information. And using those information you try to get the final schedule. So in a way this is method is similar to ASAP because you try to allocate a computation to a time step as early as possible subject to the constraints okay. Without violating

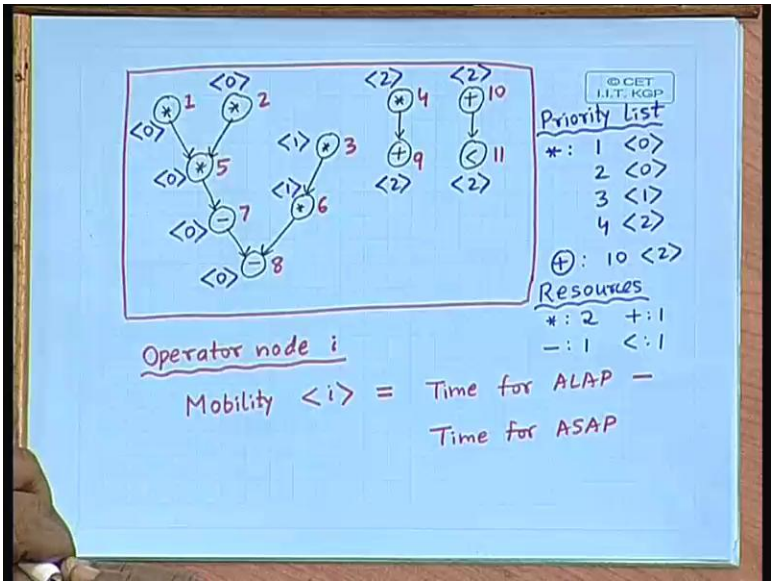
the constraints you try to allocate them as early as possible so if there is no resource constraint then this list based scheduling will be the same as the ASAP okay. So first let us see what this method is actually involved then we will just take an example to illustrate.

(Refer Slide Time: 22:22)

Now the basic concept of this method is to maintain some kind of a priority list of ready nodes. Ready nodes are those nodes means these are actually computation nodes or operation nodes say this is a computation node this node will be called ready if all is all its inputs are available. That means if it is coming from some other operation this I have been already being allocated to a previous time step. So at this point in time both the operands are available so this this operator is free to be allocated in the current step and we maintain not just a list but we also try to assign some kind of a priority. This we will see how but the concept of priority goes like this see in the dfg well you can you can understand that there will be some critical paths like in the previous example there were some paths which were of height 4 like I am showing you. In this example you see this was the critical path or this was the critical path there are 4 time steps in sequence. Now the total time was 4 so I will try to assign maximum priority to the operators which lie in the critical path. Because if you give any slack there then the total time may increase from the t from 4 to 5.

But in the other ones this addition comparison. Comparison multiplication addition here you have some flexibility. You can push down the order this one if you need so based on that we can assign some priorities to the operators that means which is the operator which is the highest priority. Means if you do not allocate it now possibly you will be ending up with a longer schedule that is the idea and during its each iteration. What you try to do we try to use up all resources as far as possible with respect to the dfg that is given to us. And if there are some conflicts if there is only adder but there are 2 additions which are ready then we will try to assign the one with the higher priority right. So now let us try to take an example and illustrate that how this actually looks like well in fact we will be using the same example as we have taken earlier but I am redrawing it for the purpose of understanding a few things. So first let us quickly draw the dfg once more quickly.

(Refer Slide Time: 25:48)



So I am just showing you side by side. So this was the dfg now I just give some numbers to each of these nodes just for future reference. 1 2 3 4 these are the multipliers 5 then 6 these are the multiplications 7 8 subtractions 9 ten and eleven these are the operators which are available. Now what we do this is our given dfg. Now what we do we look at each of the operators for an

operator node I you consider any operator node I you define something called mobility of the node I which we denote by writing some number within an angular bracket. Now this mobility I is defined as follows mobility I is defined as time for ALAP schedule minus time for ASAP schedule see these time I am just I am talking about this particular node.

