

Design Verification and Test of Digital VLSI Designs
Prof. Dr. Santhosh Biswas
Prof. Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

Module - 2
Scheduling, Allocation and Binding
Lecture - 1
Introduction to HLS: Scheduling, Allocation and Binding Problem

Welcome to module 2 lecture 1. So, in module 1, what we have generally discussed, so we have seen that there is a digital VLSI design flow how it works and then we have seen that we require cad tools or automated tools, which can automate the design flow. Because I mean it is very it is a digital designs you start for specification then you have to go to the final product. Then it involves lot of steps like high level synthesis, gate level synthesis, layout and then final test pattern generation and so forth; which are extremely complex and cannot be done by a human being on by in a by manual procedure.

So, we require automatic algorithms to solve solve all the problems for us, and then we have seen told that in the next form the next module onwards that is from module 2 onwards. We will be looking into cad tools or algorithms how we can develop good algorithms which can enable our designer to take help of those cad tools and he can get a very smooth in a automated design flow. So, that is the basic goal of this course that we will be learning about digital design flow, verification flow and test flow and how we can develop cad tools which can help us automate this flow. So, that is the very, very basic emphasis of the course.

Then we have seen that for any a procedure or any algorithm you have to if you want to process some data from the algorithm. So, data should be represented in a, what do you say a formal manner by using some formal model, so that it can be easily transformed or it will be easily processed. Then we have seen that control and dataflow graphs are very important I mean are very important parts, I mean is a very important actually data structure or you can say a model which can represent our design specifications.

We represent in hardware definition languages and then we can very easily represent them as control and data flow graphs, which can be processed by all the algorithms or all the cad tools, which will see down the line, so that you can get a automated design flow.

(Refer Slide Time: 01:53)



So, in this course in this lecture I mean second I mean module 2, first lecture what we are going to see where. Or in other words say I mean module 2 will be basically dealing with high level synthesis that is we will be taking a CDFG or a verilog code at in the beginning. And in the end of the flow we will generate, what we will generate a architectural design for this for this specification.

So, this is actually called the high level design flow, so in the high level design flow as we will see we will consist of the problem schedule allocation scheduling allocation and binding. So, we will be looking at algorithms, which can automate this three problems. So, in the first lecture we will introduce the problem of high level synthesis that comprises three parts scheduling, allocation and binding. So, first we will see the problem and then from lecture 2nd, 3rd, 4th onwards in this module we will see how we can design cad tools which can automate this process.

So, we saw in the last lecture that first step is to obtain for VLSI design, we have to design VLSI design we start with the specification and we first obtain is a RTL level circuit. So, what is the idea, so we write a specification in verilog or VHDL, so that because this is unambiguous and if you write it in English language, so it may have ambiguity and all.

(Refer Slide Time: 02:35)

Introduction

- Any VLSI design we start with specifications and the first step is to obtain the Register Transfer Level (RTL) circuit.
- RTL circuit is obtained from specifications using High Level Synthesis (HLS) algorithms. As specifications are processed by HLS algorithms, they need to be represented using some modeling language.
- Control and Data Flow Graph (CDFG) is one of the most widely accepted modeling paradigm for specifications that are processed by HLS tools.
- Transformation techniques in the CDFGs, which lead to efficient circuit implementation in terms of area, frequency, power etc. HLS takes as input, the optimized CDFG, performs Scheduling, Allocation, Binding and generates RTL design.
- In this module we will study algorithms pertaining to these steps—Scheduling, Allocation, and Binding. To start with, in this lecture, we introduce HLS and problem definition of Scheduling, Allocation and Binding.

So, we generally prefer to write the specification in verilog or VHDL kind of HDL hardware definition languages and high level synthesis will transform it to what do you call the RTL circuit or a architectural block level diagram. So, to obtain this from specification to RTL we require high level synthesis algorithm. So, high level synthesis basically transforms your specifications for RTL as specifications as we have already seen, are generally represented in terms of some kind of a control and data flow graphs. So, CDFG is one of the most widely accepted tools for this and then what do we will do is that we will take a CDFG and we will convert it into RTL level circuit or block level architecture. So, we have to, so high level synthesis actually takes as input the optimized CDFG and performs scheduling, allocation and binding.

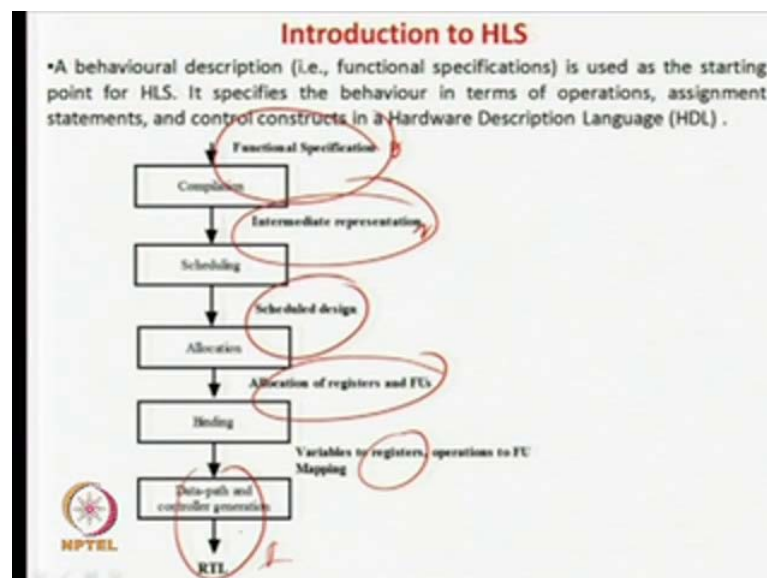
So, why do we talk about optimized CDFG because in the last lecture of that is the third module of the first sorry third lecture of the first module. So, what we have seen sometimes I mean the rtl designs or actually, sorry the what you call, the verilog designs or specifications in terms of verilog designs are generally written by human beings. So, you express your specification, which is the manual activity. So, the it from the designers experience, he looks at the specification and then he writes a verilog code or hardware code, hardware definition language code, so which can satisfy the specifications.

Now, we have seen that as a human procedure, so it may have lot of inconsistency it may it may have redundant course like, it may have dead course it may have moved in variant computation and so forth. So, if you can do a preprocessing like we have seen compiler, I mean compiler based optimization, then we have seen graph based optimization. Then we have seen the last level like what do you call the, I mean technology library, that is fabrication library or tech library based optimization.

So, similar optimizations are I mean when we have reduced the eliminated redundant course or we made our our design more optimized by using a increment instead of an adder and so forth. So, all this about optimization steps we have already seen in the last lecture. So, they are applied actually to the main specification, so that we get a very optimized specification in terms of hardware power, area and frequency.

So, that is actually sometimes called a preprocessing step. So, we take a we take a CDFG which is comes to the input form the specification then we go for the preprocessing step that we optimize it and then we go for scheduling, allocation and binding problem. We generate the RTL design or in other words which is your architectural level block level design, so that is the idea of high level synthesis.

(Refer Slide Time: 05:26)



So, high level synthesis will take you preprocess design. So, preprocess means you have already done the optimization on the CDFG's or in the HDL course and finally, the CFG or your input will be schedule, allocated and binded. And finally, we get a RTL level or

we get a block level design, so that is the basic idea of high level synthesis. So, in this lecture we first see what is scheduling, allocation and binding problem.

Then in the next lecture onwards we will see how we can develop a automated cad tool which can solve the problem. So, so now, so we other words we are not looking at the high level design flow. So, whole broad level VLSI design flow already we have seen we start with high level design then we go for gate level synthesis, then we go for back end design and and from from back end design you go for fabrication and also in in mean time we also generate the test pattern then it is called the test flow.

So, now we will go in depth in one of the flows one by another, so first we are looking at the high level design flow. So, what do we have we have functional representations that we say what we want to do, like we want to add, we want to implement the multiplier etcetera, etcetera. Then we go for a intermediate representation, so what do you mean by intermediate representation, in this case it was a control and data flow graph.

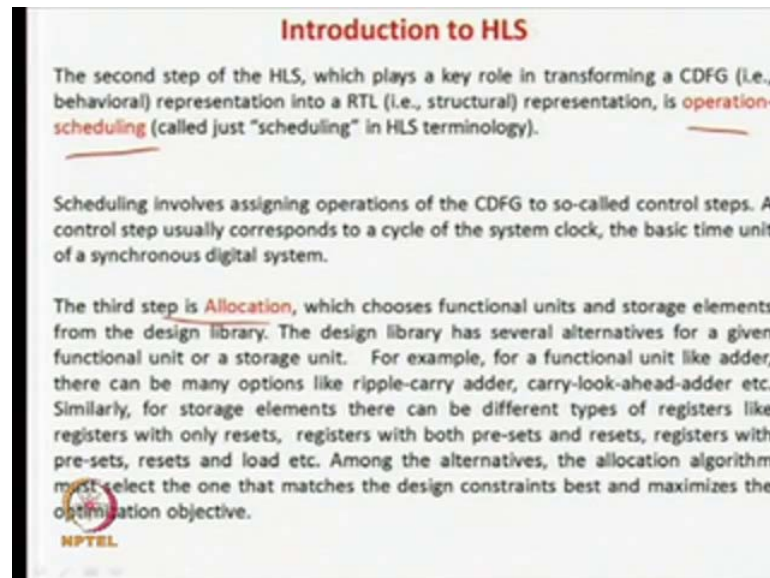
So, we were representing in a manner, so it is very easily usable by your automated cad tools. Then actually here also sometimes with your lot of preprocessing as well here you have seen to get an optimized design, because functional functional specifications what we are writing, basically are manual driven procedure. So, they are done manually, so sometimes you require optimization.

Then the first stage is that we go for scheduling, so we will all see in details what the scheduling means. Scheduling means in case for each of the operations we give a time step that is operation will be done or in this time step and so forth. Then whenever we have schedule all the operations, then we actually allocate some hardware units for this, like for some case, we may require a hardware and also for some case we require a multiplier and so forth. And finally, we are actually allocating what do you call the variables will be allocated through registers then there will be multiplexers and so forth.

So, first you go for scheduling then we allocate it to the hardware and then we then we bind the operations to the hardware like say addition 1, addition 2 will be done by adder 1, addition 4 and 5 will done by additional adder 3 and so forth. So and finally, we go for control and data path generation and that is we will see what you mean by control and data path, and finally we get the RTL design or what do you call your block level

architecture design, so this is the basic flow. And now in today's class or today's lecture we will look at all the problems in details in a more formal way.

(Refer Slide Time: 07:22)



So, as I already told the first step in high level compilation is an internal representation. So, it is a control and data flow graph, then we go for preprocessing step, so what do you mean by preprocessing step; that means, you go for some kind of optimization. So, that the code is more optimized or it have though not even redundancy or it is more what you call it uses better hardware elements like incrementers or shifters greater than divide adders or dividers multipliers kind of a stuff.

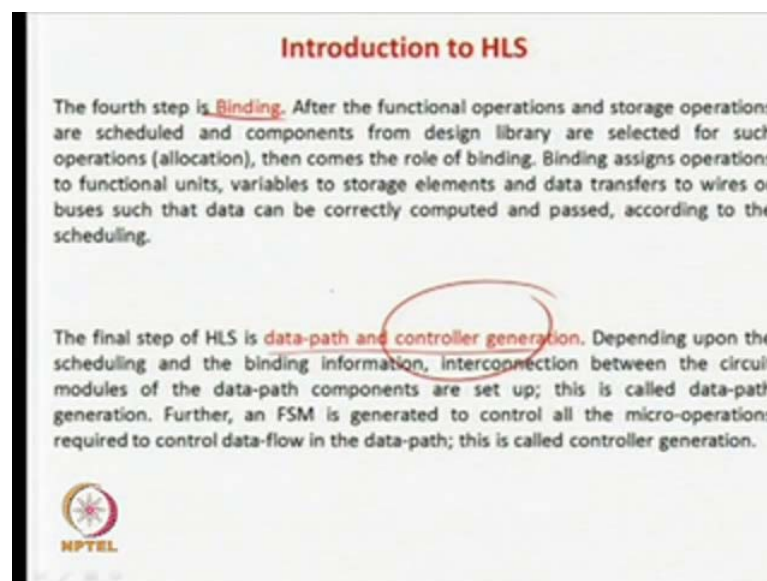
So, we are using better we are we are keeping the circuit very optimized, because we do not have any redundancies, we do not have any what you call dead course etcetera. And finally, we are using very good quality blocks, so this type of optimizations we do it. So, that is actually called the preprocessing of the high level or your specification.

