Design Verification and Test of Digital VLSI Designs

Dr. Santhosh Biswas
Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

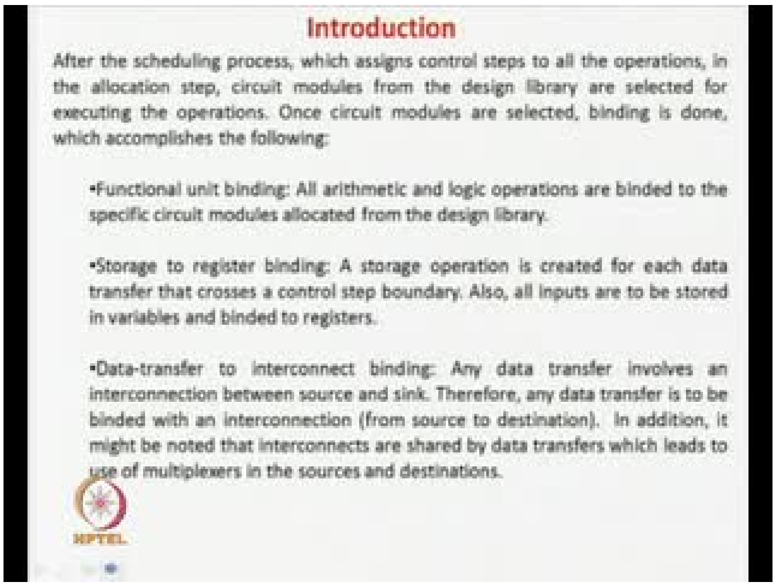
Module - 02
Scheduling, Allocations and Binding
Lecture - 04
Binding and Allocation Algorithms

So welcome to the 4th lecture on module 2, that is on allocation and binding algorithms, so in the last couple of lecture what we have seen is that the high level synthesis problem comprises of 3 steps scheduling, allocation and binding. Then we define what are the problems in formal terms and then we have seen that we require some automated cad tools or algorithms which can go on given a circuit and given some specifications.

So you can automatically get what you call schedule design, allocated design and binded design kind of a stuff. So in the last couple of double lectures what we have seen, we have seen some automate algorithms like what you call as soon as possible, as late as possible then force directed least scheduling and finally, individually programming the ILP based solutions or ILP based algorithm. All these days some algorithms were given input specification they will give you a schedule design based on some timing constraints or based on some resource constraints as in the case. \

So I mean then what do you find out that as already discussed in the high level synthesis problem the after the schedule has been done then we have to go for scheduling, then you have to go for allocation and then finally, we have to go for binding. So I mean after as we were discussing automated tools or algorithms so we have given a input specification you can either use the heuristics like FDA or as soon as possible, as late as or least scheduling or even if you have enough time I mean it takes prohibitic amount of time for ILP but, still you have time you can use exact algorithms. Then you can get a schedule design based on your time constraints or your resource constraints now. So once this is done…

(Refer Slide Time: 02:22)



Introduction

After the scheduling process, which assigns control steps to all the operations, in the allocation step, circuit modules from the design library are selected for executing the operations. Once circuit modules are selected, binding is done, which accomplishes the following:

•Functional unit binding: All arithmetic and logic operations are binded to the specific circuit modules allocated from the design library.

•Storage to register binding: A storage operation is created for each data transfer that crosses a control step boundary. Also, all inputs are to be stored in variables and binded to registers.

•Data-transfer to interconnect binding: Any data transfer involves an interconnection between source and sink. Therefore, any data transfer is to be binded with an interconnection (from source to destination). In addition, it might be noted that interconnects are shared by data transfers which leads to use of multiplexers in the sources and destinations.

So this is the input to the next 2 step of allocation and binding. So now will be looking in this lecture, we will be looking algorithms which will take you through other 2 steps of the high levels synthesis procedure that is allocation and binding. So if you recall so what is scheduling. Scheduling means you have given all the operations like addition 1 or 2 addition 1 addition 2 or subtraction 1 subtraction 2 kind of stuff. So all the operations will be given some time, some steps of operation then there is actually this is called scheduling. So after the scheduling has been done you have to go for allocation and then finally, binding.

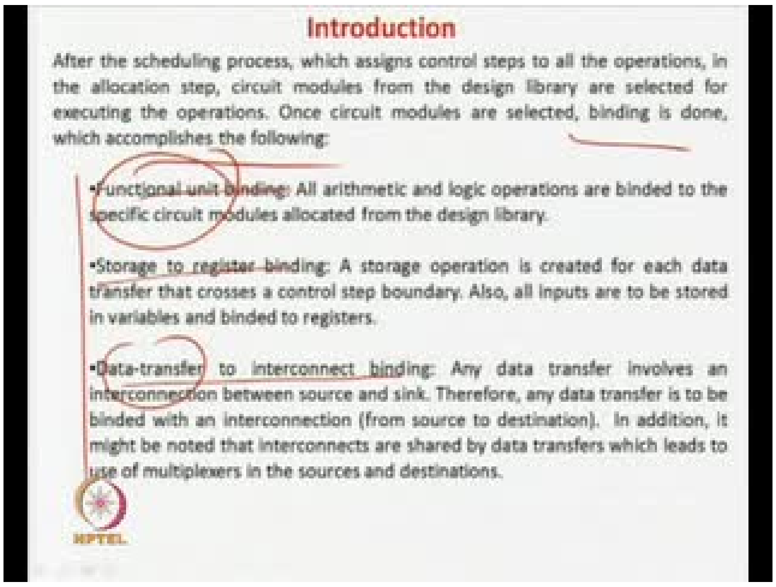
So what is allocation so if you remember what is allocation. So allocation means there will be lot of resources in the hardware like hardware, Carrey adder, array multiplier then some sequential multiplier so many so fourth hardware unit will be available in your library. Then you have to allocate different hardware units for different operations which have been already scheduled like for example, for adders you may require slow adder it may do your function I mean as you have discussed in first lecture of this module. So slow adder may do your purpose so you can use adder but sometimes we require a fast multiplier you can use what do you call a array type of a multiplier that will actually give you a very fast solution and sequential type of multiplier so that means in allocation step.

So what you do we allocate different hardware looking at the design library to all the operations. So you can think that this does not mean it a independent algorithm study so what the idea is even if there algorithms there will be semi automated type of an algorithms so in which case some manual intervention will be equated because just after if you just look at what is the scheduled stuff and how that scheduling has been done you moving at all this things. You can decide what type adder what type of multiplier what type of dividers or what type of hard ware function units are used in these operations. In that case I mean there is no such direct algorithms which requires to do the mapping you can find out what is the time taken, what is time step you know, what is delay of each of the adders or each of the hardwares you know, then you can decide on the time step delay or time given to each time step in the control structure and the time step 1 time step 2 what if you assume delay then what will be the delay amount based on that you have to select what is the hardware type you want.

So, in the first lecture example we have seen that if we require fast multiplier but, a slow adder will do the job for us so based on such type of studies we select the we actually allocate the operations. So you can understand that we do not have requisites not a sophisticated procedure neither it equates are too much of a algorithms automatic algorithms to do it just by looking at the some of the parameters you can decide on the allocation procedure. In other words I should not say that there is no algorithms if there is algorithms they will actually aid the designer they can help to find out what is the power requirement of the different hardware units available in the library. They can tell you what is the delay of those stuff then what is the delay that is tolerable for each of the control step and so forth.

So some of the information you can gather out from the designs using some tools and then you can decide what are the what units are to be allocated for what operations. So in this course we are not going into depth of such semiautomatic algorithms or what do I say semi automated procedure. Rather we think that this is a easy type of a job and even there is algorithms they will just give you information about different hardware units and the delay tolerable in your design now that is your control step delays and all these stuff. So information will be gathered by some of the tools and based on that steps designer can do what the designer can allocate different hardware units from design library to each of the operation. So what we see next that is the actually binding step so after there the allocation has been done that is all the control steps has been given. Some operation scheduling process allocation means allocating different hardware units from the design library to this operations then we have to go for the binding step ok that is the binding step.

(Refer Slide Time: 05:28)



Introduction

After the scheduling process, which assigns control steps to all the operations, in the allocation step, circuit modules from the design library are selected for executing the operations. Once circuit modules are selected, binding is done, which accomplishes the following:

•Functional unit binding: All arithmetic and logic operations are binded to the specific circuit modules allocated from the design library.

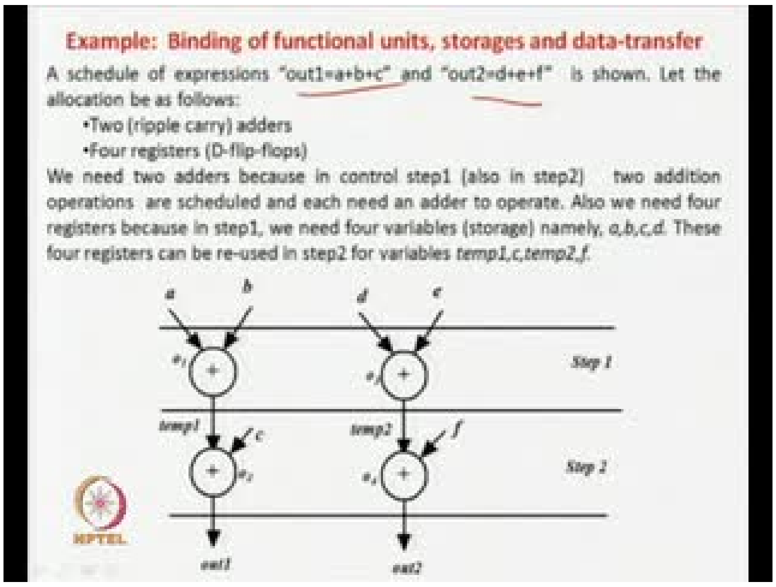
•Storage to register binding: A storage operation is created for each data transfer that crosses a control step boundary. Also, all inputs are to be stored in variables and binded to registers.

•Data-transfer to interconnect binding: Any data transfer involves an interconnection between source and sink. Therefore, any data transfer is to be binded with an interconnection (from source to destination). In addition, it might be noted that interconnects are shared by data transfers which leads to use of multiplexers in the sources and destinations.

So already discussed in the first lecture of this module that we have 3 type of binding that is functional unit binding that is addition or multiplication subtraction those things has to be binded to different hardware units that are available because if you require 3 adders in 1 step so that is the maximum number of adders required in 1 control steps. So you will have 3 adder units and all other addition operation will have to be shared within those functional units like we have adder 1, adder 2 and adder 3 for different operation. Say o 1 o 2 o 3 dot dot dot o n so all these operations has to be binded to within these 3 adders. That is all this adders addition operation will be shared among this 3 adders so that is how which addition operation you will bind to which hardware unit among the 3 adders you want to do operation this actually call the binding step or precisely functional unit binding then we have seen the storage to register binding.

So, we will have some n number of storage elements or registers then you have to find out which registers will be used to which variables this actually call storage register binding. Then let transfer to interconnect binding that is because of this type of sharing of resources we require some multiplex arrangements and inter connection between hardware units, functional units to storage units. And so for this so this actually, that actually control the data transfer that is putting up multiplex and wirings for doing this function units, storage register functional units, data transfer we may require some multiplexing arrangements and wire connections. Those actually comprise your inter connect or data transfer for interconnect binding so already we have seen that there are 3 different types of binding now we will some general algorithms which can actually help you to do this binding procedural.