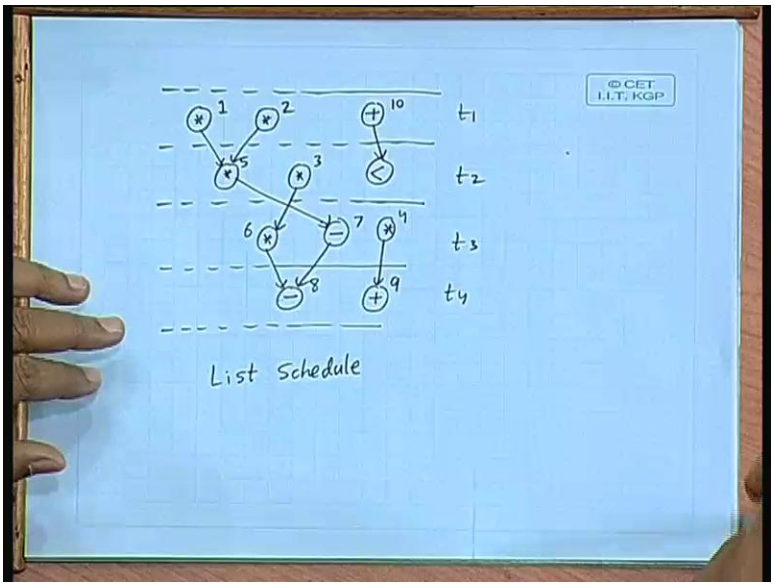
This particular node when was it assigned for example if you look at this node number one node number one was assigned time step t 1 in both ALAP and ASAP so mobility is zero. Node number 4 this was assigned time step t 1 in ASAP. But time step t 3 in ALAP. So mobility is 2. So in t 2 this way you have to construct the ASAP and ALAP schedules and subtract the times that each operation was assigned. And accordingly you get the flexibility or the mobility of each of the nodes. So I am writing down the mobility of each node side by side just for reference. These lie on the critical path so their mobility's are all zero but for these nodes there is going to be mobility of one these nodes mobility 2 okay. So now based on the mobility you can construct a so called priority list. See there are so many multiplications 1 2.

See for multiplication you construct the priorities starting with highest priority. Highest priority are the ones with less values of mobility. So you see there are 3 multipliers with mobility zero but to start with not all multipliers are ready priority list will contain only those operations which are currently ready. So there are 3 in fact 4. Yes 1, 2, 3 and 4. But 5 is not ready. So the initial priority list will contain node number one with a mobility of zero node number 2 with a mobility of zero node number 3 with a mobility of one and node number 4 with a mobility of 2. Now other than this the only other operators which is ready in the beginning is this addition. So there will another entry for this addition node number ten with a mobility of 2.

Now the idea is that as you go on allocating the operators to time steps some of the some of the entries will get remove from the priority list and some new entries will get in. Like for example if we assign this one and 2. Now this 5 will with a new node which will be coming in so if you assign 3 6 will be the new node which will be coming in. So this is the starting point and we assume also some constraints on the available resources we assume that we have 2 multipliers one adder one subtractor separate and one comparator. So under this constraint we will have to allocate fine. So now let us see that how this allocation can be done I am constructing the list

schedule based on these constraints. So I have this graph I have this priority list. So with respect to priority list. Just let me try to see.

(Refer Slide Time: 31:31)



I will try to again construct the time steps so I do not know how many steps will be required t 1, t 2 okay. So in the beginning in the first step well only I can assign these 2 from here. They are the see there are only 2 multipliers available recall. So I cannot assign all 3 multiples multiplication in time step t 1. But since the first 2 are of the higher priority I assign them node number one and node number 2. So I try to allocate all other resources the only other resource I can allocate is the addition okay. So addition 10 [Student Noise Time: 32:29] node number 4 has the priority of 2 mobility is there because it is not mandatory or it is means it is not that important to allocate this node number 4 in time step t1. Even if I push it down so it will not matter in terms of total time. But if I push node number one down it hamper my schedule.