Then what we do then we actually go for operation scheduling, so as I already told you what do you mean by operation scheduling. So, in a control and data flow graph we have some operation nodes, so each of the operational nodes we should assign it to one you should assign it to thumb time step that is this operation will be done in time step 1, this will be done in time step 2 and so fourth.

So, that is actually called operation scheduling, all the scheduling in high level synthesis is terminology. We will see in details the actual formulation of the problem then this is actually second step, then third step is called allocation. Now, that is what you mean by allocation now we have an adder, so we say that this is the adder, how we want to implement the adder.

So, you can use it use a use a hardware adder like ripple carry adder, also you can use a carry look ahead adder or carry save ahead adder. So, in the allocation step for each of the operational load or each of the operators like adder, multiplier, subtractor we allocate a hardware block. And we and we can also say that in this step we actually physically look at the design library what are their components available there or what are the adder multipliers, divider, subtractor available there. And we try to allocate that to each of the operational loads in the CDFG or in the scheduled CDFG is the better way to say.

(Refer Slide Time: 09:02)



Then in the fourth step is actually called the binding, now say we have taken 3 adders only and 2 multipliers only, but then may be 10 additions, which can be done. So, obviously, all the additions cannot be done in one step we all see that, because it is one may not be possible, even if there are 10 additions that are required to be done in the CDFG you cannot do it in one step. So, we put all the three or four adders may be and we try to reuse the adders.

Now, reusing the adders means we can say that operation 1, 2, 3 or addition 1, 2, 3 will be done by adder 1 and addition 3, 4, 5 will be done by adder 2 kind of a thing, so this is actually called binding. So, in binding what happens each of the operations which are already scheduled and allocated. So, you will actually bind it or physically assign it to one of the hardware, which is have been which have been taken in the case of allocation step.

The allocation step will take some hardware adders, multipliers, subtractors whatever from the library. In binding step we say that operation 1 will go to this hardware, operation 2 will go to this hardware and so forth. And also we take care of registers to show the variables etcetera, all these things are done in case of binding; and finally, we will go for data path and control path generation.

Now, we will see I mean what do you mean control because if you say that one adder 1 will do operation 1 and operation 2. So, in case of operation may be a and b should be fed to the adder in case of operation 2 c and b may be fed to other fed to this adder block. So, we require lot of multiplexing arrangement and all, so depending on which time step you are or which which are the two variables you want to add in the adder. So, you may require to generate some control signals. So, in the final step we generate the control signal and the data paths. So, this is the actually control I mean this is in a nut shell is a high level synthesis flow, now what we will do.

(Refer Slide Time: 10:30)

Scheduling Problem

The scheduling problem involves determining the sequence in which the operations are executed to produce a control step schedule, which specifies the operations that execute in each control step.

Let O be the set of all operations to be scheduled, which are obtained from the HDL code. If there is an operation $o_i \in O$ which depends on the result of another operation $o_j \in O$, then o_i must finish its execution before operation o_j can begin. In such a case we say that there is a data dependency between the two operations o_i and o_j and o_i is an immediate predecessor of o_j . Data dependency results in a precedence constraint between the two dependent operations in scheduling. In other words, an operator can be scheduled only after all its predecessors are scheduled.

The slide includes a diagram showing two operations, o_i and o_j , represented as circles. An arrow points from o_i to o_j , indicating a data dependency. The text explains that o_i must finish before o_j can begin, creating a precedence constraint.

NPTEL

Now, we will take we will take each of the problem in a more formal way and with examples we will try to explain what do you mean. So, then it will be the now the definitions will be a bit fuzzy to you, but that is actually we are telling it in a very layman kind of a language. Now, we will come to a in a very depth level or what you can say we will come we are coming to a more technical level of doing it.

So, now, what is the scheduling problem, so already said that is a CDFG we have some operations. So, this procedure what it will do it will try to allocate a time step to each of the operations. So, what it says that let O be the set of operations, which are to be scheduled. So, set of operations you can easily get from the CDFG and each individual operation we say that O_j belongs to O .

So, now what in scheduling problem what you do you will take each operation from the set of operations say O_i is an operation which belongs to capital O . So, each of them has to be given a time step. So, you can say that these happens in time step one, this should happen in time step two and so forth. But one thing you should be very careful in doing that if there is an operation O_j which depends on the result of O_i . So, it will happen that output of O_i is actually controlling output of O_j . So, in this case you should be very much assert if that this has already been scheduled before this before the second guy is scheduled, why because the output of this depends I mean the output of sorry O_j depends on the output of O_i .

So, O_i has to be computed first then you can compute O_j . So, it is very obvious that you have to give a prior times time time what you call time step to O_i compare to O_j , then you can I mean you can easily do this, so this actually called data dependency. So, if the output of O_i actually controls the output of O_j or O_j is dependent on the output of O_i , so they are data dependency. So, when you are scheduling it, you should take very much care that or it is a mandatory requirement that this partial order or what you call this dependency is satisfied.

(Refer Slide Time: 12:11)

Scheduling Problem

For any HLS platform, there exists a module library comprising circuits for different functionalities like adder, multipliers, registers etc. Further, the library also has information regarding different parameters of the modules namely, frequency, area, power etc. Let T be the set of different types of modules that are available. For a given operation o , the type of the operation is determined by a type function $Ty: O \rightarrow T$; $Ty(o) = t$ implies that operation o can operate on module of type t .

Based on the above basic formulations we will discuss the following four types of scheduling problems

- Un-Constrained Scheduling (UCS) problem
- Time Constrained Scheduling (TCS) problem
- Resource Constrained Scheduling (RCS) problem
- Time-Resource Constrained Scheduling (TRCS) problem

Now we elaborate on each of these types using the simple example expression $(a+b+c+d)*e$.

Otherwise I mean obvious it is very obvious that if some O_j depends on O_y and O_y is not yet scheduled then if you try to schedule O_j before O_y . So, you may get inconsistent result right, so that is the very basic idea. So, that is actually data dependency and then has to be very much taken care or the mandatory requirement in case of scheduling. So, let us now see that I mean they are different type of scheduling problems, so we will see one by one. So, one is called the unconstrained scheduling problem, so there is no constrained.

So, there is only one constrained actually it should not unconstrained there is only one constrained that is data dependency has to be managed. Otherwise there is no constrained like you have to compute all the operation in four times, say there is no requirement that you require, so many adder, so many subtractor, there is no resource constrained there is no time constrained etcetera. So, this is called the unconstrained scheduling, but of course, you have to follow the dependency.

Second thing is called actually time constrained scheduling. So, what are he say he say that you will give a time constrained that you have to go for scheduling should be possible in three times four times. So, sometimes you may get a successful result saying that yes it is possible and sometimes you may say that this is not at all possible, why it is not at all possible, because you may say that we we have given a constraint of t time step and he is not doable in three times step.

Then there is a resource constrained time scheduling in which case time is not a matter, but we will tell you that we have only 1 adder, only 1 multiplier on, only 1 subtractor kind of a thing. And you have to reuse them again and again and you have to get the solution or you have to get a scheduling you have to schedule all the operations. So, it may happen that you can take a very long time to do it, because there may be a constraint on the resources.

And sometimes the resource constraints also may be something like that you may not be able to solve your problem. Like for example, it may happen that you require a subtraction operation and you say that I do not have a subtractor then; obviously, is an infeasible schedule problem and you cannot schedule the schedule the CDFG at all. And finally, we have time and resource constrained scheduling problem in which case both of the problems are together; that means, you have a different time constraint as well as you have to have definite resource constraint.

So, I mean this is a very I mean most practical time scheduling problem in most of the cases we use resource and time constrained scheduling problem, because we say that I have to do all the operations in so and so time. So, many time steps as well as I have this, this, this is the maximum number of resources you cannot have anything more than that. Now, you can understand that sometimes our constraint may be too tight and you may need to an inconsistent solution, then you have to relax one of the one of the criteria.

Like you may say that I have I have to extend my time step or I have to put more hardware to get my solution. But in all of them you have to understand that, one constraint always remains that is the data dependency you cannot anyway violate the data dependency, so that is the idea. So, we will now elaborate all the examples and all the problems using this expression $a + b + c + d + e$. So, this is the basic expression we will try to do it is an unconstrained scheduling problem.

(Refer Slide Time: 14:41)

Unconstrained Scheduling (UCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$ and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T and obeying the partial order.

The diagram shows a data flow graph with four operations (circles with '+') and three functional modules (rectangles). The operations are labeled 'a', 'b', 'c', and 'd'. The functional modules are labeled 'temp1', 'temp2', and 'temp3'. The graph is divided into three steps: Step 1, Step 2, and Step 3. Handwritten red annotations include 'x1', 'x2', 'x3', 'x4' and arrows indicating dependencies. The NPTEL logo is visible in the bottom left corner.

So, what is said that we have a set of operations, so so O_1, O_2, O_3, O_4 are set of operations and T is the set of different type of functional modules available. So, in this case say we can say that T_1 stands for adder and T_2 stands for multipliers and T_1 belongs to capital T and also T_2 belongs to capital T this you have to know. So, this means o was small 1, small 2, o_2 and o_3 are the operations, so all they belong to capital O and t is the set of different types of functional modules, so this is the idea.

So, we have T_1 and T_2 only two types are required for that we want capital T and we have to define a function Ty actually what it does is maps O to T that is each it maps each of the operations each of the operations to a to a hardware module and a and a partial order as you constraint. So, what let me see let me tell you what is the idea. So, to solve this what we are given, so we are given o that is the set of operations in this case 1, 2, 3, 4 are the operations and the different types of functional units available. So, in this case T_1 and T_2 are available or required because this multiplication addition only and a partial order on O is also given.

So, what is the partial order, partial order in this case you can see that you see that without computing O_1 and O_2 you cannot go for O_3 . So, that is what this is the data dependency that you have to compute these two first, then you can go for this, but O_1 and O_2 do not have any data dependency, you can do any one before other or after the other, because there is no dependency on this; but you cannot compute O_4 before O_3 .

Similarly, you cannot compute O_3 before O_1 and O_2 , so this is the partial order order data dependency. Now, what you have to go for the this unconstrained time scheduling is that a feasible schedule schedules you have to schedule all the elements in O taking the appropriate modules from T that is you have to schedule O_1, O_2, O_3, O_4 that is you have to give some time steps to this. And also you have to say that for O_1 , I allocate T_1 for O_2 I allocate T_1 and like for O_4 I allocate T_2 , because T_2 is a multiplier and all other say adder and you should obey the partial order that is data dependency and data dependency has to be followed.

But, now it is unconstrained time scheduling problem, so unconstrained problem. So, what is the idea we do not have any requirement that how many adders we can give how many multipliers we can give or what are the time steps, we did not at all bothered. So, this can be very well unconstrained time schedule, which is given it says that in O_1 I will allocate T_1 , O_2 I will allocate T_2 , O_3 I allocate T_1 because they are all adders and O_4 I will allocate T_2 it is a multiplier. And O_1, O_2 and O_3 I will do in time step 1, O_4 I will do in time step 2 and O_3 I will do in time step 3.

So, I have given all the operations some type of hardware as well as I have given time steps, as well I have said that as well as I have taken care of the data dependency that is the partial order that dependency I have satisfied. So, this is a schedule unconstrained schedule because I have not thought that what would be the time limit maximum or what is the number of adders require I do not think, but I give a feasible schedule for this one.


So, this is you consider unconstrained schedules for this expression now how many adders, multipliers, subtractors you require if you think. So, you can think you can see that I require 2 adders because they are happening in parallel in one time step. Now, in time step two you can either use this or you can reuse this and then we require one multiplier, so resources require two adders and one multiplier.

(Refer Slide Time: 17:51)

Unconstrained Scheduling (UCS) problem

As the schedule is unconstrained we need to see that all elements in O are scheduled, appropriate modules from T are taken and partial order is maintained. In the above example, there are four operations (3 additions denoted as o_1, o_2, o_3 and 1 multiplication denoted as o_4), all of which are scheduled. Let the library have two types of resources, adders (denoted as t_1) and multipliers (denoted as t_2). It may be noted that appropriate modules from T are taken— o_1, o_2, o_3 are assigned to t_1 (i.e., adder is assigned to addition operations) and o_4 are assigned to t_2 (i.e., multiplier is assigned to multiplication operation).