(Refer Slide Time: 07:02)



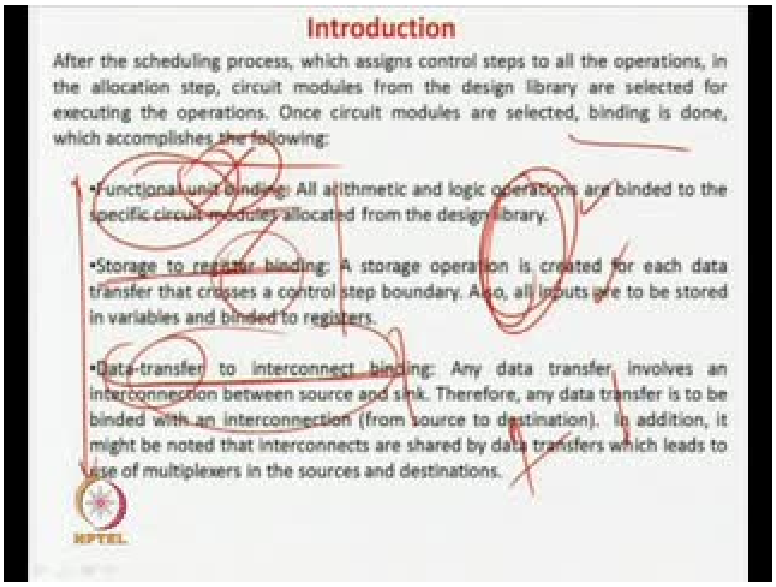
Example: Binding of functional units, storages and data-transfer

A schedule of expressions "out1=a+b+c" and "out2=d+e+f" is shown. Let the allocation be as follows:

- Two (ripple carry) adders
- Four registers (D-flip-flops)

We need two adders because in control step1 (also in step2) two addition operations are scheduled and each need an adder to operate. Also we need four registers because in step1, we need four variables (storage) namely, *a,b,c,d*. These four registers can be re-used in step2 for variables *temp1,c,temp2,f*.

(Refer Slide Time: 07:23)



Introduction

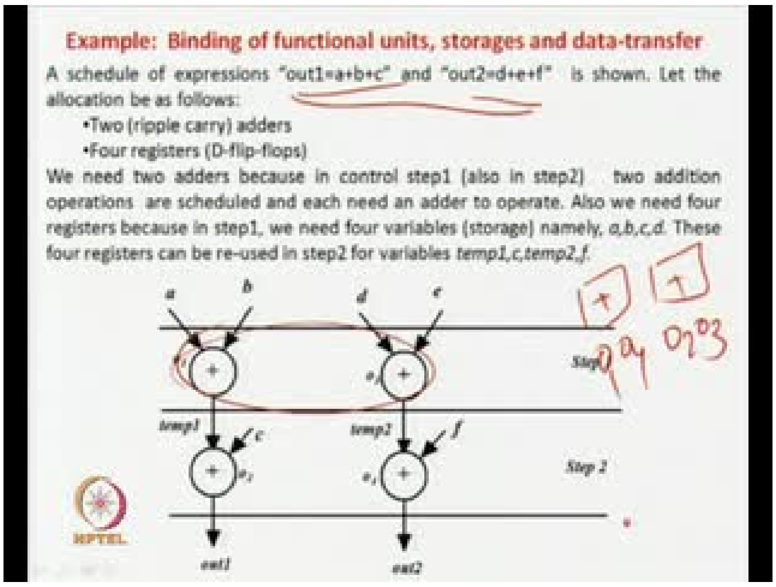
After the scheduling process, which assigns control steps to all the operations, in the allocation step, circuit modules from the design library are selected for executing the operations. Once circuit modules are selected, binding is done, which accomplishes the following:

- Functional unit binding: All arithmetic and logic operations are binded to the specific circuit modules allocated from the design library.
- Storage to register binding: A storage operation is created for each data transfer that crosses a control step boundary. Also, all inputs are to be stored in variables and binded to registers.
- Data-transfer to interconnect binding: Any data transfer involves an interconnection between source and sink. Therefore, any data transfer is to be binded with an interconnection (from source to destination). In addition, it might be noted that interconnects are shared by data transfers which leads to use of multiplexers in the sources and destinations.

So, now we before go to the automatic algorithms so what we will see first we take a very simple example like out 1 is equal to a plus b plus c and out 2 is equal to d plus e plus f. So these are the 2 operations we want to carry out. So we will see that what do we find out that these 2 different type of bindings you do for storage as well as these two functional unit and storage register binding based on different type of storage register and what you call the functional unit binding. We will see that automatically the third part that is your data transfer or interconnect binding area changes that is very important. So what is the goal? How to go generally go out to solving the problem? So what you do you first go for the functional unit binding then we will say that we will go to register binding. I mean you should not say that they all are separate actually they are hand in hand so actually go hand in hand, let us see how it goes hand in hand. So for example, we have to start at 1 point.

So let us start with functional unit binding then what we will do they will go for storage register binding then finally, we will find out that once these binding are done automatically this data transfer binding will come into picture. Because you have to put multiplexer and wiring then we will to find out based on these two the area of third step actually vary. So if you want to minimise these area then again you have to click with two stuff that why we call hand in hand. Some cases the area will be very high in some cases the area will be very low. So in this case you have to adjust this first two parameters functional unit binding and function storage binding. So you have to adjust accordingly so that you can get minimum inter connect binding. So we will see algorithms that will actually take care of this procedural into picture this factor into picture the arise will share.

(Refer Slide Time: 08:39)



Example: Binding of functional units, storages and data-transfer

A schedule of expressions "out1=a+b+c" and "out2=d+e+f" is shown. Let the allocation be as follows:

- Two (ripple carry) adders
- Four registers (D-flip-flops)

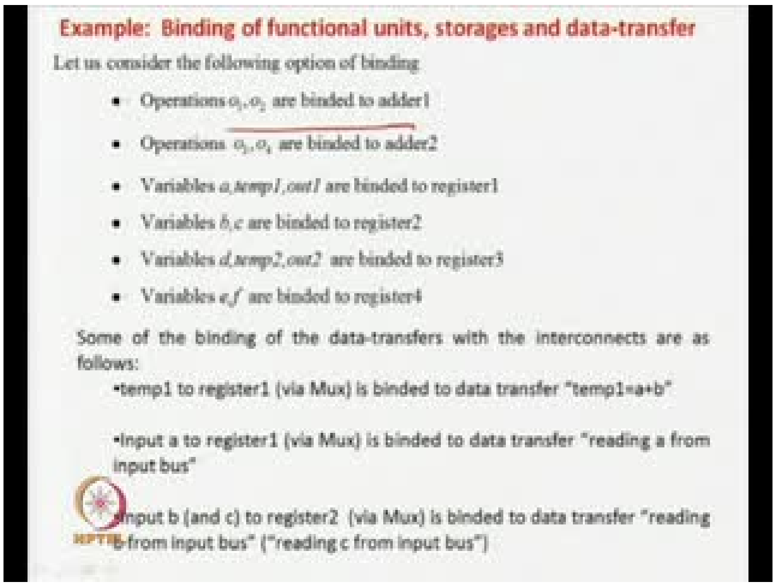
We need two adders because in control step1 (also in step2) two addition operations are scheduled and each need an adder to operate. Also we need four registers because in step1, we need four variables (storage) namely, *a,b,c,d*. These four registers can be re-used in step2 for variables *temp1,c,temp2,f*.

So before that we will give an example a very small expression that how would exactly happen and how based on functional unit binding and the register binding your inter connect binding area will change. So let us think that a plus b plus c and this is the case so let us this be schedule based on these as soon as possible.

So whatever you take so o 1 o 2 o 3 o 4 these are the schedule step it is done. Then you get the allocation we use all ripple carry adders because the speed or time step for which control the step. So the ripple carry adder frequency time taken by the ripple carry adder to do the job is enough that needs this delay. So we have taken ripple carry that is actually the allocation has already been done with ripple carry adder and schedule is o 1 o 2 o 3 o 4 as shown in the figure. Now we have to go for binding so you can see that as we have only 2 adders in 1 time step. So you can easily visualise that what is the idea that to have to adder block maximum.

So 2 adders are required now you have to allocate bind actually o 1 o 2 o 3 o 4 to this adders. So you can think that I will do o 1 o 2 here and o 3 o 4 here. This is the new 1 binding actually we are doing functional unit binding as I told you we will do functional unit binding because also we can think that way that we will do o 1 o 4 here and o 2 o 3 here that is all possible permutation combination can be attempted because we have 2 adders and 4 addition operation. So these are shared between these 2 adders then you have to see for we will take all the different difference binding allocations different permutation and combination are possible that we will take and then we will see what is the area over it due to it interconnect binding that automatically comes into the (( )).

(Refer Slide Time: 10:22)



Example: Binding of functional units, storages and data-transfer

Let us consider the following option of binding

- Operations o_1, o_2 are binded to adder1
- Operations o_3, o_4 are binded to adder2
- Variables *a, temp1, out1* are binded to register1
- Variables *b, c* are binded to register2
- Variables *d, temp2, out2* are binded to register3
- Variables *e, f* are binded to register4

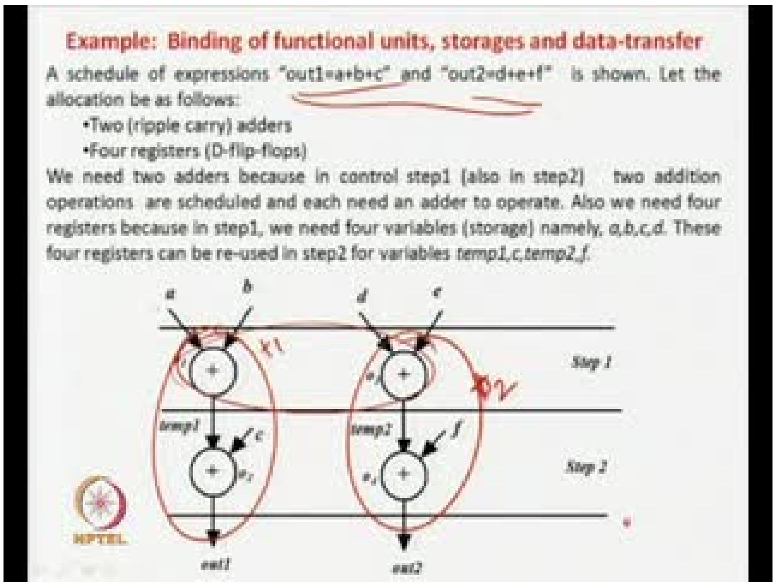
Some of the binding of the data-transfers with the interconnects are as follows:

•temp1 to register1 (via Mux) is binded to data transfer "temp1=a+b"

•Input a to register1 (via Mux) is binded to data transfer "reading a from input bus"

•Input b (and c) to register2 (via Mux) is binded to data transfer "reading b from input bus" ("reading c from input bus")

(Refer Slide Time: 10:27)



Example: Binding of functional units, storages and data-transfer

A schedule of expressions "out1=a+b+c" and "out2=d+e+f" is shown. Let the allocation be as follows:

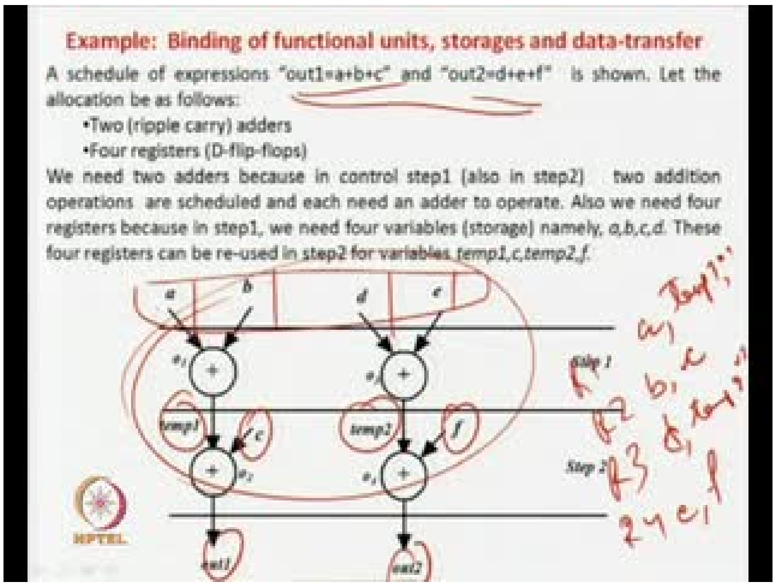
•Two (ripple carry) adders

•Four registers (D-flip-flops)

We need two adders because in control step1 (also in step2) two addition operations are scheduled and each need an adder to operate. Also we need four registers because in step1, we need four variables (storage) namely, *a,b,c,d*. These four registers can be re-used in step2 for variables *temp1,c,temp2,f*.