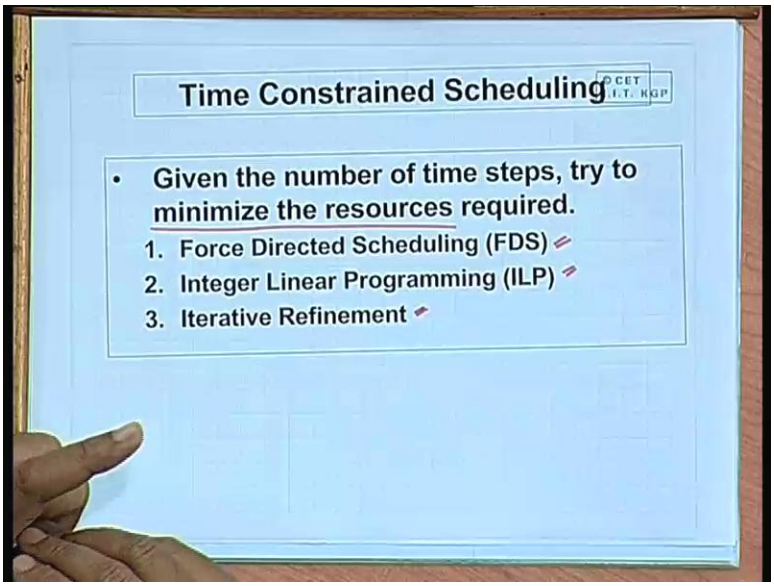
So in this way after this after you allocate 1 and 2 and ten what will happen these entries will get removed from this table and some new entries will come in. For example 1 and 2 have been remove the new entry coming is 5 6 will not come in still because 3 have not yet been completed so now this 5 will have a higher priority with respect to 3. But since we have 2 multipliers you

can you can assign both 5 and 3. These are the next 2 means high priority. [Student Noise Time: 33:33]The mobility is reduced but okay you can do that also. But that that same thing will happen for all other the number actually so means either you decrease all of them or you do not disturb any of them. [Student Noise Time: 33:50] Yeah, yeah. So just as you basically go down you can decrease these numbers yes this 3 was one with respect to time step one but as you go to time step t 2, now 3 becomes zero yes true.

Because now 3 becomes a critical node sort of. So in this way you continue you can allocate the comparator out here. So I am just showing the schedule now. Now at time step t 3 you can have the other multiplication here that is the 6 which will come from here you have the subtraction this one this subtraction this one this 7. This will come from this 5 and you can also start this multiplication which was out here this 2 with 4 this is happening because of the constraint because already 2 multipliers were allocated here. So this multiplication has to be pushed down here. So now you can have this 8 which will come from 6 and 7 and from 4 they would be 9 this is t 4.

So you see this is a schedule which is a list schedule which has been obtained taking into account some resource constraint. But in this example incidentally you require the same number of steps. But in general if you put some resource constraints it can be more than of ASAP or ALAP or so now you can see that the way you have allocated the operators to the time steps you do not require more than 2 multiplications in each step or more than one addition or subtraction right. So now this is actually how the list scheduling works now there are some other schedules also I am not going into detail of them I am just mentioning.

(Refer Slide Time: 36:17)