As the scheduling is unconstrained, we consider two adder modules (one for o_1 and the other for o_2) and a multiplier module (for o_4). The adder module for o_1 can be reused for o_3 . The control steps required is 3.



So, I mean, so whatever I discussed actually is written over in the slide that is 3 additions O_1 and O_2 and one multiplication this thing, so we use T_1 is the adder. So, O_1, O_2, O_3 are all assigned to T_1 O_4 is assigned to T_2 and then actually we depend time steps that O_1, O_2 in time step one O_2, O_3 in time step two and O_4 in time step four, so that is what has been done. So, we require three control step and we require 2 adders and 1 multiplier module for this one. So, this is an unconstrained time schedule.


(Refer Slide Time: 18:21)

Time Constrained Scheduling (TCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$, a time constraint (deadline) D (i.e., maximum control steps) and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T , meeting the deadline D and obeying the partial order.

It may be noted that schedule of last example satisfies all requirements of unconstrained scheduling problem and along with that, it satisfies the three steps deadline (of timing constrained problem). Further, we may note that we cannot have a successful schedule if timing constraint is two control steps, as it will lead to violation of partial order. The time-constrained scheduling required two adders (for o_1, o_2 , which is reused for o_3) and a multiplier (for o_4).

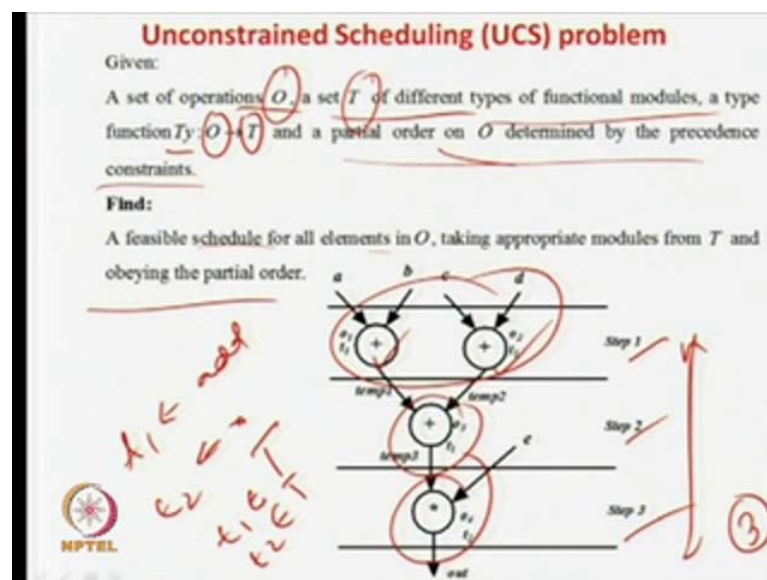


Now, we go for a time constrained, so in this case we should give some time constrained that you should say that we have to do all these things like O will be given, T will be given, a function will be given that is actually we have to say that this type of like for adders you have to apply to addition operation you have to apply to adders, the multiplication operation we have to have multipliers and so forth that is for I mean.

So, so in terms of operations and different type of functional modules type function all these things are given along with that all these things are available in unconstrained time scheduling problem also. But in in in addition of that we are also given a deadline that you have to say that you have to do everything in, so many number of time steps. We are not allowing more than say three time step or more than four time step is not allowed.

So, you are giving a deadline and everywhere this partial order is there, because you cannot; obviously, violate your timing constrained. So, that is, so now, what you have to do you have to find out a feasible schedule, for all the elements taking appropriate modules from T meeting the deadline. These are new constrained that is added because of time and obeying the partial order that is what you have to do.

(Refer Slide Time: 19:22)




So, in the last example if you look at it, so in this case the time constrained is if you say that my time constrained was three. So, this unconstrained scheduling problem was converted can be converted into a time constrained three time constrained three scheduling problem, because here we are able to do it everything in three time steps.

(Refer Slide Time: 19:27)

Resource Constrained Scheduling (TCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$, resource constraints $\max_k, 1 \leq k \leq |T|$ for each functional module of type $t_k, 1 \leq k \leq |T|$ and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T , meeting the resource constraints for each functional module type and obeying the partial order.




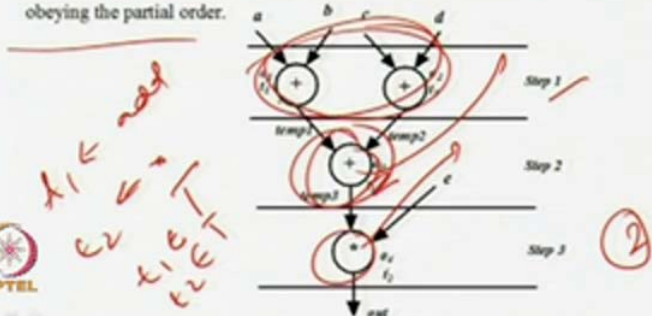
Now, let us see, so let us define, so in last example you can say that it was an example of our time constrained scheduling, when your time requirement or deadline was 3. So, what is the resource, so in this case one more thing you have to observe. So, like in this example, so this is a scheduling a problem with time constrained three those is satisfied. So, if I tell you that I want similar I mean scheduling schedule to be done, but my time constrained is 2.

(Refer Slide Time: 20:03)

Unconstrained Scheduling (UCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$ and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T and obeying the partial order.



So, you will find out that I will land into an inconsistency because we cannot do everything in two time steps, because at least ah for I mean you cannot compute this in parallel with this one because results of t_1 and t_2 , sorry O_1 and O_2 are required to get the value of O_3 . So, you cannot put this guy here it is not possible, so you require at least one step for these two one step for this one. And similarly this one also you cannot put parallel over here because I mean result of O_3 is dependent I mean O_3 controls the result of O_4 , so you require minimum three time steps to solve the problem.

So, if you say that I want to solve the time constrained scheduling problem with time step two, so you will not be able to do it, so if you say it is three or more then you will be able to solve the problem, so this was about your time constrained schedule problem.

(Refer Slide Time: 20:45)

Resource Constrained Scheduling (TCS) problem

Given:
 A set of operations O , a set T of different types of functional modules, a type function $T_y: O \rightarrow T$, resource constraints $\max_{k, 1 \leq k \leq |T|}$ for each functional module of type $t_k, 1 \leq k \leq |T|$ and a partial order on O determined by the precedence constraints.

Find:
 A feasible schedule for all elements in O , taking appropriate modules from T , meeting the resource constraints for each functional module type and obeying the partial order.

HPTEL

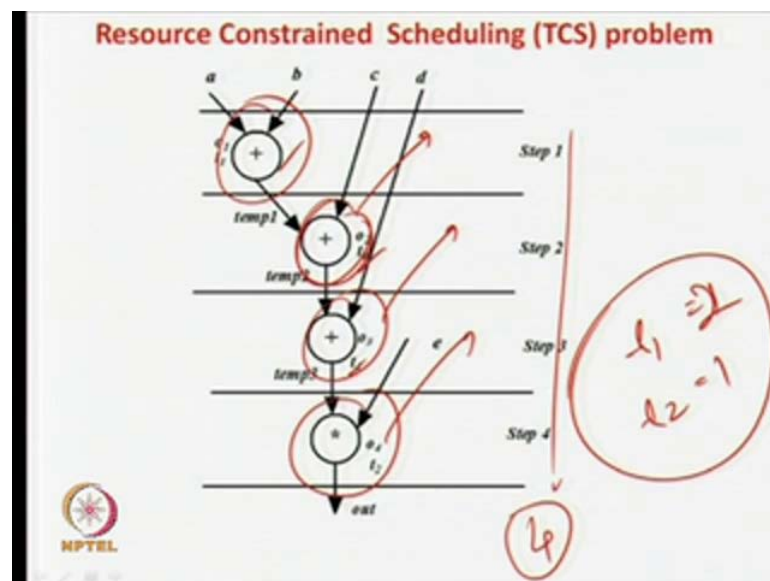
Now, we go for the resource constrained scheduling problem. So, resource constrained time scheduling problem is very similar we have O T , and then the function it may have solve the what you call the operations to different type of hardware's and along with that you have a resource constrained. So, what is the resource constrained says it says that k is the type of operator, like it adder, it can be multiplier, it can be subtractor and it has a limit it says that it is 1 to max of t that is what. So, it says that you can have a say I will use maximum two multipliers. So, k is the type, so like in this case you see for each functional type of T k you can have a function like this oh sorry you can have a inequality like this it says that T it says that T k say you can say that equal to 1. So, that

means, the for the particular type of hardware operator like like in we can say T 1 may be adder in this case.

So, we can say that T 1 is equal to max of T 1 say 2, so it says that maximum you can have 2 adders in your circuit. So, if I say that T 1 equal to 3, so you can say that maximum 3 adders can be possible. So, like T 2 stands for a multiplier, so you can say that T 2 equal to 1, so it says that maximum you can have only 1 multiplier in this one. And so how many you also what is the limit, so minimum should be one. So, if you do not have any hardware any multiplier no problem can be solved. So, it says that T k is actually a constrained it is says that what is the maximum number of operators available or maximum number of hardware each hardware type that is possible. So, that is possible, so it will it starts from 1 to max of T.

So, that, so T is the maximum number available. So, you have to specify that along with the number of operations different type of hardware elements, partial order and also we have to tell, what is the maximum number of elements per hardware type you are having you want to put it that is one time resource constrained. So, you are putting a resource constrained there and then you have to perform a, I mean feasible schedule for all this all the operators for all the modules in T as well as we have to need the resource constrained here.

(Refer Slide Time: 22:40)



So, in the last case we were talking about the time constrained, now we are having a resource constrained. Now, let us assume the case resource constrained what I say that maximum T 1 is equal to 1 and T 2 is equal to 1. So, we can have a maximum 1 adder and maximum 2 adder time constrained is not there; that means, what you can have on you can do parallelly one addition at a time. So, feasible schedule is this is one T 1, this is one T 1 and this is one T 1. So, what we do in first step we go for operation O 1, then the same adder we will use for operation T 2 in the same adder you operation T 3 and finally, the multiplier for T 4.

Now, you can see that this is our feasible schedule because now O 1 has been given T 1 O 2 T time step two O 3 time step three and O 4 time step four. So, what are you require four time step, because we have the constrained is not 2 adder, but the constrained is a single adder, so you have increase in your time step.

So, if you remember last example, so if our constrained was at adder is two then we could have shifted this guy over here then this thing will be shifted here and this thing will be shifted here and in a time constrained requirement will be at two. So, now because you have gave an severe resource constrained, so your time actually your time step have increased.

(Refer Slide Time: 23:41)

Resource Constrained Scheduling (TCS) problem

Illustrates a resource-constrained scheduling involving, one adder and one multiplier, for expression $(a+b+c+d)*e$.

As the schedule is resource-constrained we need to see that all elements in O are scheduled, appropriate modules from T are taken, partial order is maintained and resource utilization does not cross the limit.

As there is one adder module (for o_1, o_2, o_3) and a multiplier module (for o_4), we cannot schedule o_1 and o_2 in one control step. So we schedule o_1 in step1 and o_2 in step2. To maintain the partial order, o_3 is scheduled in step3 and o_4 is scheduled in step4; it may be noted that these operators cannot be scheduled earlier.

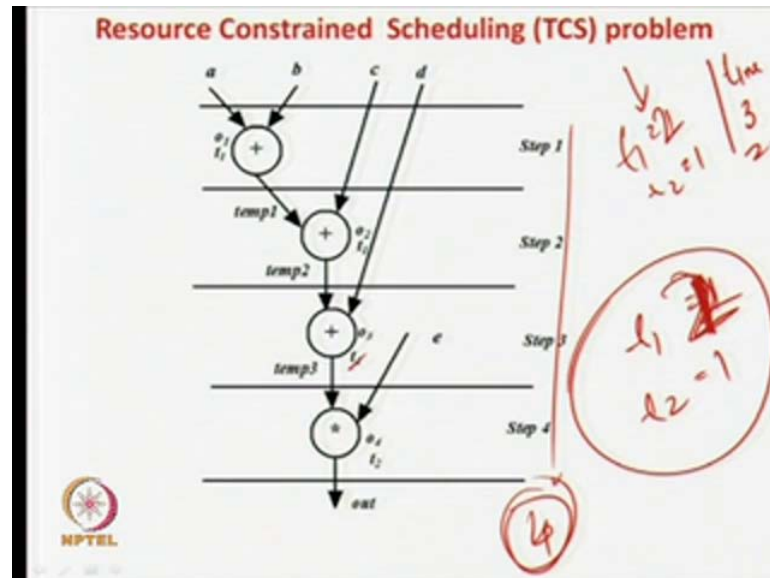
Therefore, the number of control steps is 4. Due to meeting the resource constraint, we cannot have a schedule in 3 steps

NPTEL

Now, resource constrained time scheduling problem that is what I have discussed. So, in this case the number of control step is three and because in the last example it was in

time step three because of because of the time constrained. So, what I have discussed is written in the slide, so I mean because the time resource constrained, so you have increased your time step.