So, now we also have some will say now we take this 1 so o 1 and o 2 are binded with adder 1 and o 3 and o 4 are binded to adder 2. So this 1 is for adder 1 and this 1 and this 1 is for adder 2 so 2 adders are there so adder 1 will be doing this 2 operation and adder 2 will be doing this 2 operation. So this is about the functional unit binding we have done now you can see that what are the variables and so that we can go for register binding. So you can see that a b c d there are 4 variables are there they are alive in 1 go so minimum 4 register has to be there you cannot use 3 registers because a b c d they are all appearing in the first time step. So there are 4 register to store that you cannot reuse so in this case we have 2 adders in 1 control step.

(Refer Slide Time: 11:07)



Example: Binding of functional units, storages and data-transfer

A schedule of expressions "out1=a+b+c" and "out2=d+e+f" is shown. Let the allocation be as follows:

- Two (ripple carry) adders
- Four registers (D-flip-flops)

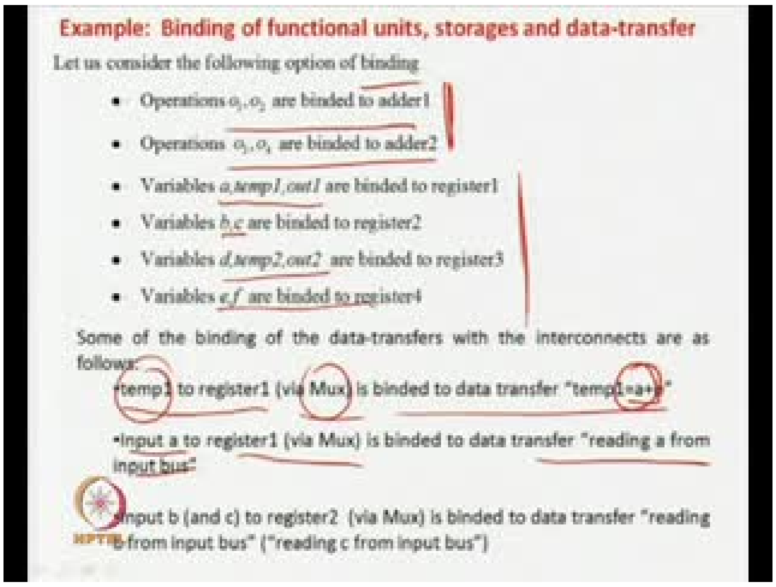
We need two adders because in control step1 (also in step2) two addition operations are scheduled and each need an adder to operate. Also we need four registers because in step1, we need four variables (storage) namely, *a,b,c,d*. These four registers can be re-used in step2 for variables *temp1,c,temp2,f*.

So at least 2 adders max minimum 2 adders you have you need to have otherwise you cannot go for this what you say execution of this circuit. So similarly, we have 4 registers over here 4 variables over here a b c d so 4 registers minimum are required so we can say that r 1 r 2 r 3 r 4 minimum are there. And then what we have to do you have to allocate you have to bind a b c a b d e to this things and what are the other variables like temp 1 will be generated after the first addition is done c is another variable temp 2 is another f is another variable and finally, out 1 and out 2 are generated. So you can think that a b c d are all alive in time step 1 so they have to be binded to be different.

So I say like this r 1 r 2 r 3 r 4 i say a b and I say d and e so this is 1 stuff I say a b d e a b d e they all are allocated at two different registers binded to different registers a b d and e a b d and e. Because they all are felling in the first time step so you cannot share register among them but now you can see that after this a plus b has been done then register r 1 is free. So I can say that so I use it for temp 1 so I can bind temp 1 with that you will also be free in the time step 2. So I can say that d and c I put it in the register 2 this way I can do similarly, in case of this stuff you can see that r 3 will be free after d and e has been added. So you can put temp 2 over here because temp 2 you can store in the register number 3 and then what you can say that after first addition d plus c that register 4 is also free. So you can allocate f to this 1 right it is been done and then in the final step out 1 and out 2 will be ready.

So when out 1 and out 2 will be arising all the registers will be free. So you can store out 1 and out 2 anywhere excepting you cannot share them. So you can say that in register 1 I put out 1 and in register number 3 I can put out 2 so any of them you can try out. So first you have to add 4 variable 4 registers so 1 2 3 4 will go over there then anyone of them you can reuse for temp 1 and another for c 3 c because temp 1 and c cannot be over lapped because they are having same life time. Similarly, temp 2 and f they also cannot be over lapped with this one so temp 1 c temp 2 f they can be binded to different 4 registers as we have. Then finally, the second series of addition are also done then out 1 and out 2 will be ready so they can be again stored in two different registers because out 1 and out 2 life is not overlapping with anyone of these variables.

(Refer Slide Time: 13:29)



Example: Binding of functional units, storages and data-transfer

Let us consider the following option of binding

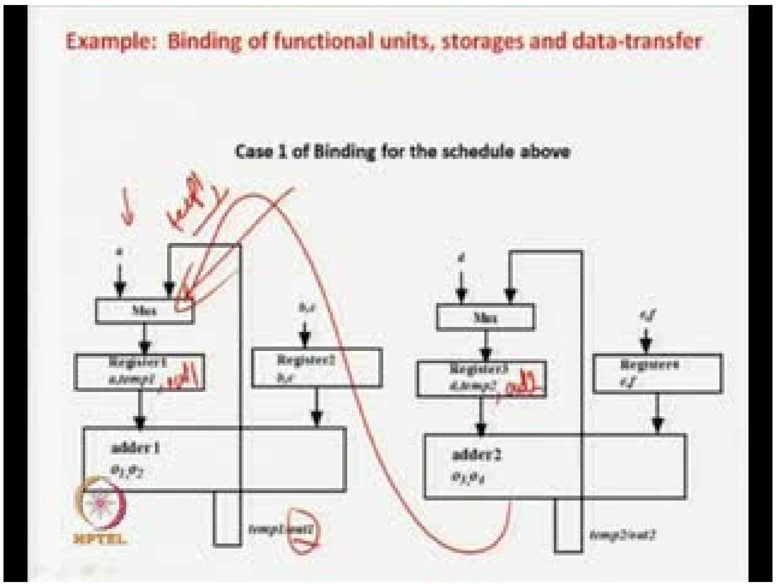
- Operations o_1, o_2 are binded to adder1
- Operations o_3, o_4 are binded to adder2
- Variables *a,temp1,out1* are binded to register1
- Variables *b,c* are binded to register2
- Variables *d,temp2,out2* are binded to register3
- Variables *e,f* are binded to register4

Some of the binding of the data-transfers with the interconnects are as follows:

- temp1 to register1 (via Mux) is binded to data transfer "temp1=a+b"
- Input a to register1 (via Mux) is binded to data transfer "reading a from input bus"
- Input b (and c) to register2 (via Mux) is binded to data transfer "reading b from input bus" ("reading c from input bus")

So let us see first this one first what you call optional binding o 1 o 2 o 3 o 4 have been added have been binded to register 1 adder 1 adder 2 respectively. Now in register number 1 they have done a temp 1 out and register 2 they have done b c variable d temp 2 out are binded in register 3 and e and f are binded to register 4 that has been already binded. Binding already has been done so these are your operational unit binding and these are your register or storage binding you have done. Now you see automatically will be requiring some multiplex will automatically require to do the inter connections because you can see in register a you will be storing a temp then out 1 in this case you will be storing d c in this case you will be storing d temp 2 and out and so different variables will come to the registers so again you require multiplex.

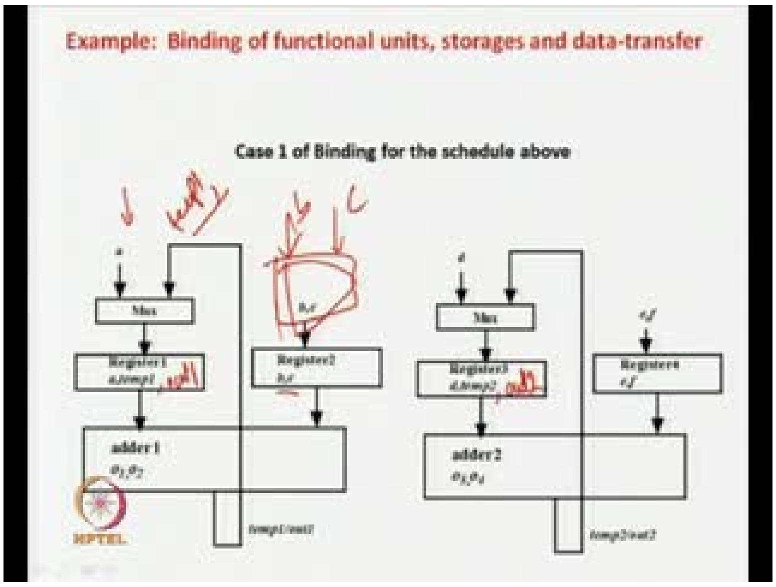
(Refer Slide Time: 15:00)



So when you require multiplexes throughout channels different variables are expected to arrive then use a multiplexer. So you can see temp 1 to for register 1 a via multiplexer is binded to data transfer this 1. So what did you see that temp 1 is bringing temp 1 a plus b a plus b will be binded to register 1 via some multiplexer so now why the multiplexer is required because input a is also binded to register 1 that is data transfer what is data transfer? Data transfer is reading from input bus and also same register r 1 temp 1 is binded where the data is arising from the operation a plus b. So obviously we require multiplexing over here that is what has been told over here so let us see the figure then again will come back what you have seen telling you that eight temp 1 and out 1 are binded to register 1 then b and c th register are 2 d temp 2 and out 2 this one is out 2 is binded over this one and a and f are the register 4.

Whereas see what happens so when you say that first step that is a plus b now see that this register is there what is going to register a is going to register that is some input bus temp is also coming to here that is actually coming here output of the add here. So that is actually said temp 1 and then find out the after all the operation out 1 will also go to register 1 that is again temp 1 and out 1 is coming from temp 1 and out 1 is coming from the output of the adder. So output of adder is one kind this one is register and a is one pin which is connected to the input to the register so you accurate two in to one multiplexer so had it with the out is arriving from some other place from other circuit and so for let us assume so it could have been done is coming from this circuit and so in that case you have required 4 is to 1 multiplex because of 4 will be not used in this case but, then 2 passes would not have been 2 is to 1 multiplexer could not done the job for you because you required 3 pins i mean 3 input ports or 3 port from where data is arrived so in this only 2 part i mean 2 connection from the data is arriving.