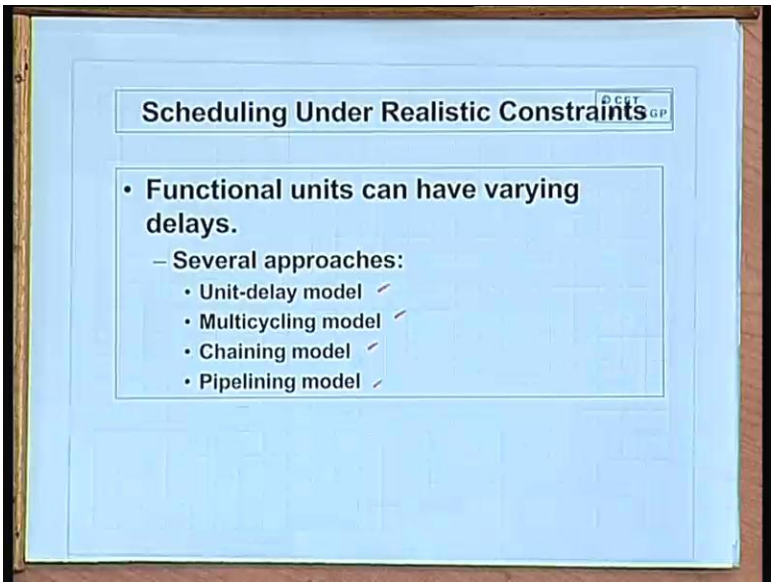


But here we try to approach from the other side here we give some idea or guideline regarding the maximum number of time steps. We can have and from there you try to obtain an allocation. And at the same time minimizing the number of resources you require. So there are actually 3 methods which are being proposed force directed scheduling. While force directed scheduling means if there is connectivity from operators to the next it will try to bring it closer by some kind of a force so the basic concept goes like this. And integer linear programming it frames the problem as an ilp and tries to solve it iterative refinement like simulated annealing some algorithm like that was also proposed. But for scheduling still list scheduling is the one which is used most widely because of its simplicity and because it gives quite good results. These are the other methods which were which were reported and explored. [Student Noise Time: 37:20] See the steps and resources required these are conflicting requirements. So if we want to want if we want to go for some kind of a compromise then you can go for some kind of iterative improvement or iterative refinement.

But what the method rescheduling does it tries to minimize the number of time steps subject to the resource constraints given the resource constraints you get the minimum time. But some of these methods do the other way round you that means you give the number of time steps as

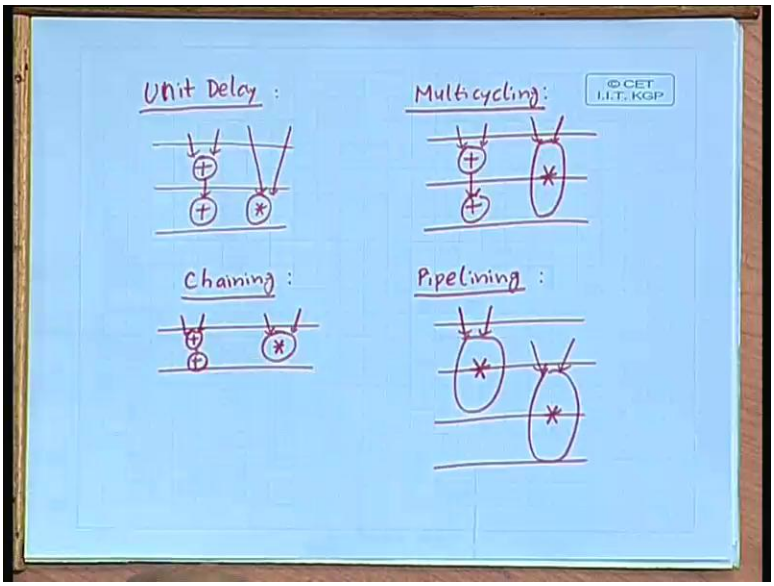
input. [Student Noise Time: 39:58]. It is true it is true that cost of hardware has gone down. So actually the resource constraint is not that much of a so we can say it may not play that much of an important role as it should earlier. But still suppose if you have a multiplier multipliers are still expensive a thirty 2 bit or 64 bit multiplier. So if you see that I am requiring ten multipliers in a step that may be still little too much. Because not only the floor area you also need to worry about the total power consumption so if you have lot of resources or complex hardware's working in parallel power will also go up okay. So now let us look at few of these points so here we had made some simplifying assumptions so far.

(Refer Slide Time: 38:58)



The assumptions were like this that all the computations can be assigned to particular control step. So means we are assuming that every that means multiplication and addition they are of the same complexity. But in practice they can have different values of delays. So we can have a number of different approaches to tackle this problem. So several heuristics were proposed I am just trying to briefly mention what these are. Its unit delay multi cycling chaining and pipe lining these are fairly simple in concept this can be incorporated with ASAP ALAP or list scheduling algorithm to get a modified schedule. So let us see what these are.