(Refer Slide Time: 24:00)



So, now if you say that I want to have a resource constrained of one and one and till I want to get a I mean schedule in three type step, so this is the infeasible requirements. So, you have to either increase your time steps or you have to increase your hardware requirements. So, that is the is the... I mean next definition that is called time resource constrained schedule. So, in this case it was only about the time only about the resource constrained. So, he is saying that or the or the designer is saying that the T equal to 1 and T equal to 2 that is maximum 1 adder and maximum 1 multiplier, so you require four steps.

So, it is one more than the case when your requirement when the constrained constrained may be two that is 2 adders kind of a thing, so we have increased from 3 to 4. Now, in what is the next problem, so scheduling problem we have seen that is both resource as well as time both may be constrained. So, if somebody says that T 1 equal to 1 and T 2 equal to 1 and time is equal to time is equal to 3. So, you can say that it is impossible to generate or is a infeasible situation, to generate a schedule in this way. So, you may have to either say that either increase my time resource requirements to say 2 and then you can

follow this or we have to say that even if i want to keep my resource constrained less or or see here that is 1 1 1. So, you have to go for a time constrained of 4.

(Refer Slide Time: 25:09)

Time Resource Constrained Scheduling (TCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$, a time constraint (deadline) D , resource constraints $\max_{k, 1 \leq k \leq |T|}$ for each functional module of type $t_k, 1 \leq k \leq |T|$ and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T , meeting the deadline D , meeting the resource constraints for each functional module type and obeying the partial order.

In time resource constraint scheduling, we need to meet both timing and resource constraints.

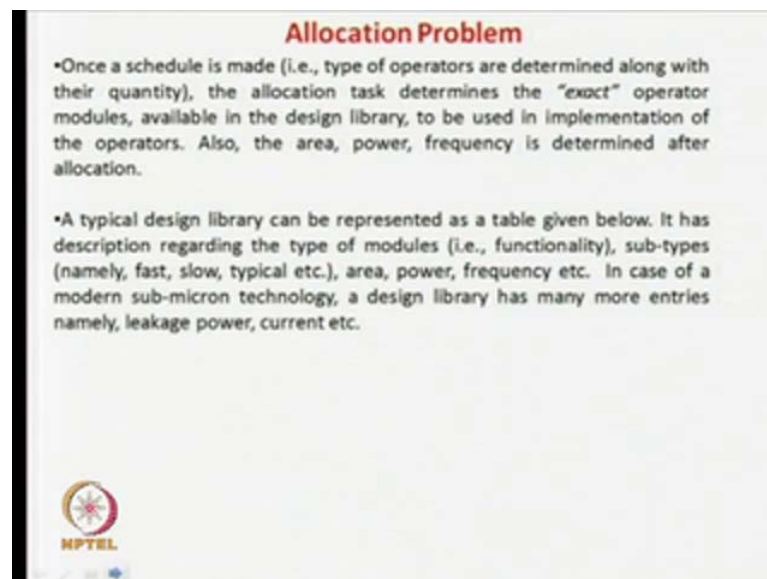
So, this is how sometimes we need to inconsistency schedules and then you have to go for a better schedule i mean we have to I mean restrict one of the we may have to relax one of the requirements and then we can give you a successful schedule. So, what we are learning over now, so what we are seeing is that we are looking at the schedule problems we are not giving you the solution that given a constrained how we get this graph.

Given 1 1 and three how do you get this graph or given 1 1 and two we said this is not possible. So, how automatically we can get it that we are not discussing that we will be discussing in the following lectures. In this lecture what we are doing we are just defining you the problem what are the problems that algorithms will be looking in the few I mean next in the next lecture onwards.

So, now you do not do not do not think about how you can automatically do that just understand try to understand the problem. So, in time resource time resource constrained scheduling problem is o set will be a T set will be there there is set of operation different type of elements this function is there which maps the operation to the type of hardware; then there is a resource constrained as well as a deadline constrained. So, we are truly increasing the number of constrained as well as there is also a what do you call the data dependency, that has to be also that the partial order you have to always follow.


Then you have to go for a we have proper schedule and then you have to say that the proper schedule matches both this, what do you call this resource constrained as well as the time constrained. Obviously, you have to follow the partial order of data dependency for everything there is a feasible schedule, that if one of the say that one of the parameter is not being met, like for meeting the time schedule you are not able to meet the resource constrained. And then for missing the resource constrained you are not able to meet the time constrained, then you said infeasible then you have to relax any one of them.

(Refer Slide Time: 26:43)



Allocation Problem

- Once a schedule is made (i.e., type of operators are determined along with their quantity), the allocation task determines the "exact" operator modules, available in the design library, to be used in implementation of the operators. Also, the area, power, frequency is determined after allocation.
- A typical design library can be represented as a table given below. It has description regarding the type of modules (i.e., functionality), sub-types (namely, fast, slow, typical etc.), area, power, frequency etc. In case of a modern sub-micron technology, a design library has many more entries namely, leakage power, current etc.



So, that was about the scheduling problem, now assume that this scheduling have been done, so that is all the operation have even given some time step. Now, what you have to do, so now, you have to find out exactly now in this case if you remember we are saying that we require a adder this is T 1 we require a add here, we require a multiplier. So, this is actually we had just said as a name that you were using a adder, you were using a multiplier and you were using a what do you call, so what do you call call called a subtractor.

But, now we are not telling exact nature of the multiplier exact nature of these adder or something like that you know the lot of adders are possible and then it available in the library like ripple carry, adder carry loop ahead adder carry save ahead adder. So, some adders are faster, some adders are slower, faster adders are generally high in area they

take more power, we all know that slower adders are generally less area they take less power and so forth.

So, in case in this type of allocation procedure we have to look at the design library and then you have to typically find out or exactly find out, which is the type of adder or which exact hardware module here going to be implement for each of the operations. Like in this case you may see that there are two adders like t one and these are two different additional operation.


So, it may happen that you may take it to be a carry loop ahead adder and it may be a ripple carry adder sorry in this case they both are reused we should not take this example like in this case you may think about this example. So, you may think that these are the two different adders they are working in parallels you have to put two adders. So, you can think that if I will be taking as a ripple carry adder and this I will be taking as a carry loop ahead adder this also possible they are both adders, but you can take you can they can be allocated a different, what you can call different hardware actual hardware element.

(Refer Slide Time: 28:30)

Allocation Problem

Sl. No	Name of Module	Type	Sub-type	Frequency	Area	Power
1	Adder-slow ✓	t_1	$t_1 - S$	F_{1-S}	A_{1-S}	P_{1-S}
2	Adder-fast ✓	t_1	$t_1 - F$	F_{1-F}	A_{1-F}	P_{1-F}
3	Multiplier-slow	t_2	$t_2 - S$	F_{2-S}	A_{2-S}	P_{2-S}
4	Multiplier-fast	t_2	$t_2 - F$	F_{2-F}	A_{2-F}	P_{2-F}

It may be noted that a fast module has higher frequency, higher area and higher power compared to its slower counterpart; so $F_{1-S} < F_{1-F}$, $A_{1-S} < A_{1-F}$, $P_{1-S} < P_{1-F}$ and $F_{2-S} < F_{2-F}$, $A_{2-S} < A_{2-F}$, $P_{2-S} < P_{2-F}$.



So, allocation problems will take you take in the schedules which you already made. So, what you do you take the exact hardware that is required and you put it. And like like this today the design library will tell you what is the power requirement? What is the

area requirement? What is the frequency? Everything will be mentioned there, so it will be very easy for you to select this one here, what type of adder or multipliers you require.

So, let us take a typical example, so let let be this be table be your design library. So, what we have, we have two adders one is a slow adder, one is a fast adder, I am not typically mentioning it is what is the type like people carry or carry say we are not saying that. We are saying that there is a slow adder, there is a fast adder the type is t_1 , then subtype we have to also mention the subtype. So, t_1 , S is slow t_1 , F is fast then we can say the frequency is F of t_1 S and F of t_1 fast, so this is the frequency of the fast adder and slow adder respectively.

Similar this is the area, this is the power then and then we have two types of multipliers also t type is t_2 subtype is t_2 slow t_2 fast and similarly here we have defined our frequency area and this one. So, it is a general rule of thumb is that generally if the elements are faster, so frequency will be higher area will be higher and power will be higher, so generally this relation will hold.

So, what what are you mean what I am say that, so these are slow adder. So, frequency will be less sorry. So, fast adder, so this frequency will be higher this frequency will be low. So, area of this fast adder will be also higher than the area of the slow adder similarly power of the slow adder will be also less than the power taken by the fast adder. So, everywhere this one will be higher than this one, similarly in the case of multiplier. So, we see that always the slow frequency of the slow adder will be less than the frequency of the fast adder; and similarly the area of the fast adder will be higher, area of frequency power of the fast adder will be also higher and similar case of multiplier.

Generally the hardware which is faster I have more frequency, so it is obvious it is generally it will take more area and more power kind of a thing, so there is general rule of thumb. Now, we have to now let us take this unconstrained problem.


(Refer Slide Time: 29:57)

Allocation Problem

Let us consider the unconstrained schedule of the expression $(a+b+c+d)*e$. From the output of scheduling we know that o_1, o_2, o_3 are of type t_1 and o_4 is of t_2 . Further, we need two modules of type t_1 and one module of type t_2 .

Now, depending on requirement of frequency and available area-power overheads we can select the sub-types for t_1 and t_2 . If we have high area and power constraints then we would use $t_1 - S$ for t_1 and $t_2 - S$ for t_2 .

It may be noted that time period of each control step is dependent on module having the lowest frequency because system clock frequency depends on the critical path. In general, a multiplier has much higher area and power requirements compared to an adder. Also, frequency of a multiplier is lower compared to an adder.


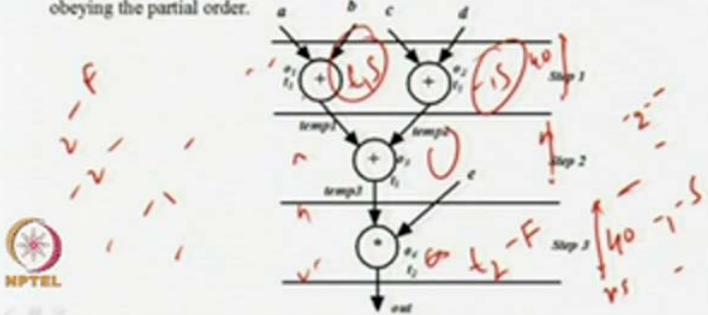


(Refer Slide Time: 30:12)

Unconstrained Scheduling (UCS) problem

Given:
A set of operations O , a set T of different types of functional modules, a type function $Ty: O \rightarrow T$ and a partial order on O determined by the precedence constraints.

Find:
A feasible schedule for all elements in O , taking appropriate modules from T and obeying the partial order.



Now what we have to do let us take this this one, so this guy and then we have to say that which is the exact adder now it was $t_1 t_2 t_3 t_4$ sorry t_1 and t_2 kind now you have to say that which I will use you can have you have to say $t_1 f$ or t_1 slow and or $t_2 f$ and t_2 slow. So, now we will we will explain that later on with the slides, but now let us just see how we can do that. So, generally what happens multipliers are very slow circuits, is slow circuits because very complex multiplier or complicated operation compare to adders.