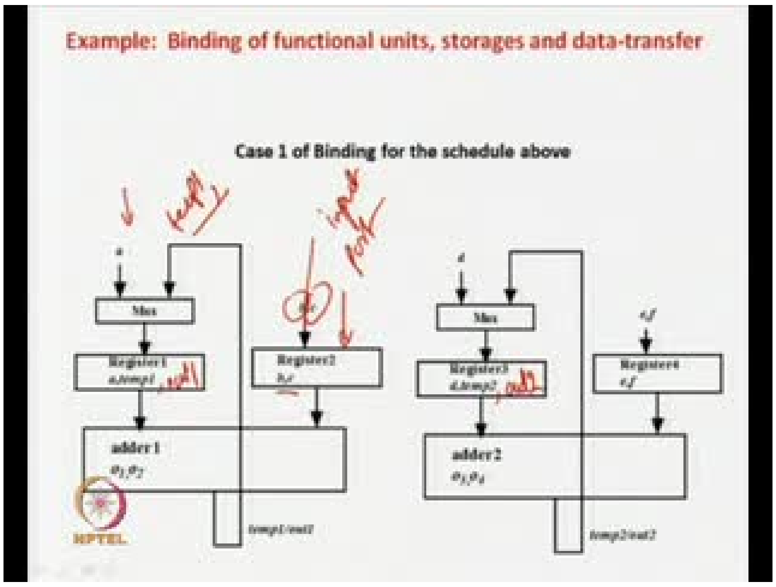
(Refer Slide Time: 16:32)



Example: Binding of functional units, storages and data-transfer

Case 1 of Binding for the schedule above

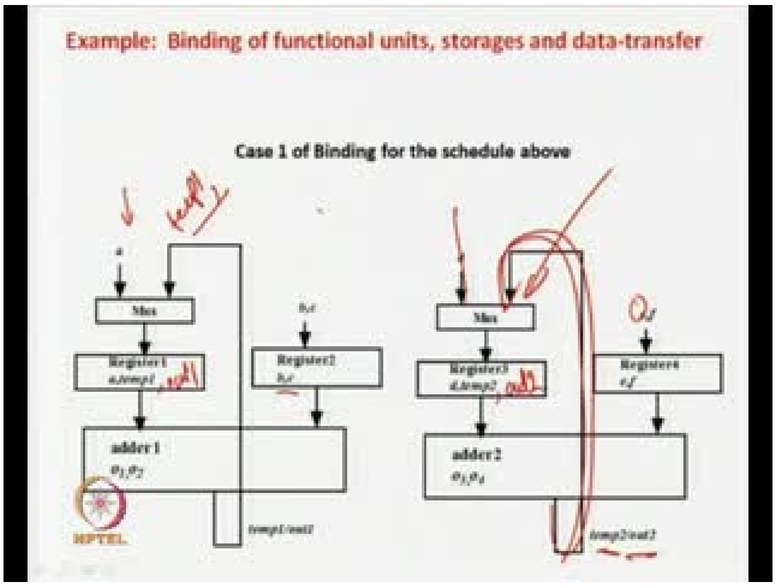
(Refer Slide Time: 16:44)


Example: Binding of functional units, storages and data-transfer

Case 1 of Binding for the schedule above

So you can thing that these 2 is to 1 margins is fine so in this case that is actually this story 1 register is 1 this register 2 and see now how do you have not put the register b and c over here i have not put multiplexer over here i could be b or c here why i have not done that i have directly connected b and c over here the answer is simple actually b and c are connected directly to the input port input port is directly connected and in this case what you have done in the time step 1 b is available and in the time step 2 c is available so you dont actually required to different ports to or 2 these available b is available and in time step 2 c is available so we do not require a multiplexer even those 2 variables are binded to this 1 similarly, in case of this register number 3.

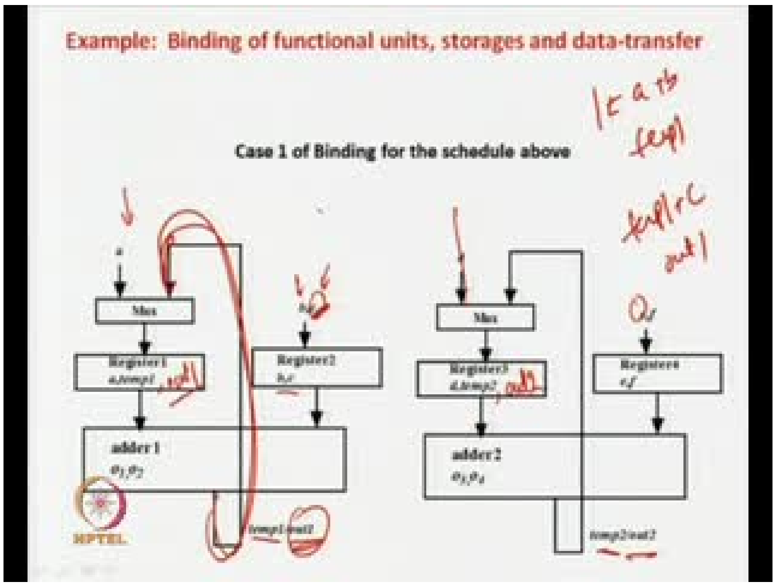
(Refer Slide Time: 17:10)


Example: Binding of functional units, storages and data-transfer

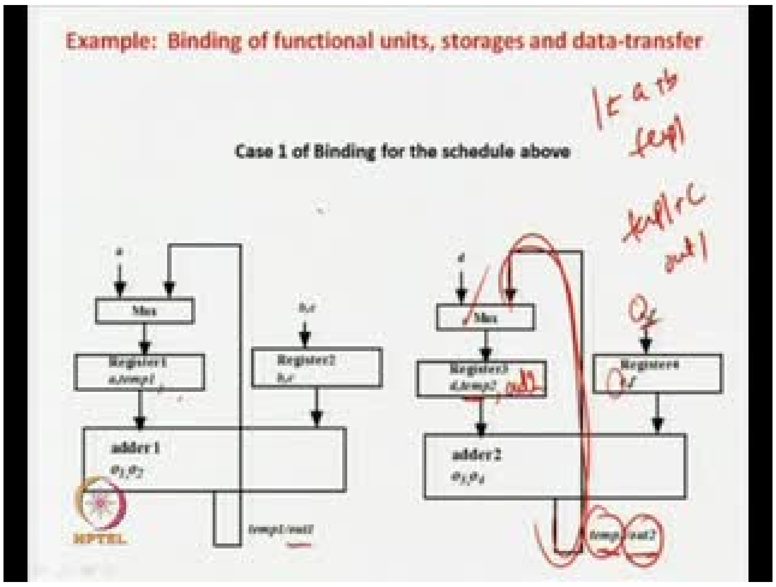
Case 1 of Binding for the schedule above

So 1 port is d and it control step 1 where the data is coming from the control bars input bars and then time steps time steps 2 temp 2 will generated by adding d plus e so you require the where the data will drive here from another line that is the output of the register number 2 output of adder number 2 so require a multiplexer over here and finally, out 2 will be generated which is also be binded register 3 but in this case what happens if you see that is the same output port of the adder is actually bring in temp 2 and out 3 so you do not have any third pin so 2 input 2 by 1 multiplexer is actually doing the job so what you have found out so in this case at time step 1 a is going over here to the register b is going over here addition is done temp 1 is generated it is told as by marks is again stored at register number 1. After that what happens out 1 is adder to b sorry out 1 is adder to c that generate out 1 sorry sorry.

(Refer Slide Time: 18:03)


Example: Binding of functional units, storages and data-transfer
Case 1 of Binding for the schedule above
Mux
Register1
a,temp1
Register2
b,c
adder 1
Mux
Register3
d,temp2
Register4
e,f
adder2
temp1/out2
temp2/out2
NPTEL

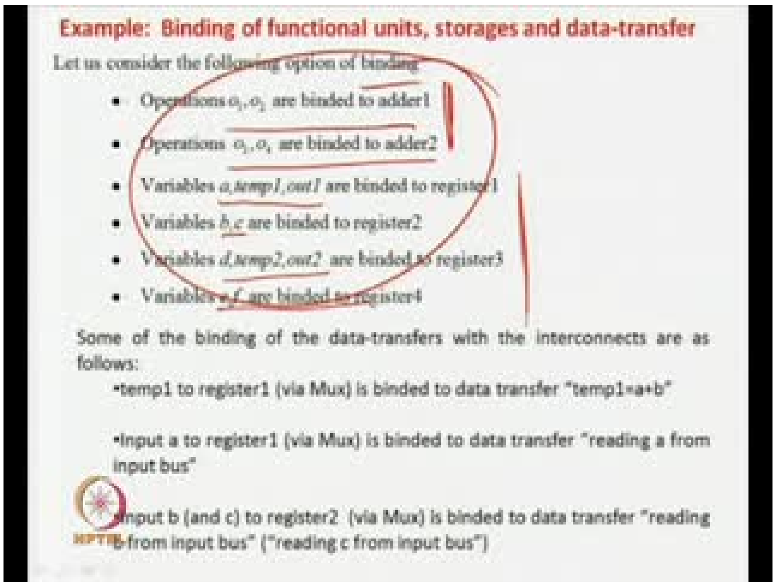
(Refer Slide Time: 18:56)


Example: Binding of functional units, storages and data-transfer
Case 1 of Binding for the schedule above
Mux
Register1
a,temp1
Register2
b,c
adder 1
Mux
Register3
d,temp2
Register4
e,f
adder2
temp1/out2
temp2/out2
NPTEL

Temp 1 is added with added in time step 1 you are adding a plus b and generated temp 1. Which is told as the register 1 through the second pin that is through this pin stored over here now in the time step 2 what is happen in temp 1 is added to this c plus c and you are generated out 1 so again the out 1 is generated by adder in time step 2 and time step 3 what happens the same variable this out 1 so actually come out of the adder and is being stored in register number 1 by this multiplexer so this what is the 3 steps that is in the being done so in this case first step b is there second step c is there so that is automatically happening form the input bus you don't require a any multiplexer over here so that is what we how the this sequence of in time step 1 and time step 2 what happens we are actually generating a plus b and then temp 1 is added to c to generate out 1 similarly, in this case time step 1 d is bring saved in register 1 by this port. And it is adding the added p then e is coming from the input port by register port.