(Refer Slide Time: 39:41)



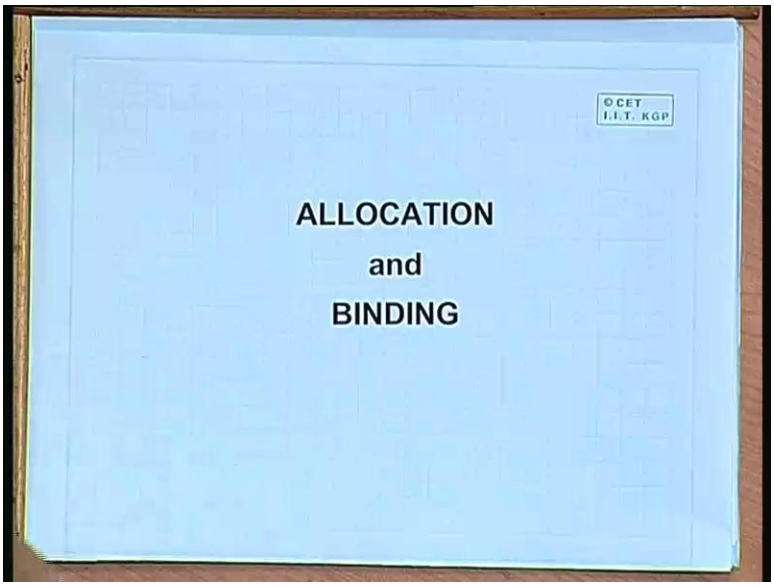
The unit delay concept says that, well this actually what we had seen so far, we assume all operators are of the same delay. So a typical schedule will look like this. Adder, multiplier so an adder and a multiplier both can be placed in the same clock step. So here you are making the clock cycle bigger basically making the clock slower in order to incorporate the slowest operation. So you can obviously this is not a very good method so what you can suggest as an improvement is something called multi cycling. Multi cycling says that you do not reduce the clock but rather you try to make the control slightly more complex like there is no problem with this part fine. But this multiplication you make it spend 2 clock cycles like this.

So now you remove that constraint that that an operation has to be allocated to a single time step. You can allocate it across a number of time steps also [Student Noise Time: 41:08] Benefit is that the basic clock frequency can be made higher suppose assuming that multiplication takes twice as much time as addition then you can have the maximum possible speed out here. Chaining does something similar but in the reverse way this says that you allocate multiplication in a time step make the clock cycle in a way where this multiplication can be done. But with respect to addition you put 2 additions in the same step you chain them together so that you have the in the same clock cycle 2 additions will be done. So some extra controlling is required now

the control unit design will become a little more complex and as an extension of this multi cycle concept you can have pipelining concept also.

Like I am giving an example suppose I have a multiplication which I am allocating like this across 2 time steps there is another multiplication whose inputs do not depend on the result of this. So I can possibly allocate it like this also there can be an overlap okay. So these are the flexibility which we have and you can understand these are heuristics normally you start with a given schedule then try to optimize using these concepts. Or in some cases you can incorporate these also in the basic algorithm right so yes [Student Noise Time: 43:24] clock frequency is not increasing but the resolution of your control unit is increasing. That means in the same clock step you are doing several things okay so you can think that there is a slower come there is a faster clock 2 clocks are running yes you can do that also yes. These are some basic concepts which you can also do together I am just independently talked about this correct. One last thing I wanted to talk about this is something called allocation and binding.