So, it is even happens that your that is the frequency of t_2 faster will be that is the frequency of the fast multiplier will be also less than frequency of t_1 slope that is the very typical thing you have to understand again and in this case say t_1 s. So, the the frequency of t_1 s that is the frequency of the slow adder will be still higher in the frequency of the fast multiplier that reemphasize multipliers are very complex circuits compared to adders. So, even if you take a very slow adder is the very interesting thing to observe that even fast multipliers till the frequency or the operating speed of your slow adder will be higher than that. So, that is the general rule of thumb will take and then we will try to go for the allocation procedure.

So, we know that this is the, these are time steps and; that means, it it takes some frequency or this time steps has some time units that; that means, what, so if you say that this is, so 25 Nano seconds. So, what does it mean; that means, that your adjust to additions they are operating in parallel this should be over much before 25 Nano seconds. So, if you take it as net 10 Nano seconds then you may need a problem because 10 you said that step one or step two or step three these are the steps are of equivalent size.

We are assuming typically for the sake of simplicity for complex examples of complex cases the time steps may also vary in size. So, sometimes or some case you have 10 sometimes in 20 the uneven time steps, but there are for complex and multiple clock type design. So, we are not going into those complexity in this course, because our idea in the course is to take what is of the three design was, four design flow greater than going in even depth in one of the areas.

So, let us assume that all the time steps have equal length I mean equal time period. So, if you say that it is a 10 Nano second and say if the addition here takes 20 Nano seconds. So, you will have problem because even before the addition has been done you will come to the next time step. So, the clock will arrive, so it will be a problem. So, whenever we say that it is a 25 Nano seconds something like that, then it should be able it should should assure that these two operations are done these two operations are done and this this operation is also done.

So, who will decide what are the time step length? So, it is very obvious the slowest guy in the whole schedule process will determine that, so if you say that this addition

multipliers are very complex that take lot of time. So, you can say that it takes 40 Nano seconds to do your solve the problem. So, obviously everything will be 40 Nano seconds by default because this here you have to use 40 Nano seconds because your multiplier is doing the solving or solving your problem in or doing the multiplication for the 40 Nano seconds. So, obviously if you put twenty third an adders are doing your job in 25 Nano seconds. So, instead of 45 to 25 Nano seconds, so this guys will not have a problem, this guys will not have a problem in generating the output, but if I instead of 40, if I put 25 Nano seconds, then the adder multiplier will not be able to do its operation and will be the inconsistency position.

So, what we do we always say that the fastest, slowest guy will determine the length of the time step, so in this case let us see the 40 Nano seconds and so it everything will be 40 Nano seconds and this will be also 40 Nano seconds. So, this may be even 25 or something like this. So, in our case what we have seen, so so our t_1 slow. So, we have already seen that the frequency of t_2 fast that is is also less than frequency of t_1 slow. What this happens because you can assume that this is the fast adder we just assume that is the fast adder this is slower slow slow adders.

So, still the frequency of your frequency of your slow adders will be higher than your fast multiplier. So, in other words the time period or time required to solve the problem will be lower in case of the slow adders compare to the time period required for even the fast adders. So, generally what happens multipliers are always very slow if you if take a fast multiplier it will be slower than the even the fastest adder. So, let us assume that we take everything very fast, so let us take everything very fast. So, we what we will do you take t_1 fast, here also you take t_1 fast here, also you take t_1 fast and here also you take t_2 fast, because your job may be there I want to do everything very, very fast, very, very quickly.

So, let us assume that it will this t_2 fast will take say 50 sorry 40 Nano seconds to do the operation and sorry. So, t_2 fast is going to take 40 Nano seconds to do the operation, so this one is actually 40 Nano seconds. Now, what you say that you say that we are using fast adders over here. So, instead of 25 we also get for the slow adder kind of data, so assume that they do your job in 20 Nano seconds.

So, now there is a big loss why you were in a big loss because this even if I am using the fastest multiplier still what happens the time period taken is 40 Nano seconds. And if I am putting a very fast adder also my time taken is 20 Nano seconds, but I cannot make this one as 20 Nano seconds, because there may be the problem. So, by default this is for t , so this will be 40 as well as this will be 40.

So, even by using a slow adder by using a fast adder I am not getting any gain, I am getting the result in 20 Nano seconds because fast adder, but still I have to wait for 40 Nano seconds because the everything is decided by the slowest guy in the in the all the operations, so that is t_2 or operation O_4 . So, that is actually slowest one it is taking 40 Nano seconds even with the fastest guy I am having for type two is 40 Nano seconds to all the time steps are 40 Nano seconds.

So, even, so there is no use of putting a very fast adder over here you know the fast adder area will be higher than a slow adder and time period may be all the case and even the area and power of the slow adder will be less than the first adder. So, when you putting very fast adder, so I am putting more area and more power and I am computing of the total Nano seconds, but still I have to wait for another 40 Nano seconds because the time period is 40 Nano seconds and it is determined by the fastest slowest element of the block which is at the multiplier.

So, I am putting the fastest multiplier available if it is 40 I can I cannot have a compute multiplication more in less than 40 Nano seconds. So, I cannot bring down the time step less than 40 Nano seconds. So, there is no use of putting anything very fast adder over n , because even if I am putting a fast adder I am getting the solution in 20 second still I have to wait for certain Nano seconds.

So, let me try the other way, so let me what I do is that this is 40 to fast I am using what I have to do, but now I am putting say t_1 slow and also t_1 slow and here also t_1 slow. So, I will get the answer in 25 Nano second here also I will get the answer in 25 Nano second. But still I have to wait for another 40 Nano second, so even I if I get the result in 20 Nano second. So, 25 Nano second it does not make any difference for the adders because time time step is 40, so you have to wait.

So, in this case because this 40 Nano second is these are bind by the fastest multiplier I have I cannot go down between this. So, why I should use a 20 Nano second fast adder

because fast adder will always take more area in slow adder and if I am putting a slow adder I am not losing anything. Because compute the addition in 20 Nano seconds or even if I do it in 10 Nano seconds, but I cannot get the answer earlier, because I have to have my time step of 40 Nano seconds because of the multiplier. So, what I will do is that typically intelligent designer will put a 40 Nano seconds multiplier is the fastest is available you will put it here and here you will not put a very fast adder, you will put a very slow adder. Because who is getting the answer in 25 Nano seconds and still I mean it is much less than 40 Nano second.

So, anything for below 40 Nano second is even if I have a very, very slow adder which will do my answer in say 30 Nano seconds, I will be prone to put that because I want to do my I do not want to I cannot the time steps are fixed to 40 and I do I can I will be then within I one my motivation will be that I want to compute all my answers. Or addition answers in less than 40 Nano seconds at least equal to 40 Nano seconds and I will be using as slow or as little hardware as possible. So, that is you will take a slow adder because slow adder will take very less area and it will take very less power.

So, what I will do I will use slow adders over here and I will use fast multiplier over here, so that is actually called the binding allocation problem. So, what I am doing. So, what I am understanding. So, what I am doing I will I am finding out the this is the slowest guy. So, I will multiply the slowest part, so even if I do my best effort. So, even if I using two s 2 type 2 fast still this is actually say 40 Nano seconds kind of a thing I cannot bring it below that. Then I will try to take the stuff or, so I will take such type of adder which will do my answer get me my answer in less than 40 Nano seconds and take the minimum amount of power and area. So, I will obviously, use t 1 slow and t 1 slow and t 1 slow over here, because they are going to give me the answer in say 21 Nano seconds still much less than 40 Nano seconds and my area and power is less than using a fast adder.


(Refer Slide Time: 38:44)

Allocation Problem

Let us consider the unconstrained schedule of the expression $(a+b+c+d)*e$. From the output of scheduling we know that o_1, o_2, o_3 are of type t_1 and o_4 is of type t_2 . Further, we need two modules of type t_1 and one module of type t_2 .

Now, depending on requirement of frequency and available area-power overheads we can select the sub-types for t_1 and t_2 . If we have high area and power constraints then we would use $t_1 - S$ for t_1 and $t_2 - S$ for t_2 .

It may be noted that time period of each control step is dependent on module having the lowest frequency because system clock frequency depends on the critical path. In general, a multiplier has much higher area and power requirements compared to an adder. Also, frequency of a multiplier is lower compared to an adder.



So, that one is that is what is the motivation over here, so it is written over this case.


(Refer Slide Time: 38:48)

Allocation Problem

So, in the example, time period of each control step be $\frac{1}{F_{t_1-s}}$.

Now, if have no area and power constraints, then we would use $t_1 - F$ for t_1 and $t_2 - F$ for t_2 . The time period of each control step is $\frac{1}{F_{t_2-f}}$.

But, in general $F_{t_2-f} < F_{t_1-s}$; frequency of a fast multiplier is generally less compared to even a slow adder. So in spite of allocating fast adders to o_1, o_2, o_3 (consuming high area and power), time period of control step is $\frac{1}{F_{t_2-f}}$, which is not dependent on F_{t_1-s} or F_{t_2-f} . So we can use slow adders without any compromise in overall time period of operation (i.e., time period of control step).



So, this is what I have written, so in this these are actually I have given you some examples of frequency and time periods what I was saying 25 seconds at the nine periods 49 seconds the time periods. So, here in the slide is written in a general formula like here you saying that a time period which control step is $F t$ to slow, so I mean just you can go through this. So, I mean if it is saying that the time period of which control step is t_1 by $t_2 F$ that is the frequency of the fastest multiplier. So, but now now you can think that

if I am taking a very slow multiplier then you will be in the another very big problem. Because now the time period of the slow multiplier may be say 50 Nano seconds then the time periods of each of the times will be 50 Nano seconds now.

So, I will not be prone to take a slow multiplier I will be taking a fastest multiplier and a slow adder because the multiplier is actually determining all the type, because it is the slowest element in the whole schedule operation. So, if have another block which is doing a very, very complex operation say which is mod computation sorry factorial computation and all. So, then I have to concentrate then I will take the faster mod computer and that will actually determine my time steps. And based on the requirement I will take such elements which will be the slowest in nature, but still do not over shoot the time step requirement. Like in my case of this multiplier, the multiplier of the slowest element, so I have taken a very fast multiplier which is 40 Nano second. Then I will take adders slow as slow as adders as possible or as slow other elements as possible that do not over share the time requirements of 40 Nano seconds.

(Refer Slide Time: 40:38)


Binding

After all the operations are scheduled and allocation is done, we get information regarding exact type of circuit modules (from the design library) to be used and their numbers.

We have seen in the allocation step, that operations in a control step are performed by different modules, however, modules are shared between operations (of same type) that are in different control steps. In the unconstrained schedule example, an adder module will be shared between o_1 and o_3 or o_2 and o_3 . Due to sharing, in addition to operational modules (adders, multipliers etc.), we need multiplexers.

Further, to store variables (a, b, c, d, e) and intermediate results ($temp1, temp2, temp3$) we need registers. Like operational modules, registers can be shared, which do not lie in same control step.

All the above-mentioned steps (after scheduling and allocation) fall under Binding.

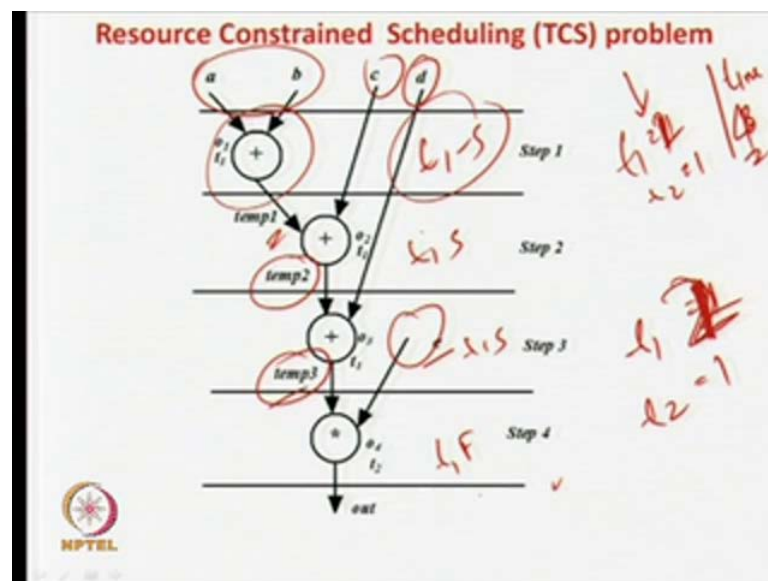


So, it is saying that nobody we have to find out the slowest element of the whole lot in in that is in set O and that is actually set t or other. So, you have to find out the slowest element in set t that is the type of operators and then you have to take the fastest version available for that then determines the time time step length. And accordingly you have to use very slow elements of the other guise still you have to think that you should not over

show the time step that one. So and that is what has been written in the slide I mean you can read through that.