With e is coming from the input port wire is added and you are generating the value of temp 2 so temp 2 is stored over here and in control step 2 temp 2 is added over f and we are finding here out to again stored over register 3 by this output of the number 2 these how you are generating what do you say

(Refer Slide Time: 19:26)



Example: Binding of functional units, storages and data-transfer

Let us consider the following option of binding

- Operations o_1, o_3 are binded to adder1
- Operations o_2, o_4 are binded to adder2
- Variables *a,temp1,out1* are binded to register1
- Variables *b,c* are binded to register2
- Variables *d,temp2,out2* are binded to register3
- Variables *e,f* are binded to register4

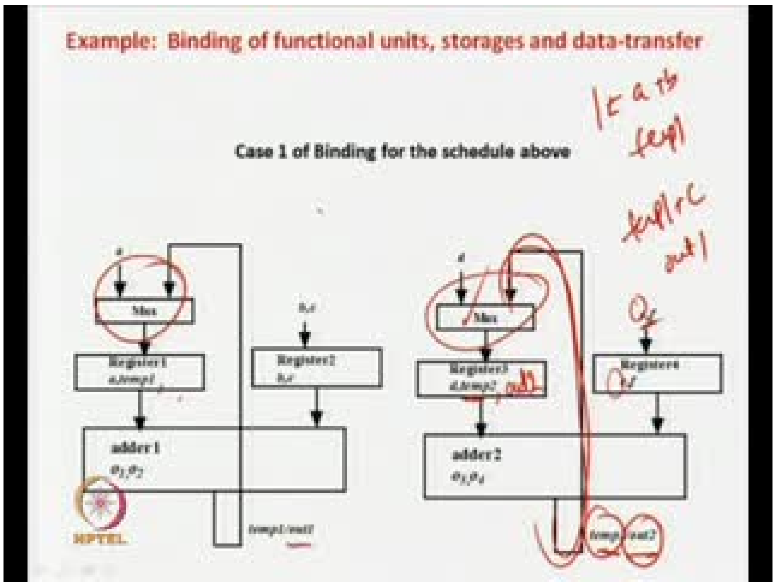
Some of the binding of the data-transfers with the interconnects are as follows:

•temp1 to register1 (via Mux) is binded to data transfer "temp1=a+b"

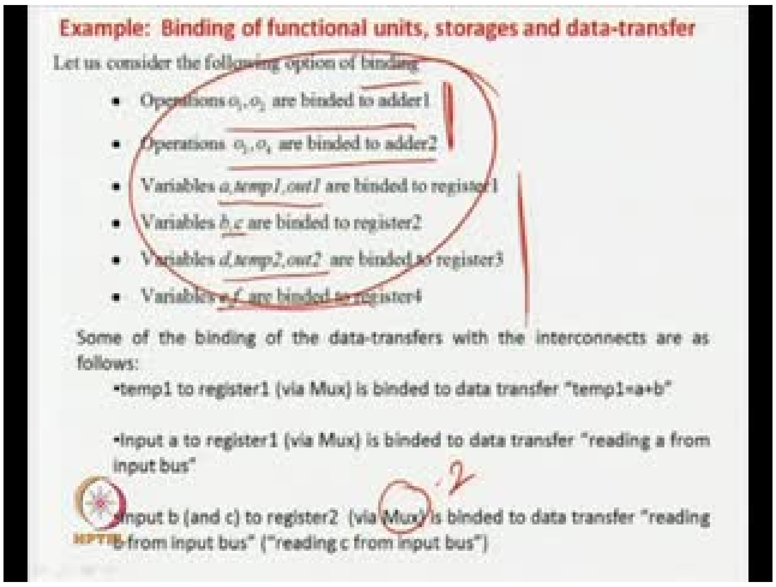
•Input a to register1 (via Mux) is binded to data transfer "reading a from input bus"

•Input b (and c) to register2 (via Mux) is binded to data transfer "reading b from input bus" ("reading c from input bus")

(Refer Slide Time: 19:29)



(Refer Slide Time: 19:39)



Example: Binding of functional units, storages and data-transfer

Let us consider the following option of binding

- Operations o_1, o_2 are binded to adder1
- Operations o_3, o_4 are binded to adder2
- Variables *a,temp1,out1* are binded to register1
- Variables *b,c* are binded to register2
- Variables *d,temp2,out2* are binded to register3
- Variables *e,f* are binded to register4

Some of the binding of the data-transfers with the interconnects are as follows:

•temp1 to register1 (via Mux) is binded to data transfer "temp1=a+b"

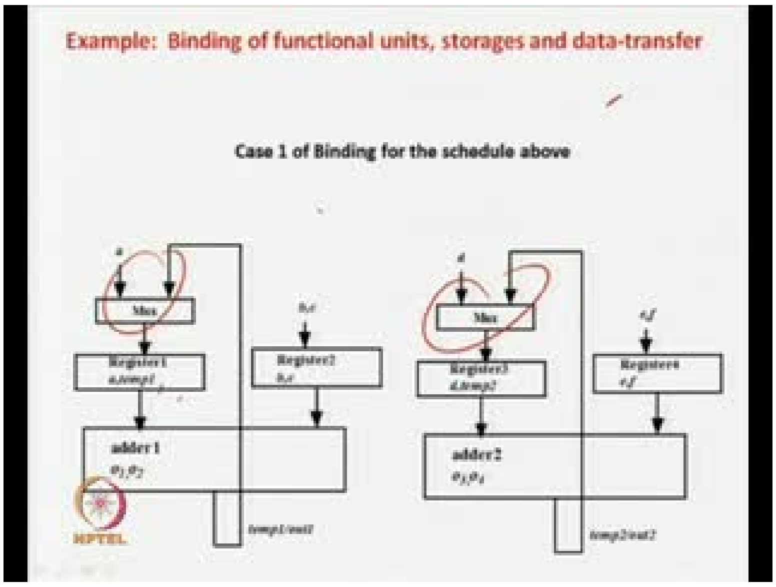
•Input a to register1 (via Mux) is binded to data transfer "reading a from input bus"

•Input b (and c) to register2 (via Mux) is binded to data transfer "reading b from input bus" ("reading c from input bus")

2

Temp 1 and temp 2 plus c and in this case plus a and finding out 1 and out 2 so by these bind we are actually having this binding so what are the interconnect binding used 1 multiplexer over here and 1 multiplexer over here so in an action what we have seen if we are going for these type of operation binding the time what do you call the marks required is actually 2 which is the interconnect binding over and some of the wiring which come under interconnect binding now we will see other option because only 1 option we have seen in this case we have added o 1 o 2 over here o 3 o 4 over here and then we have gone for a temp 1 out 1 b c a temp 2 out 2 and a in this case and this is 1 option the later the area was with this.

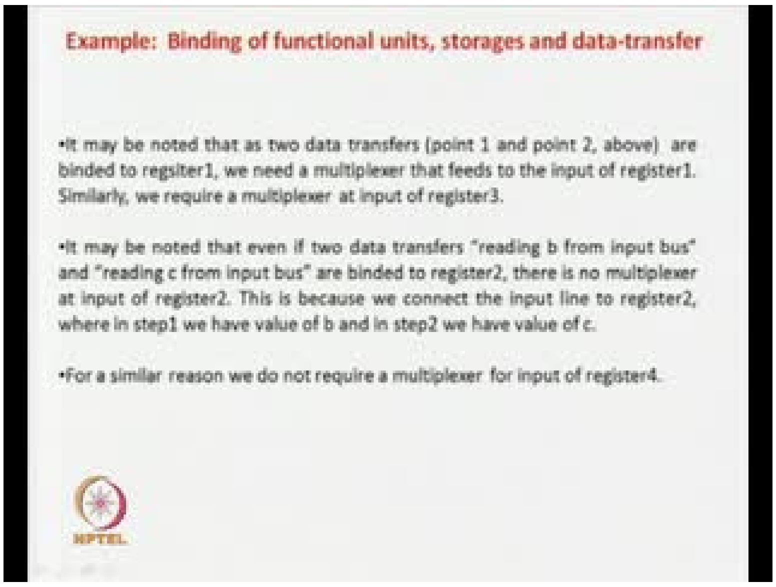
(Refer Slide Time: 20:01)



Example: Binding of functional units, storages and data-transfer

Case 1 of Binding for the schedule above

(Refer Slide Time: 20:04)



Example: Binding of functional units, storages and data-transfer

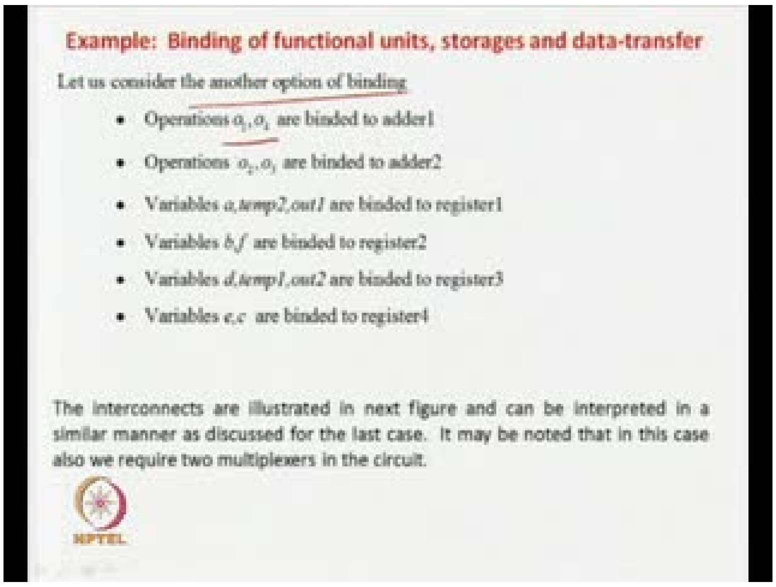
•It may be noted that as two data transfers (point 1 and point 2, above) are binded to regsiter1, we need a multiplexer that feeds to the input of register1. Similarly, we require a multiplexer at input of register3.

•It may be noted that even if two data transfers "reading b from input bus" and "reading c from input bus" are binded to register2, there is no multiplexer at input of register2. This is because we connect the input line to register2, where in step1 we have value of b and in step2 we have value of c.

•For a similar reason we do not require a multiplexer for input of register4.

2 multiplex at with this time of binding. And now late us take 1 more option 1 more option in this case 1 another 2 3 options will see for different type of bindings and we will see the interconnect area. So in this case whatever i have told you that why do you require a register and all.

(Refer Slide Time: 20:33)



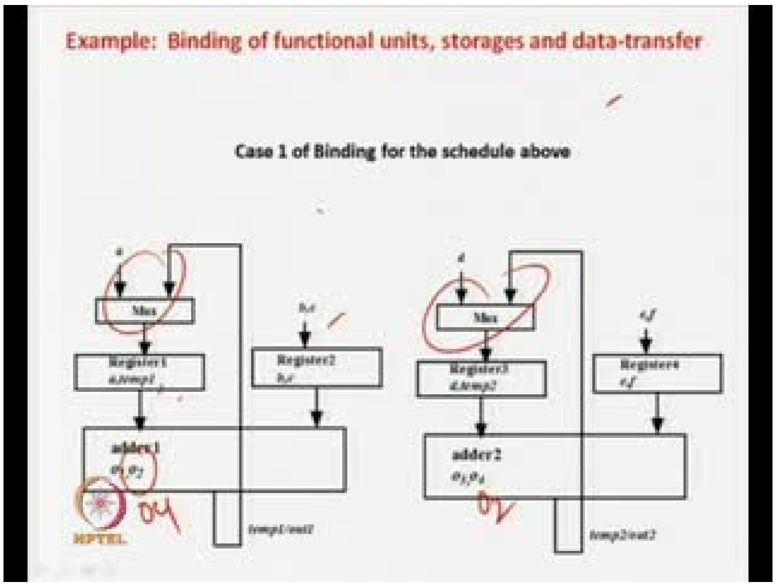
Example: Binding of functional units, storages and data-transfer

Let us consider the another option of binding

- Operations o_2, o_4 are binded to adder1
- Operations o_1, o_3 are binded to adder2
- Variables *a,temp2,out1* are binded to register1
- Variables *b,f* are binded to register2
- Variables *d,temp1,out2* are binded to register3
- Variables *e,c* are binded to register4

The interconnects are illustrated in next figure and can be interpreted in a similar manner as discussed for the last case. It may be noted that in this case also we require two multiplexers in the circuit.