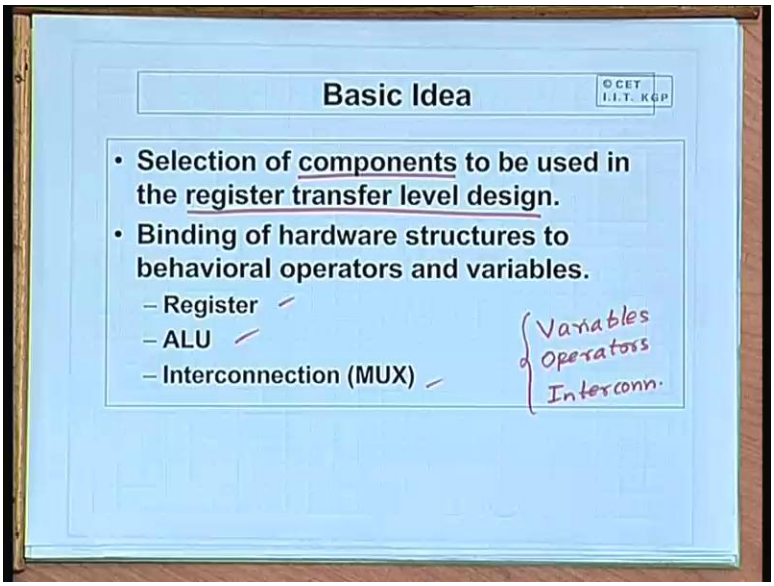
(Refer Slide Time: 43:49)



See after you have done scheduling you have obtained some kind of you can allocation of different operation to the different time steps. But after that in order to get the final hardware

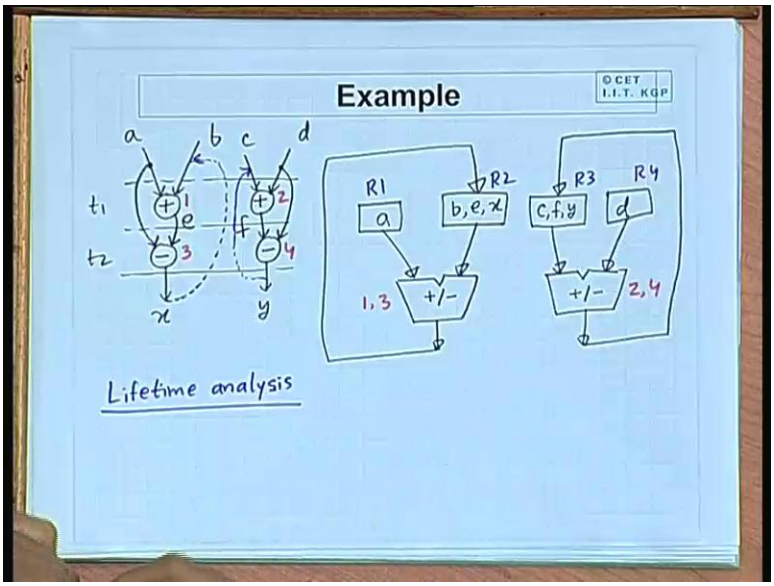
corresponding to the data path you need additional steps for allocation and binding. Now what is the basic idea?

(Refer Slide Time: 44:18)



The basic idea is that see means you have a c d f g. Still now we have a c d f g nothing else. Now you will have to select the actual hardware components to be used in the rtl level design. Because ultimately we are trying to arrive at the rtl level design. So we will have to select the basic components here there are variables there are operators there is also interconnection between them interconnects. So I will have to take care of all this things in order to arrive at the final hardware registers ALU's multiplexers variables to be reside in registers. Operators will be mapped to ALU's interconnections may require additional multiplexers right.

(Refer Slide Time: 45:13)