So, then then actually this is now you are coming to the third step. So, third step is the I mean a bit complex step I should not call it complex, but what happens here. So, in this case what we do in this case what we do is now we have your scheduled operations, now we have binded elements, binded elements means now all the scheduled operations has been binded sorry allocated to some specific hardware like hardware fast adder, slow multiplier etcetera, etcetera kind of a thing. So, now what you have to do, so in case of binding what happens. So, now, you have to say that like for example, you might have say that we are using like, if you take this time control step like if you take this example that is better to explain here.

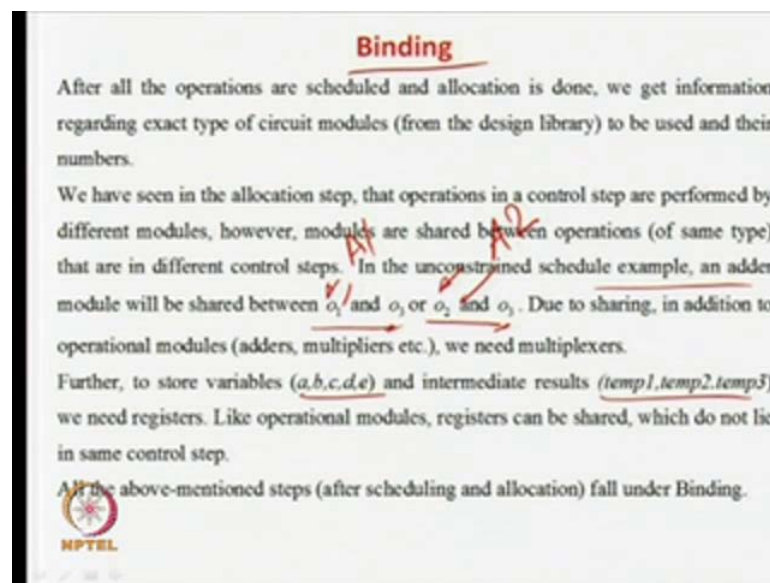
(Refer Slide Time: 41:15)



So, in this case you see that we have used only 1 adder and 1 multiplier. So, you can think that this one will be t_1 slow, this one is also t_1 slow, this one is t_1 slow and this one is t_1 fast. Now, this is your actually scheduled and this one is actually binded everything is done sorry allocating the scheduling is this one and these are the allocations now you have to bind it; that means, what we have only one adder. So, there is only one adder which is of type t_1 s and you have to first you have to say that a and b will be first done with this one then again this temporary variable 1, $temp_1$ and $temp_c$ will be again done by the same adder will reused of the adder.

Then you can see that again temp 2 and t will be again I mean binded in same adder for the third time which is being reused and then you can say that temp 3 and e has been allocated and and schedule allocated and binded. So, you can say that temp 3 and e has been binded then this adder number 4 that is t 1 f. So, in other words what I am saying is that in case of binding, what I am doing as we were reusing the elements, as we are shown in the examples.

(Refer Slide Time: 42:17)

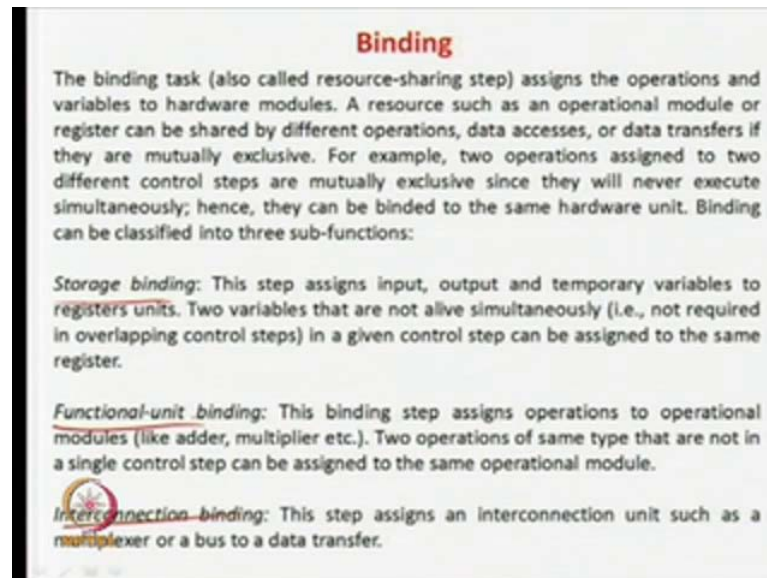


As we were clearly, always reusing the elements like adder multipliers you are reusing many times we do not have to use do not use parallel is not even if you are using parallel is not like even we are using two adders in state one still one of the adder will be reused in the third step as you have already seen like we actually called the reusing. So; that means, what; that means, same variables or same operations, which is binded to same same hardware like first it will be a plus b, then it will be c plus b and so forth. So, actually binding thus binding actually the problem is that one, it says that which hardware elements binds to which of the operations, so that is the idea. So, here it says that in the unconstrained time schedule first example O 1 and O 3, O 2 and O 3.

So, there is actually two hardware's are available, so in one actually we do one the other we do two and then in the second time step we do O 3. So, we can take saying it is say adder one you do O 1 and O 2 or you can say that adder two I do O 2 and O 3. So, O 3 can be buying the two either a 1 or a 2 they are the two adders. So, that is actually then

also you have to think about the variables like a, b, c, d, e where the input variables and they are the temporary variables, so they has also to be binded to some kind of registers. So, that is particular operations or particular variables then they are actually buying there two on which hardware or on which allocated hardware, they have to do the operation is actually called the binding. So, we will explain with examples then it will be more clear.

(Refer Slide Time: 43:40)



Then actually we have three type of bindings let us just define the type storage binding. So, storage binding means if it is input variables a, b, c, d, e, f temp 1, temp 2, temp 3. So, which are the registers which they will be used to store, so there is actually called the storage binding. Functional unit binding is very simple like operation O 1, which adder which has been allocated, which allocated adder they will be operated on or which the which multiplier allocated multiplier will be multiplying a particular operation, I will actually called the functional unit binding. That is which hardware will do which operation and similarly interconnect binding is the actually using a multiplexers and wires.

So, that is about actually three types of binding, so that we will all illustrate by an example which were already using. So, actually already said that there the three different binding steps, which are like this one storage storage function unit and interconnection binding. But there are three different steps although written in in all text, but actually they are all intertwined and they are done hand in hand.

(Refer Slide Time: 44:14)

Binding

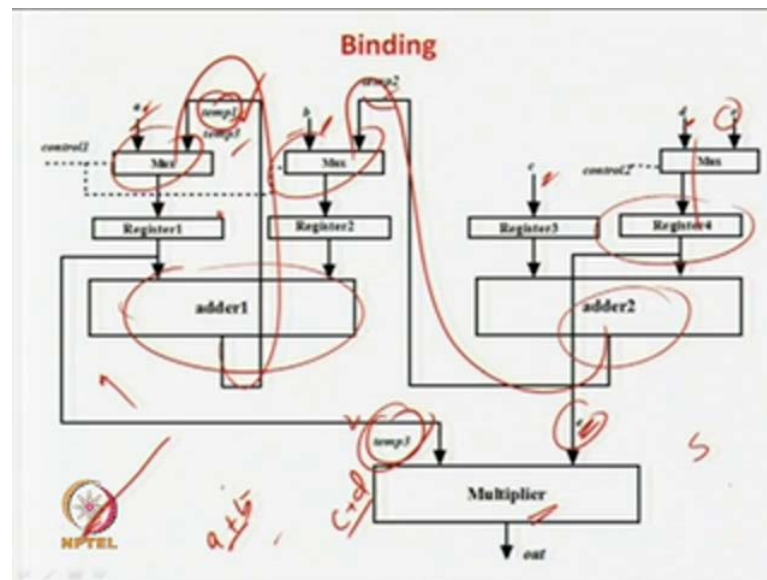
Although listed separately, the three sub-functions are intertwined and are to be carried out concurrently for optimal results.

Now, we will illustrate Binding for the unconstrained schedule when allocation is-- two number of modules of type $t_1 - S$ for o_1, o_2 , and one module of type $t_2 - F$ for o_4 .

So, now we illustrate the binding operation for the unconstrained schedule, when the allocation is two types of modules that is slow adder and fast multiplier. Already we have considered this and O 1 and O 2 O O 1 o two and sorry O 3 they are your this is O 3 this one is your addition operation and O 4 is the multiplication operation. And we are taking the case where it is unconstrained schedule that is this is your two adders that is O 1 O 2, then you add temp 1, temp 2 that is O 3 and finally, in the last step we have this multiplier operation which is actually O 4. This is your unconstrained operation, so two adders are in the first step, so you got two adders same adder can be reused. And obviously, there is one multiplier block, so that has to be done to this one about the multiplication O 4.

So, let us look at this is the basic architecture of the binding element. So, what we have we have one register one adder as you can see over here this is another adder 2 as you can see over here then a is one input, then b is one input, c is one input and d is one input. So, now, we have to say that a plus b that is one operation and c plus d is another operation. So, we allocate a plus b to other one sorry we bind a plus b to adder 1 and c plus d to adder 2. So, what will happen a has to be go to adder 1, b has to go to adder 2 in one step, simultaneously c has to go to adder 2 and d has to go to adder 2 that is what is being done.

(Refer Slide Time: 45:19)



Some then some results will be generated which is nothing but your temp 1. So, temp 1 and plus temp 2 we are saying that we will again bind it to adder 1. So, this temp 1 from adder 1 will be fed back to this same adder and the output of temp 2 that is of output of adder 2 will be again also fed back to your adder 1, because you are binded temp 1 plus temp 2 to adder 1. Now, what happens the output is temp 3. So, it is generated and this e that is actually temp 3 plus e is actually a plus b, temp 3 is a plus b plus c plus d that is actually actually multiplied with e. So, temp 3 star e is actually binded to multiplier that is obvious because there is only one multiplication operation and one block is there.

Now, what we will see that there are different operations, I mean now we will see how this happens that is in first step you add a plus b then you add c plus d simultaneously then you go for temp 1 plus temp 2 and then actually you go for temp 3 star temp 3. So, there is some step time step involve, so let us look at how the hardware works like that and how actually the binding is happening.


(Refer Slide Time: 46:48)

Binding

At control step1, we have 4 active variables (a,b,c,d) , at step2 we have 2 active variables $(temp1,temp2)$ and at step3 we have 2 active variables $(temp3,e)$.

So we have a maximum of 4 active variables at step1, thereby leading to the fact that we required 4 registers; a,b,c,d cannot share any register. However, registers can be shared between (a,b,c,d) and $(temp3,e)$; (a,b,c,d) and $(temp1,temp2)$; $(temp1,temp2)$ and $(temp3,e)$. However, variables listed in the brackets cannot share registers among themselves. As discussed in last section, we have two adder modules and one multiplier module. Based on these facts a possible binding is as follows

$(a+b+c+d) * e$

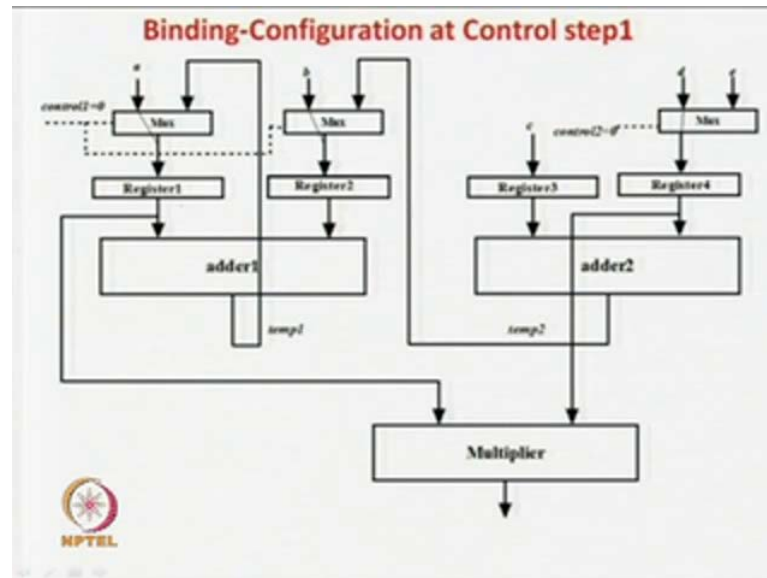


So, this was a very basic block level diagram, so now, we will see step by step. So, in control step one what happens you actually have four active variables a, b, c, d because in the first time step we are adding a plus b and c plus d. In time step two we have two active variables temp 1 plus temp 2 because we are adding in time step two temp 1 plus temp 2.