(Refer Slide Time: 20:46)

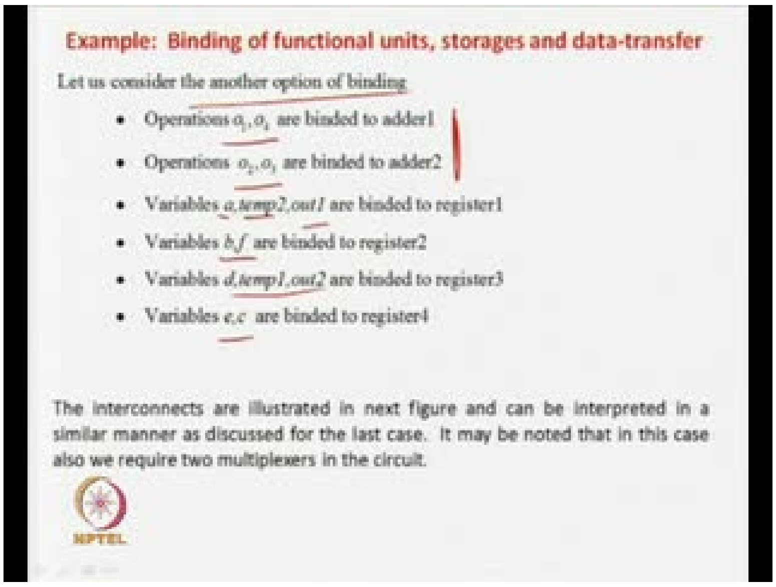


Example: Binding of functional units, storages and data-transfer

Case 1 of Binding for the schedule above

So why do you require multiplexer at some points and like why do you require multiplexer here and why do not you require a multiplexer over here etcetera written in this is i mean written this s slide you can through this go to the slide i have told you the same thing is mention in this slide. Now we are going for another option of binding so what is the next option of binding so in this case we are actually saying o 2 and o 4 in adder 1 so in this case it is o 1 and o 2 now we made here o 4 and in this case it is o 3 and o 2 so this 4 has gone there and 2 has come over there that is what we have done over so you can see there you are seeing that o r and o 4 are binded to either o 1 and o 2 and o 3 has been binded to adder 2 so that are some different.

(Refer Slide Time: 20:58)



Example: Binding of functional units, storages and data-transfer

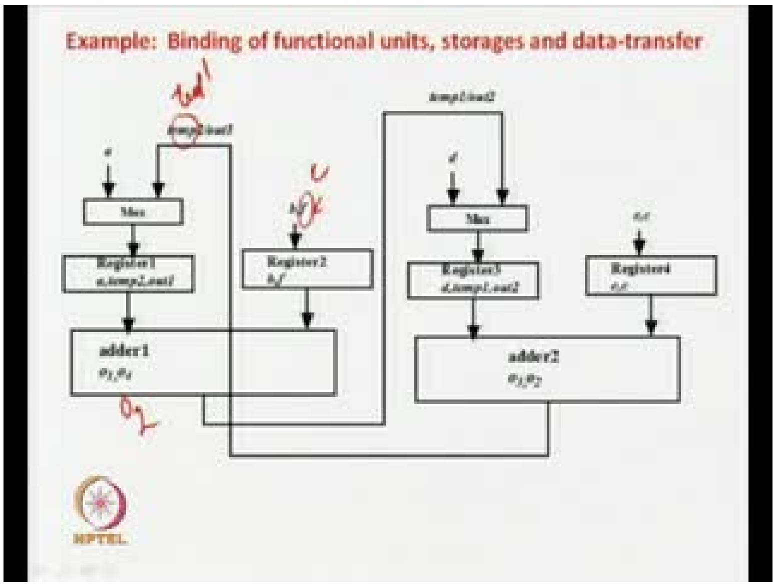
Let us consider the another option of binding

- Operations o_1, o_4 are binded to adder1
- Operations o_2, o_3 are binded to adder2
- Variables *a,temp2,out1* are binded to register1
- Variables *b,f* are binded to register2
- Variables *d,temp1,out2* are binded to register3
- Variables *e,c* are binded to register4

The interconnects are illustrated in next figure and can be interpreted in a similar manner as discussed for the last case. It may be noted that in this case also we require two multiplexers in the circuit.

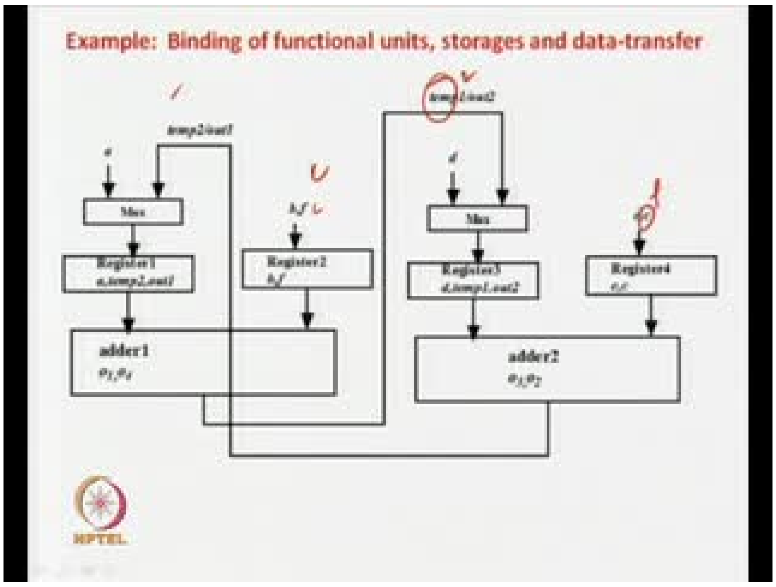
(Refer Slide Time: 21:40)

Example: Binding of functional units, storages and data-transfer



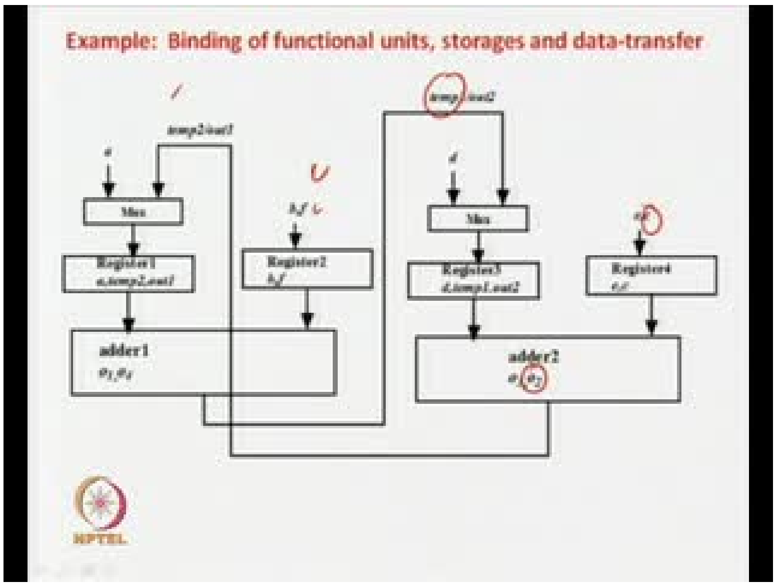
Stuff we have done and again some permutation combination also have done over so variable a temp1 and out 1 stored has so initially it was step 1 now a may be temp 2 here binding to register 1 b and f has been binded with register 2 so initially we has b c now you have made it b f because actually 4 is coming over here in for adder 1 4 over in adder 1 4 in the now we are actually doing operation o 1 and o 4 so o 4 is actually taking temp to plus f so we are actually allocating binding temp to register 1 and f 2 register 2 kind of a thing and similar by d temp 1 and out 2 adder by set register 3 and e and c are binded to register 4 now let us see in this slide i mean the slide how we what we have done so in this case initially in the adder 2 now it is adder 4 operation 4 so operation 4 is actually takes temp 2 and f so instead of having temp 1 here and c over here we have made in b and made it temp 1 because if it is adder o 2 then is actually add temp 1 plus c so that would have there was the previous.

(Refer Slide Time: 22:09)


Example: Binding of functional units, storages and data-transfer

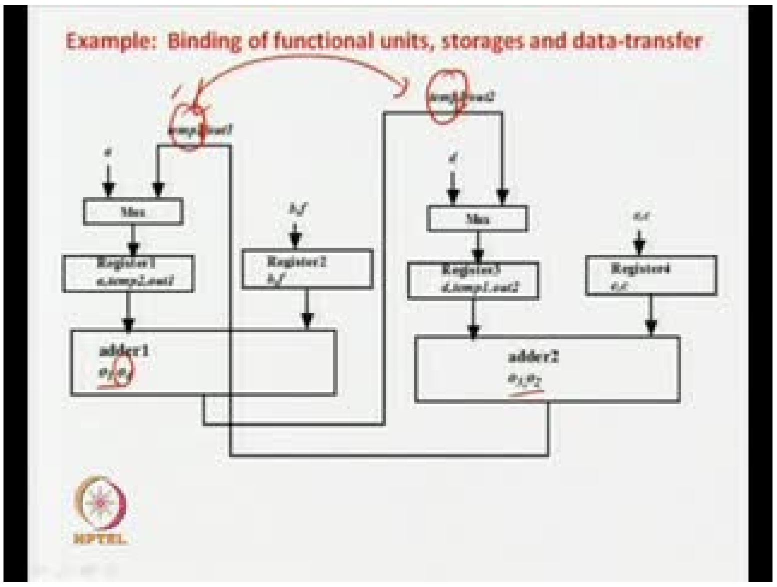
Case now we have put o 4 over here so we are actually we have put in temp 2 over here and f over here similarly, we have taking o 2 over o 3 over here so, in this case if you see so it is o 3 over here so, if you take a o 3 over here so in this case o 3 will be actually temp 1 plus c so that you generate actually output 1 so in this case instead of having a temp 2 in the previous case and f over here so we have replace with the temp of 1 and c so by doing this step of we are achieving by doing this what we have achieved this that we have actually broad this o 2 over here.

(Refer Slide Time: 22:35)

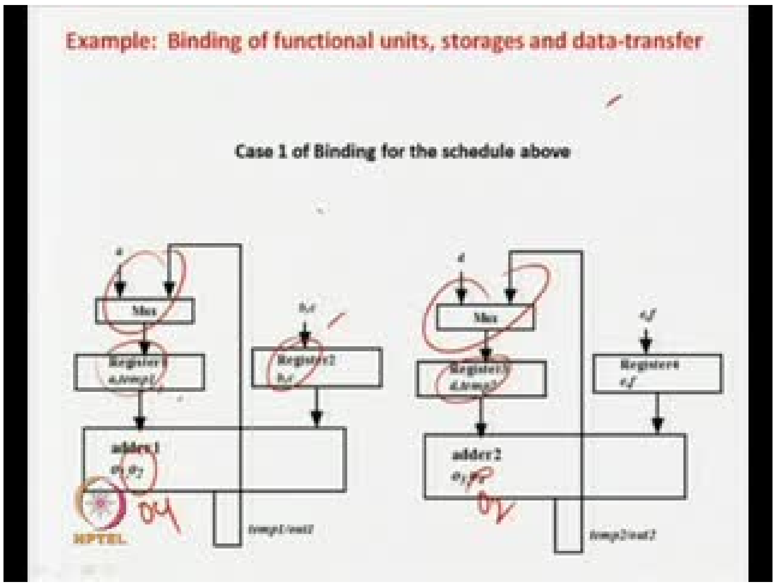


So o 2 will be adding temp 1 plus c some allocation sorry some binding we have reshuffled here . Now if you can also see in this case also, so our multiplexer require the 2 as will very shortly see so but, 1 thing i should tell you that we are not i mean in this case we have just starting some arbitrary bindings not that we have saying that o 1 and o 4 and o 3 and o 2 have been done here that.