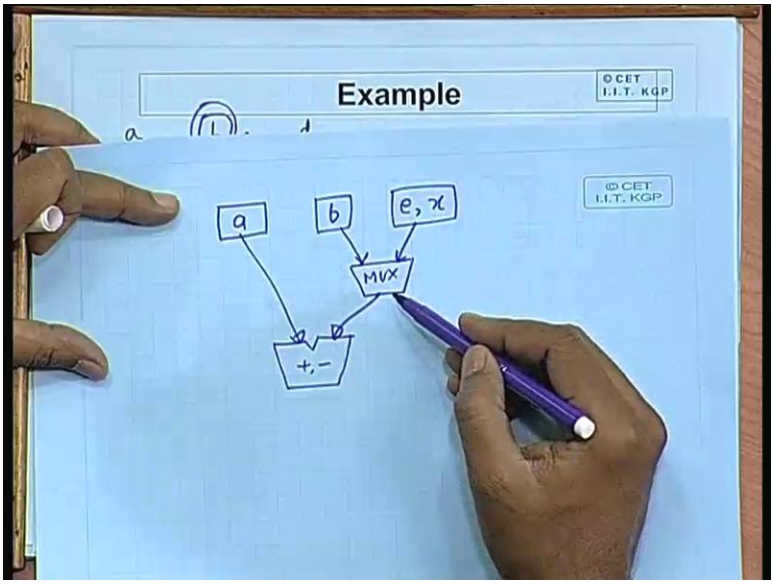
So let me take a simple example to illustrate it we will have an idea that how it works. So let me take a simple example. There are only 2 time steps t 1 and t 2. There is an addition followed by a subtraction there is one input operand a, another b. This is I am calling e and the other input to this subtraction is coming from here itself this also is a. And whatever is coming out this is say x and say here I have c and d here again a similar thing is there I am calling this f this same d is fed back and calling this y and in addition there can be a feedback like here whatever value of x you are computing. This will be fed back to this b in the next iteration similarly whatever you are computing here y. This will be fed back to c in the next iteration. So from this dfg you straight away you can come or arrive at an ALU design you see there are 4 computation steps. Let us call them one 2 3 and 4. Now these are adders and subtractors.

Typically addition and subtraction can be done by the same hardware unit same ALU. So I am assuring that there are 2 adder subtractors available with us. These are the ALU's we need 2 ALU's because 2 things has to be done in parallel both will be having the capability of addition and subtraction. Now in terms of the operators I am mapping 1 and 3 out here and I am mapping 2 and 4 out here. Now the first input to this will always come from a right. So I assuring that there will be one register which will be storing the value of a in a dedicated form. But the other

registers which we have out here. Now in terms of this it can store b it can store e or this x when it is fed back x b e x this is the second input and the output of this, whatever is coming out this will be loaded back into this. So there have to be a data path from here back to here similarly the first input to this ALU can be c f or y the second one will be again that is similar to this mirror image d. And whatever is coming out that will have to be loaded back here.

So see here what I am shown is that I will require 4 registers r one r 2 r 3 and r 4. e and x they can share a register while if this is the complete computation where you do not need b anymore. Then this is valid through some analysis these are called means life time analysis of variables. These are fairly standard techniques used by compilers that you find out that during which time step a variable is active. Active means you need to keep them in a register. But if the life times of different variables are disjoint they can share a register. So in this allocation and binding you try to find out which are the variables that can share a register. While if they can share a register you can put them together. Now assuming b e x and cy can share a register this is the optimized data path. But suppose b is required some where later also just assumes that b is required some time later you cannot overwrite the value of b with e or x. Then at least for this part of it the modified data path would have been like this.

(Refer Slide Time: 50:05)



So instead of writing like this what I would have got is that I would have one register for storing a. I would have one register for storing b also because b is required as I said and there will be one register which will be sharing the values of e and x. Now they will be a multiplexer I would need a multiplexer out here which will take either the value of b or e x and then it will feed to the ALU. So one input will be coming from here other input will be coming from here. So after the life time analysis if you find out that there are some variables which cannot share a register. Then you may need to allocate dedicated registers and you will be requiring some multiplexers to select one of them to the output. So these are the different things which need to be done in the steps of allocation and binding I am not going with the details of the algorithms. Because there are number of algorithms and heuristics which have been proposed. They are used primarily to optimize this data path that is what is the minimum number of registers required if you cannot share a register?

What is the minimum number of multiplexers required and so on. So using these kinds of operations in sequence you can finally end up with a data path which will contain the functional units which will contain the registers to stored the operands and which will contain also interconnections and multiplexers to route the different values in the diff different places. Now with this we finish our discussion on synthesis for the time being starting from our next class what we will talking about. We would be talking about this step or the steps we follow once the synthesis is over means we go towards the backend design. We now plan that how we will be mapping our design into our target you can say target may be silicon it may be it can be anything. So we have to plan how we will have to target our netlist which we have obtained into our target. Typically we will assuming our target to be a silicon. You want to design an ASIC but the different steps in the algorithm will be looking it in our next classes. Thank you.