In time step three what happens you have two active variables temp 3 and e because temp three is a plus b plus c plus d and if you remember example we are taking is a plus b plus c plus d and we are actually starting with with this is our expression which we are doing. So, now you see, so we have to store this variable. So, there is we have seen that there are three types of I mean what you call this variable three type of bindings are there. So, storage element binding let us say, so we have four elements a, b, c, d, so we have to give some registers where there can be stored.

Now, you can see in time step one a a, b, c, d all of them are acting. So, here actually you cannot save any kind of this thing. So, in time step one you cannot I mean share any kind of a hardware like for a plus b plus c plus d all the a, b, c, d are all alive at time step one. So, as all of them are in time alive in time step one, so obviously you require four registers for this one.

(Refer Slide Time: 47:56)



So, 1, 2, 3, 4, so four registers are mandatory because in time step one, I mean all the four i mean what you call this four variables are simultaneously to be stored.

(Refer Slide Time: 48:05)

Binding

At control step1, we have 4 active variables (a, b, c, d), at step2 we have 2 active variables ($temp1, temp2$) and at step3 we have 2 active variables ($temp3, e$).

So we have a maximum of 4 active variables at step1, thereby leading to the fact that we required 4 registers; a, b, c, d cannot share any register. However, registers can be shared between (a, b, c, d) and ($temp3, e$); (a, b, c, d) and ($temp1, temp2$); ($temp1, temp2$) and ($temp3, e$). However, variables listed in the brackets cannot share registers among themselves. As discussed in last section, we have two adder modules and one multiplier module. Based on these facts a possible binding is as follows

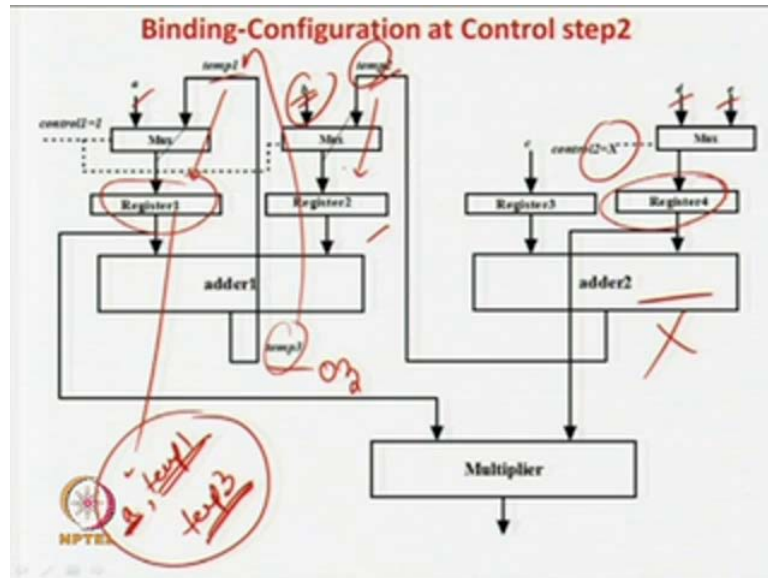
(a, b, c, d) are
Temp1 Temp2

The NPTEL logo is visible in the bottom left corner.

So, four registers has to be assigned and that is actually you have to allocate four registers and a is binded to register 1, you can say b is binded to register 2, c is binded to register 3 and d is binded to register 4 kind of thing. See see a then b then c and then d. So, all of them are binded to 1, 2, 3 and 4. But now you see in the second life cycle second life stage after a plus b plus c plus d happens. So, this will generate temp one and

this will will generate temp 2. So, now, you can see the temp 1 and temp 2 do not share any kind of a common life time with a, b, c, d. So, whatever for registers we have taken or we have allocated four registers and binded with a, b, c, d they can be reuse to store temp 1 and temp 2, so that actually we can do over here.

(Refer Slide Time: 48:48)



So, you see a is actually in time step one what happens time step one this is say this is actually at control time step one. So, I mean this multiplexer actually sends a over here b over here then c is I mean allocated directly to sorry binded directly to register three and d is going to register four. So, we are using two multiplexes because some as I have already told you adder one will do a plus b and then it will be do temp 1 plus temperature 2.

So, there is a multiplexer over here, so in time step one control step one you made the multiplexer control to be zero you can see that we have made control this one equal to 0. So, what happens, so these guy is flowing over here this already shown in the slide. So, this guy is flowing over here, d is flowing over here, c is flowing over here and d is flowing over here, so you get temp 1 and temp 2 as the output.

Now, what about the control time step two in control time step two if you see we are making actually control as equal to 1. So, in this case what happen your temp one should go here temp 2 should go here and you are generating temp 3 in this case we do not require because in time step two we are not at all requiring this adder. So, now you see

what happens, so this register is actually being shared by a in the first time step temp 1 in second time step register 2, is shared in three, I mean register 3 is only stored in c not to bother.

And we will see that register four is storing d in the first time step. So, it is binded d in the first time step, but in the third time step when you are going for the multiplication operation, then what we have we have already done temp three plus temp sorry temp two plus temp three plus temp two already we have added here and we have got the result in this one temp three and then actually we have to multiply with with e.

So, now this will come via this register, so this will be come via this register register number 4. So, register number 4 was storing at d in the first step and in the third step it is actually storing the value of e, which show you that it can be multiplied with temp 3. So, register four is actually storing the value of d as well as t in two different time step. So, in other words, so we are using four registers here that is what is been told in this slide.

So, we are using four registers if you can see we are storing four register r 1, r 2, r 3 and r 4 and in the binding a and temp 1 is binded to register register 1 b and temp 2 is store binded to register r 2, c is binded to register 3 and d and e are binded to register r 4. So, that is actually your what you say what do I say these actually storage binding kind of a thing.

Now, you have some addition operations, so already told that O 1 is done over here O 2 is done over here this a plus b and c plus d this is control time step two. Now, control time step is also it is in control time step one O 1 is binded here and O 2 is binded over here that is a plus b and c plus d. Now, in this case you see in time step two this is not doing anything this do not care and here what we are doing we are doing temp 1 plus temp 2 that is O 3.

So, O 3 is actually binded you can say to adder one and finally, the third step what we are doing we are actually multiplying temp 3 with e, so O 4 is actually binded to multiplier in the time step three. So, these actually is the story of binding in a nutshell that is we are given two adders and one multiplier and we require four registers. And so we have to bind a, b, c, d, e a b c d e they are the four input variable. So, you have to bind them to different registers.

Exact registers which will be storing the value at what time step and also the four operations we have O 1, O 2, O 3 and O 4 you have to bind them to adder 1, adder 2 and adder 4. So, this is basically the idea of binding we can explain with an example. Now, what is inter connect binding over here they are nothing but you are the multiplexers we have kept and the wirings. So, you see in register 1, where binding a and temp 3, so you require a multiplexer. So, in one go a will go and in second times steps step in the second time step temp 2 will go and in third time step temp 3 will go.

So, actually in fact, register one is been shared by three variables in this case you should see. So, in the first case I mean will be more precise here. So, in the fast time step these stored in register 2 and a is being stored in register 1 register one. So, in the second time step temp 1 is stored in register one because this is result of a 1 plus a 2 then again what you do again you add temp sorry here you are again adding what, you are adding temp 1 plus temp 2.

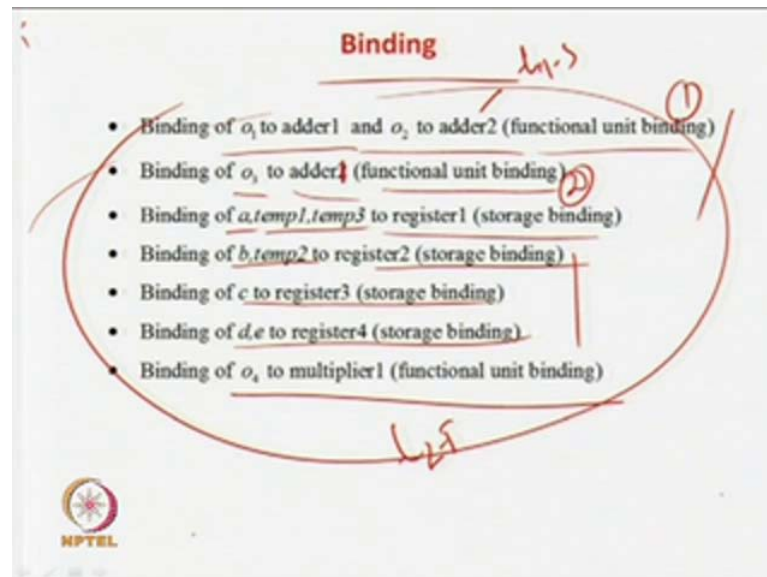
So, temp 3 will be generated then it will also be stored in register r 1. So, basically register r 1 is actually binding three variables a plus a then it is binding temp 1 and then finally, it is also binding temp 3. So, this is actually a for first time step a plus b that is temp 1 is second time step and the third time step I mean sorry second first is actually initial storage is a then first time step you are computing temp 1 that is a 1 plus a 2 that is also stored in register 1 and finally, in time step two you are calculating temp 3, which is also stored in register 1.

So, register one is actually I mean which can be shared by three variables, register 2 is shared by 2 variables b and temp 2. So, in first step and in the initial step you are reading the value of b. So, that is stored in register 2 and temp 2 is binded to register 2 at the end of fast time step, so here actually we have b and temp 2 and in register three only c is there.

So, only c is binded to register 3 and in case of register 4 d at the initial time step you read and e also is there at the initial time step, but you are requiring e in the third time step for doing the multiplication. So, in the register 4 is binded to e at the third time step. So, now, you require multiplexers, so why do you require multiplexers because in fast go registers 1 should add load a in the second time step it should add temp 1 in this third time step sorry second time step, which would add temp 2 and so forth.

In initial temp 1, then temp 2, then temp 3 all are actually rotate to register 1. So, obviously you require a multiplexer. So, there are some multiplexers over here we have to add and there we will actually comprise your, what do you say your interconnect binding, so at is in a nut shell what we have seen.

(Refer Slide Time: 54:25)



If you look at this this is your binding result. So, we say that at time step one o_1 is binded to adder 1 and o_2 is binded to adder 2 this is actually functional unit binding. o_3 of adder 1 functional unit binding will be done in adder 1 or adder 2 whatever you want to say that will be actually the second time step. Then we say that a temp 1 and temp 3 is binded to register 1, already we have told you b and temp 2 is done to register 2. So, this is actually storage binding, c is to register 3 and d and e are to register 4 by that therefore, these are storage binding and these are actually functional unit binding.

So, o_3 actually in this case we are doing in adder 1, so sorry for the actually adder 1 o_1 in adder 1 o_2 in adder 2, it is in the first time step. And in the second time step o_2 that is temp 1 plus temp 2 is again done in adder 1 and then actually o_4 is done to multiplier one that is earlier there was; obviously, one choice. So, this actually completes your binding arrangements. So, given I mean in this case adder all the adders are you should know that they were actually t_1 , t_1 slow and it was actually t_2 fast for the multiplier. So, this is what these are operations are physically they will be done by this this this hardware. So, if you define them they will actually tell your binding step.


(Refer Slide Time: 55:42)

Control Path

For the scheduling, allocation and binding considered in the running example we have the following signal sequences for *control1* and *control2* in the three time steps.