(Refer Slide Time: 22:58)

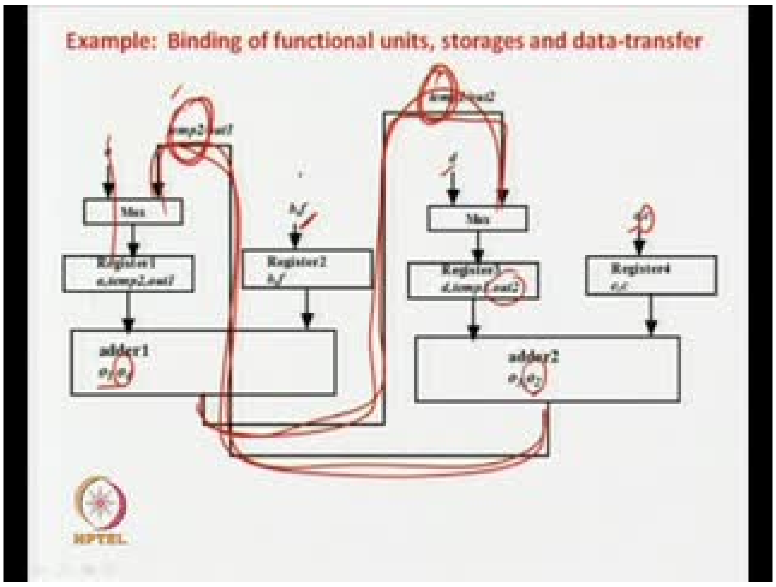


(Refer Slide Time: 23:47)


Example: Binding of functional units, storages and data-transfer
Case 1 of Binding for the schedule above
Mux
Register1
a,temp1
Register2
b,c
adder1
temp1/out1
Mux
Register3
d,temp2
Register4
e,f
adder2
temp2/out2
o4
o2

Is why we have replace temp 1 temp 2 and in this case you have replace temp 1 temp 2 that i means we have just swapped this values correspondence to the compare to the previous example previous case of by binding example so we just swapped it. So that is the 1 of the modification of doing what is the general speaking case it could be other way round also we could have put o 1 o 4 o 3 o 2 here and if you have put it f 1 here and temp 2 here that is also possible so that is i mean very logically speaking if you are moving 1 o 4 here so we should also bring temp 2 over here and if you are putting over here so we should logically put temp 1 over here because they are mean they are binded 2 o 2 but, actually very generalise case you can try also with this 1 like you can keep this what you call this register binding fixed and we could have say that o 1 o 4 o 3 o 2 that is also possible then it will started d the area that is why i told you that different type options are possible different binding types of option will be.

(Refer Slide Time: 23:57)


Example: Binding of functional units, storages and data-transfer
Mux
Register1
Register2
Mux
Register3
Register4
adder1
adder2
NPTEL

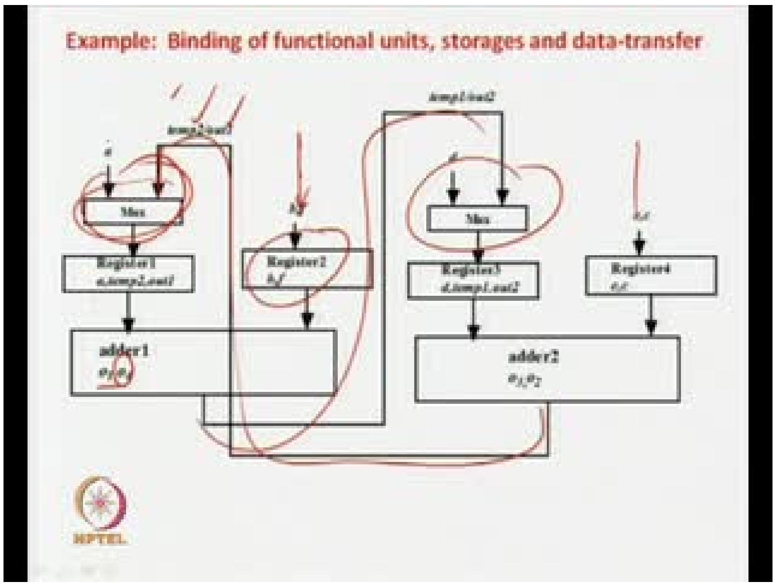
There and then you have find out that which 1 is actually giving you the minimum possible area so that is the core idea of the algorithm that will be developing to solve the problem so in this case we have put some because we are not trying at present you are not trying some algorithm we are just thinking on are range as a human being like just looking at the diagram and trying to find out which allocation which binding give you a less area kind of a thing so we are actually putting temp 2 over here temp 1 over here that was swapping and all those things there but, you have remembered that in a general case all these things may be you can again having a permutation and combination like we could have b and c over here and e and f over here we could also made b and c over so we could have done this way also. So we could also put out instead of this we could also put out 1 here also you could have put so in any type of permutation and combination are possible so all things are actually the algorithm which will try to do the best binding for us we will try to have to see all the permutation combinations possible among this then they have to pick up which is having the least area so you can understand how pick up the problem is so if you recall i always.

Telling that all the understand that all the v l s i design problem has design problems are harder and complex so very quickly we see that this is also n p complete problem that is which will not any good polynomial type of we are not doing in polynomial tonal time algorithm which is solve the problem so you can see that you try all permutation combination that is o 1 o 4 o 3 o 2 then the all different permutation combination like a here temp 2 here out here out 1 here and may be other combination so all possible combinations of both registered as well as operation binding and has to be tried and then you have to find out for each of this binding what is the area required in case of interconnection and then you have to find out which one it will be the least area of which case the area will be the least that you have to consider as the best and you have to say that this binding is taking the minimum interconnect area and this is the best binding what you can understand by permutation and combination there can be so many different the number of different type of permutation possible will be very high may be exponential or may be much more than exponential number of different permutation will be possible in this case and in as general case there are the hundreds of registers and hundreds of functional units so.

This may grow up like anything and we will not get any feasible number feasible solution in a very reasonable amount of type and again for each of this shed the binding what you have do you find out the area that is again another cause that is coming in to picture so what you have to do so you have to see that our state is first try each of the permutation and combination possible so the combinations are very very high in number and for each combination you have to find out what the is the area over it for the interconnect then you have to find out which is the least and you have to do so you can find out exp1ntial problem and top of the there is another some module that is coming in to the picture that is actually we have to find out the area over it because of the multiplexing arrangement for the interconnect for each of them so that is why again you is in practical solve this using what you call a exponential algorithm or what do you call a exact algorithm so what you have do we generally go on for about heuristic to solving the problem and then we can find out that this binding is the near optimal kind of binding may not be best one the time required will be very reasonably low so again like our scheduling case is also see different algorithms today so which.

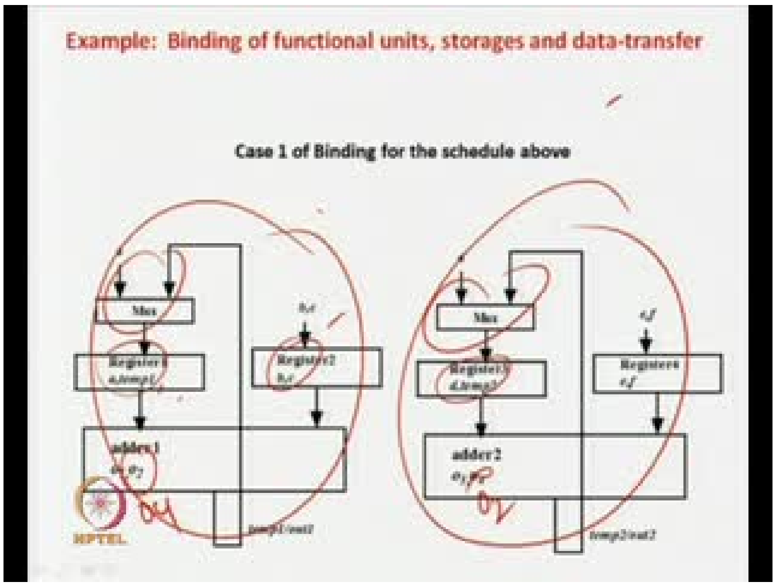
Will actually a binding solution but, that may not be the best now let us see over here so in this if seen so o 1 o 4 o 3 o 2 and this is the binding which is written a temp to out 1 d n f d temp to out 2. Temp 1 out 2 and e and c so first case you see a will be connected to the this register through input 4 and then o 1 so a plus b it will generate out 1 so if now here actually we are using o 2 so o 2 operates on temp 1 plus c so in this case this interconnect will going over here it will not go back to here as in the previous case. Now so in step 1we have generated a 1 plus b and we generate temp 1 so this temp there using temp 1 the register 3 ok. And then in this case if you see it is adding d plus e actually temp 1 temp 2 this is the temp 2 that is been now come coming over here because o 4 is there so o 4 is not o 4 is adding temp 2 plus f over here so again.

(Refer Slide Time: 28:50)



The output of this adder will becoming and this temp 2 is binded to register 1 so this i mean to a marks so you require a marks over here as in the previous case. In the time step 2 what is happening we are adding temp and f and then you are finding generating the result which is actually out 2 so again i mean out is binded to third 1 so in this case you have to have the same output of the same adder which actually feeding over this mark where this marks will register 3 and out 2 is coming over here and in this case also you can think that o 2 is actually and in the second step taking temp 1 and it is adding with c and it is generating the value of out 1 in binding into register number 1 so this is interconnection that is the term over here so as in the previous case you require to multiplexer because e and c are different control steps are directing to the input bus in the first step you will get e and second step you will get c and that can be binded register for in this case also it is b n f so first type be the is b second type variables are binded in to register 2 a 2 different.

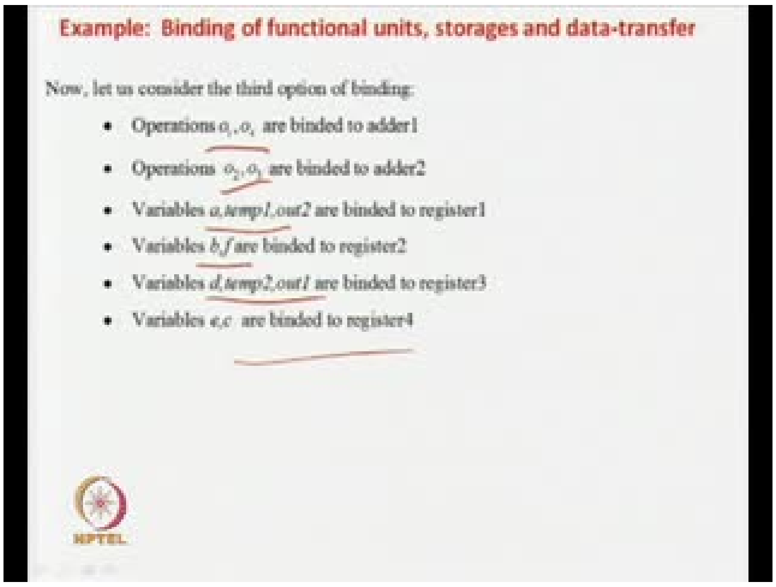
(Refer Slide Time: 29:39)



Example: Binding of functional units, storages and data-transfer

Case 1 of Binding for the schedule above

(Refer Slide Time: 29:47)



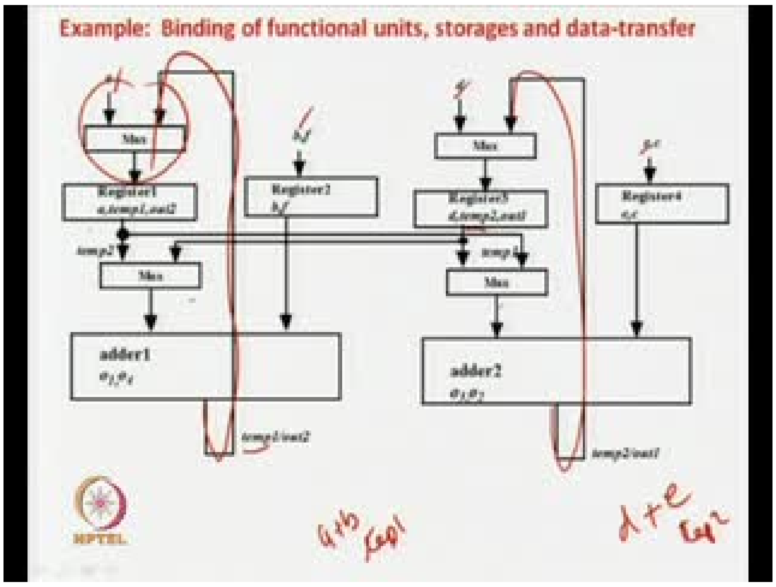
Example: Binding of functional units, storages and data-transfer