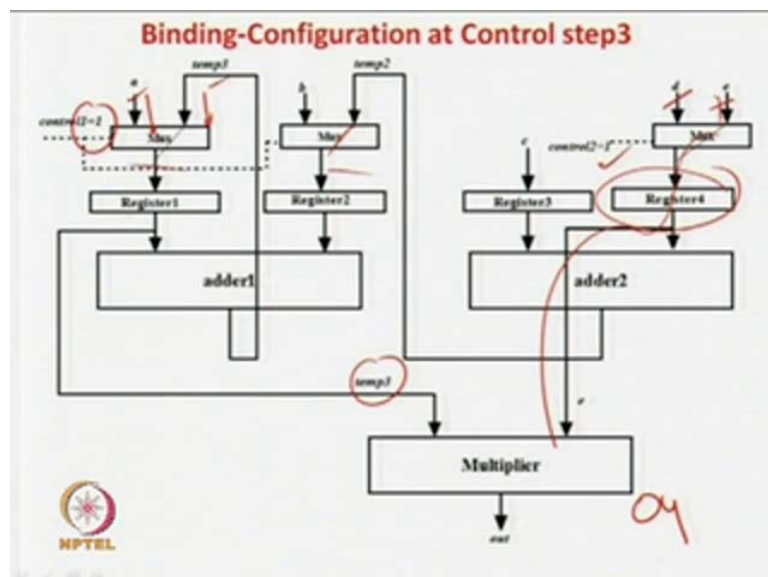
- Step-1: *control1* is 0 and *control2* is 0
- Step-2: *control1* is 1 and *control2* is X
- Step-3: *control1* is 1 and *control2* is 1

We need to develop a sequential circuit having two output bits "*control1*" and "*control2*" and they should have the values "00", "1X" and "11" in three consecutive clock edges. This circuit can be easily design using the concept of state machine implementation



So, these are this with this we actually complete the what you say the binding procedure of your high level synthesis process. Now, what you have done one small step is remaining that we will see.

(Refer Slide Time: 55:50)

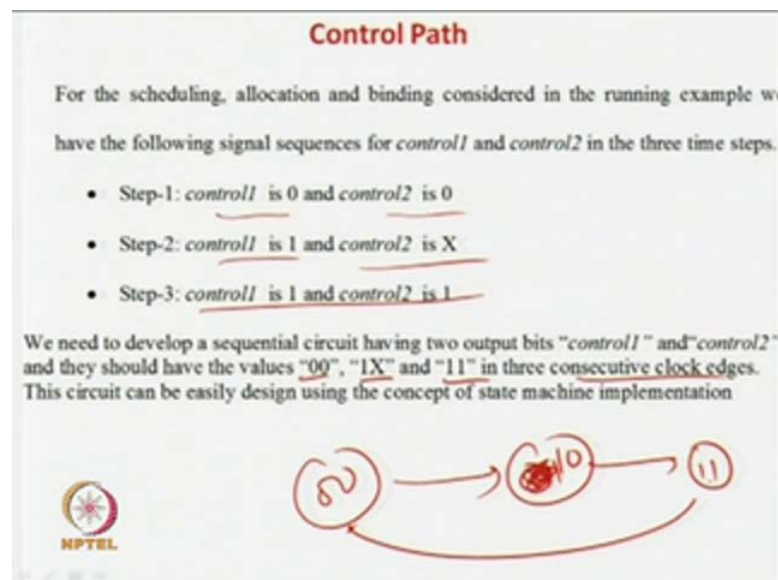


So, what we have seen that there are some multiplexers over here. So, in the first first step of the multiplexing, so what will happen we make control one equal to 0 so; that means, what a is flowing over here, b is flowing over here, c is there and; obviously, d is flowing over here. In second time step what we have done, so we require that temp one

should go over here temp two should go over here, so we made control one. So, that the multiplexer the second bit is selected.

So, in the step we made control one equal to 1 the other multiplexer and these adder is not require. So, do not care we do not require this and again in the third time step what we have done. So, again we have kept it one. So, the temp three generate kind of a thing and here actually we require control two to be one because e should flow over here. So, these are some of the control signals you have to generate.

(Refer Slide Time: 56:28)




Control Path

For the scheduling, allocation and binding considered in the running example we have the following signal sequences for *control1* and *control2* in the three time steps.

- Step-1: *control1* is 0 and *control2* is 0
- Step-2: *control1* is 1 and *control2* is X
- Step-3: *control1* is 1 and *control2* is 1

We need to develop a sequential circuit having two output bits "*control1*" and "*control2*" and they should have the values "00", "1X" and "11" in three consecutive clock edges. This circuit can be easily design using the concept of state machine implementation



The diagram shows a state machine with three states: 00, 10, and 11. Transitions are shown from 00 to 10, from 10 to 11, and from 11 back to 00. The NPTEL logo is visible in the bottom left corner.

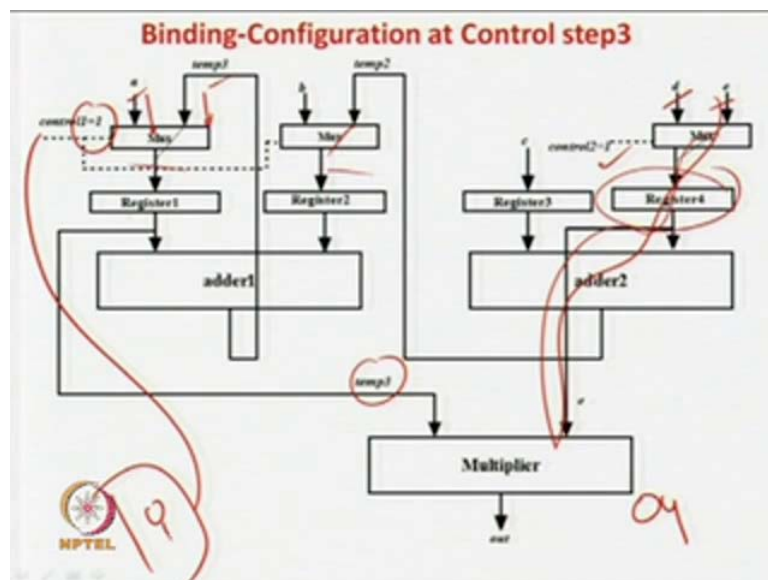
So, what happens here, so you require in step one control one is 0, control two is 0, in time step two control one is 1 control two is x. Because second adder is not required and in third step control one is 1 and control two is generated. So, you now have to generate a sequence with value 0 0 1 x and 1 1 in three consecutive clock cycles. So, that can be very easily generated by a final state machine like the output state 0 0, state 0 1 and state 1 1 you can say and these are moving back.

So, if you can add each clock pulse you will move around this final state machine. So, initially 0 0 and sorry it should be not 0 1 this is 1 0 you can take, because it is second bit is do not care over here and third bit is 1 1. So, if you go around this final state machine at each clock step, so you will be able to generate this control path. So, after doing this scheduling then we have done allocation and then we have done the binding that is you have allocated a, b, c, d to define registers then temp 1, temp 2, temp 3 to define

registers. And and and actually multiplexing arrangements we have done and also we have said that operation O 1, O 2, O 3 to different adders and O 4 to a multiplier.

So, after computing this what you can call this binding procedure then we have found out the control signals this should be 0 0 1 x and 1 1. So, that is can be very easily generated by this final state machine right. And so this can be very easily synthesized using digital circuit fundamentals and that will complete your whole RTL design design that is your high level block diagram. So, this is your high level block diagram finally.

(Refer Slide Time: 57:47)



(Refer Slide Time: 57:56)

Question and Answer

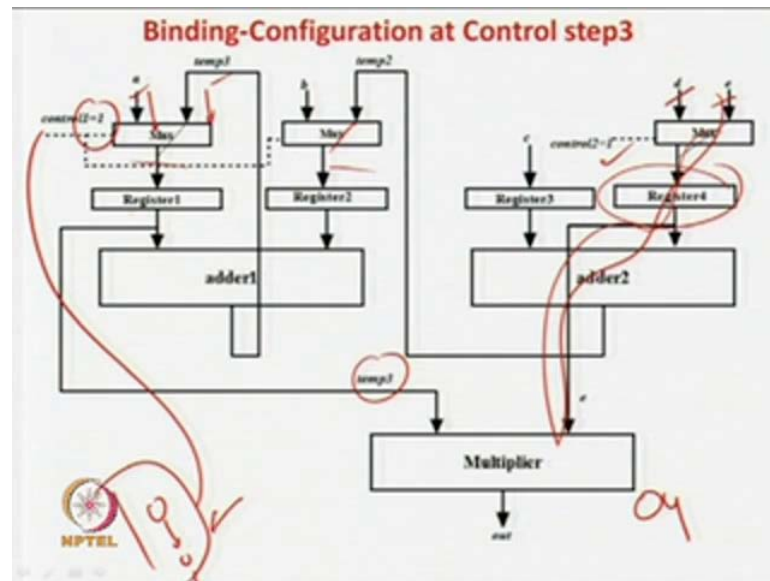
Question: Among the three sub-steps of HLS, scheduling, allocation and binding, what can be done without information regarding design-library?

Answer: Scheduling and Binding can be done without information regarding design-library. Scheduling assigns control steps to all operations in the CDFG, after satisfying data-dependency between the operations, subject constraints like number of steps, number of modules etc. So none of the parameters are related to design-library. In case of Binding, operations and variables are attached to circuit modules, which are selected from the design library during the allocation phase. As circuit modules are already selected from the design library during the allocation phase, binding can work without any information from the design library.

NPTEL

And this will be another block level diagram which will generate the control sequence. So, there we will have this final state machine which we have already drawn, so this is the output of the high level synthesis procedure. So, what we have started with, we have started with the example that is with a specification that is a plus b plus c plus d star e. And then finally, at the output we should get these architecture.

(Refer Slide Time: 58:06)



That is the block level architecture and then which is where the operations are schedule allocated and binding as well as this control structure.

(Refer Slide Time: 58:12)

Question and Answer

Question: Among the three sub-steps of HLS, scheduling, allocation and binding, what can be done without information regarding design-library?

Answer: Scheduling and Binding can be done without information regarding design-library. Scheduling assigns control steps to all operations in the CDFG, after satisfying data-dependency between the operations, subject constraints like number of steps, number of modules etc. So none of the parameters are related to design-library. In case of Binding, operations and variables are attached to circuit modules, which are selected from the design library during the allocation phase. As circuit modules are already selected from the design library during the allocation phase, binding can work without any information from the design library.

NPTEL

So, after that we go for the gate level synthesis which is the next step of the design flow. So, in the next lecture what we will be looking at, in the next lecture we will be mainly looking at how we can have automated algorithms or automatic procedures where given a specification like $a + b + c + d \cdot e$. And some design constrained like unconstrained design resource constrained, time constrained, automatically you will get a schedule allocated and binded design ready, so that we will be looking at in the in the next few lectures.

So, before we stop there is a question answer session. So, among all the three sub-steps of high level synthesis already seen scheduling allocation binding, what cannot be done, what can be done, without information regarding the design library. Like we are saying that I mean design library means whatever hardware is available to you, so that only those things that can be used with design like you cannot have a 10 input AND gate. You have a very good design library then you can have different type of adders, but if you have a very rudimentary design may be then you may have only carrying ripple carry ahead adder and so forth.

So, based on the design library which what can be done in the three steps what can be done without any information of the design library or which of them is a steps are independent of design library information. So, already we have decide seem that scheduling and binding they what they they will do they will actually in the scheduling we give different time steps to different operations. And in binding what we do we have different hardware, hardware modules where we where we physically bind which operation is to done and why.

So, this two are basically be general steps, but in allocation what we do, in allocation we actually look at the design library and then we see what are the time steps already we have seen that we have used a very fast fast multiplier that we have used slow adders. Based on the timing requirements of the multipliers and the adders and based on the time state we have already decided that it is very good to use a single, what you say, we say that it is good to use a fast adder slow slow sorry a slow adder and a fast multiplier.

So, this allocation stage looking at the frequency power requirements and all... So, allocation step it is very important to have a idea about what is in the design library. So, without having a idea about the design library you cannot go for the allocation step, but

the other two steps like scheduling and binding. So, I mean there is strictly speaking scheduling, you do not require any information about the design library it is a generalized step and your assigning operations to different time units.

But allocation it is mandatory to know that design library information is there based on that you see see that which exact hardware you are going to use and even if you do not say you by pass the allocation step, because you do not have in that information. So, even you can say that I use adder one I use adder two type adder three type multiplier type one, but exact details you do not know, because you do not have the library information still you can go about the binding step, because in this you have to physically bind the operations to different hardware.

But, without the by design library information allocation step cannot be done. So, with this we stop here and in the next lecture we are going to see automated procedure for going about high level synthesis the fast step that is scheduling. So, in next lecture we will see scheduling procedures, automated scheduling procedures that is what.

Thank you for that.