Now, let us consider the third option of binding:

- Operations o_1, o_4 are binded to adder1
- Operations o_2, o_3 are binded to adder2
- Variables *a,temp1,out2* are binded to register1
- Variables *b,f* are binded to register2
- Variables *d,temp2,out1* are binded to register3
- Variables *e,c* are binded to register4

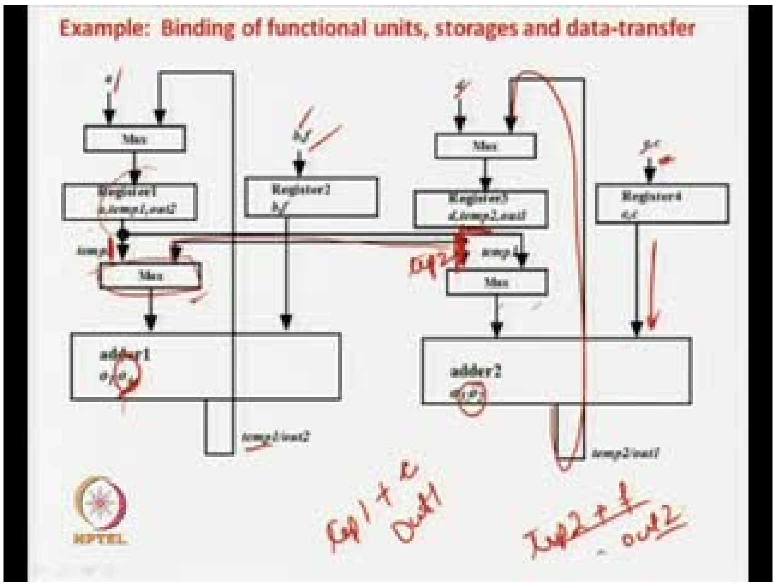
Control step so you do not requires to do that but here you require a multiplexer because you are having a from the input bus and in the second and third time step is it is temp two and out 1 so you and this coming out from output and adder number 2 but, to solve two ports different ports so you require a multiplexer and for the similarly, is in here also you require a multiplex. So 2 multiplexers are required over here but, you can think that the number of multiplexer is the same but, you can see that these are some (( )) where you so there may be some more area required because of the routing of this net so this option is actually taking some more area compare to the previous one in terms of interconnect area and it will bit because in this case you can see this was a very neat and local type of connection this was one local connection area this was local connection so the interconnect area would have been less compare to the so, the number of multiplexer equal to 2 and 2 and which is which is similar now let us take the third of option. And then see what happens in the case of interconnect area so we are again taken the o 1 and o 4 o 2 and o 3 actually same thing we are doing so o 1 and o 4 is binded over here o 3 and o 2 are binded over here.

(Refer Slide Time: 30:22)


Example: Binding of functional units, storages and data-transfer

So that is the end thing of the last example and now we have seen that a temp and out 2 and binded to register 1 temp 1 out 2 are binded to this 1 in this b n f that is same d temp 2 and out 1 are added over here d temp 2 and out 1 are binded over here and e and c are binded over here. o 3 binded here now let us see what is the area requirement now let us see how it will do you can see over here that i require 2 multiplex additional multiplexer over here so how we come over of let us see so in time step 1 so what is required you required to do a plus b and in this case you require do d plus e you generate temp 1 you generate temp 2. That is the what required so same multiplexing arrangement will be there so it is taking a here and that is taking b here and what it is doing it generating value of temp 1 so in this case temp 1 binding to register 1 so you have to take this at an your writing it over here so not a problem it is done.

(Refer Slide Time: 31:39)

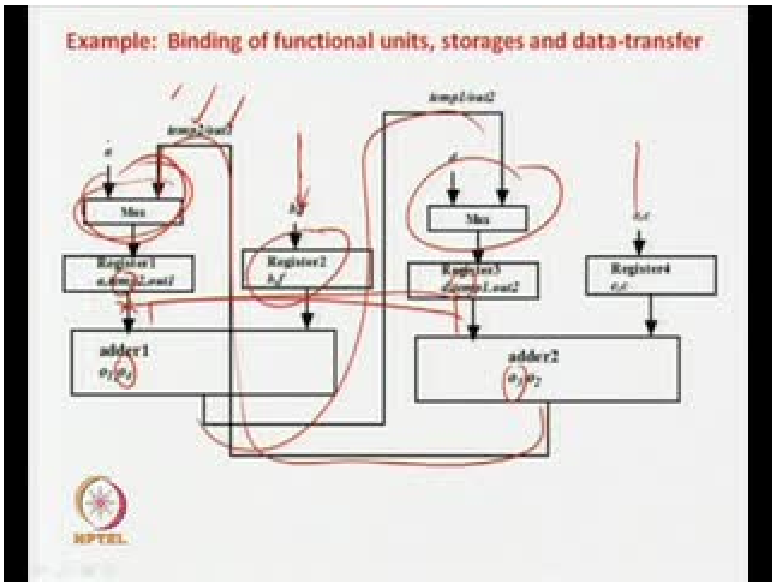


so temp on is done over here right and in this case also you have taking the b plus temp in 2 so temp 2 is binded over here so you take d over here d over here the addition is done the output it is coming over here and temp 2 is binded here and this is connection for this one you require a multiplexer over here because d and output from the register they are coming to the register 3 and here also a and e output at that is the input bus and output of register adder 1 is i mean register number 1 so you require a multiplexer this is the same reason the last 2 example give. Now let us see second control step for which we require 2 more multiplexers here and here so what is the second control step second control step we has doing as temp 1 plus you require c to generate out 1 and temp 2 plus f you generate out 2 now you can see we have done a mistake so here we have temp 1. And what do you require more so temp 1 is available here that is not.

A problem so temp is our is temp 1 is available so this temp 1 so i should not write here temp 2 here i write temp 2 . So now temp 1 is available so temp 1 is available at this port you can understand so this is nothing but, temp 1 in the control step. And what is available at this port so at port if you can think temp 2 is available so temp 2 available at this point and here temp 1 is available so now i should right this temp 1 so temp 1 is available is over here and the time b. Time in temp 1 is a available and here actually for the time temp for the time being let us not see this temp 2 is available at this port now what happen so you have to add temp 2 this is temp 1 you have to add along with this you have to add c . And then you can generate the value out 2 out 1 it will be you get the value of out 1 now you see what is the big problem where actually having o 2 over here so if we are having o 2 over here somehow this temp 1 you have to bring this to this adder c is available.

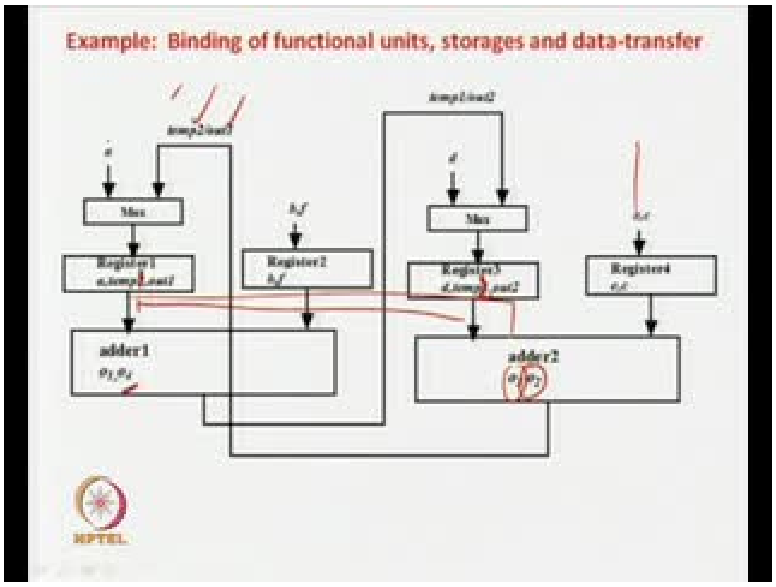
That is not a problem so c will come over here but, now again this temp 1 to bring this adder so we bring this temp 1 form here to here to this also this is the where this is the actually bringing temp 1 . Through this so this is actually temp 1 this temp 1 is brought from this register by this 1 and again you can see o 4 is here so in this case what happen you have to add temp 2 plus f to get an out 2 so f is available over here not a problem but, again this temp is not available in this register temp 1 temp 2 is require for o 4 temp 2 is not available in register 1 so temp 2 this is available here that has to be brought here and you have to bring it over here so there is some cress cross connection over here so you require and a multiplexer over here that is actually making the problem so if you just look at the last figure what is there so o 1 for o 3 not a problem so what was there if you remember so in this case.

(Refer Slide Time: 34:03)



Example: Binding of functional units, storages and data-transfer

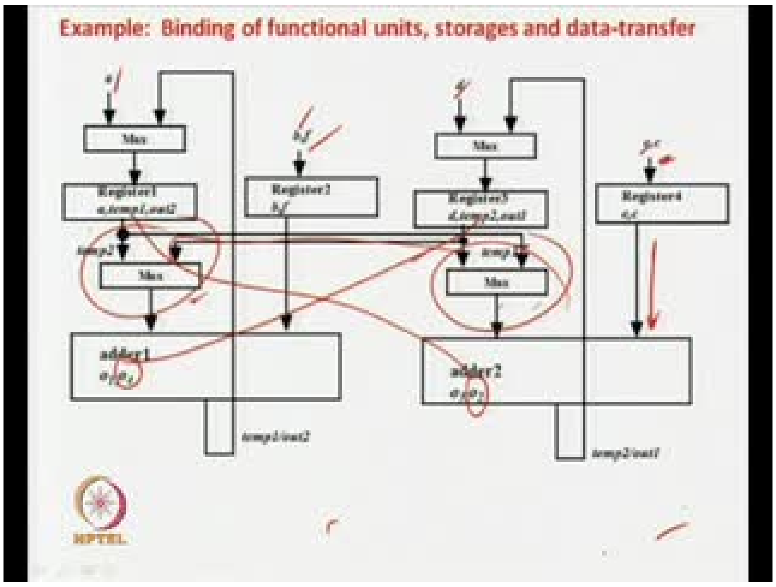
(Refer Slide Time: 34:46)



Example: Binding of functional units, storages and data-transfer

Was available over here o 4 is here o 3 is here so o 4 requires temp 2 plus f o 3 requires temp 1 plus c so temp 1 was available in register 3 and temp 2 was available in register 1 so we do not require any kind of cress cross here or here that are not required but, now what you have change in the next example that is temp 1 we kept a temp 1 here allocated over here and we have made temp 2 so that is critical problem you see o 4 requires temp 1 which is available in register o 4 equals temp 2 o 4 require temp 2 which available in register so you again bring it over here. And you see o 2 requires temp 1 which is available in register number 1 so that again has to be brought over here so but, in this present case this is not required because temp 1 and temp 2 are store very near to o 4 and this one will store very near to this 1 so if you require any multiplexing over here but, in this case as that is that has been done so you see so what you have to do so o 3 o 3.

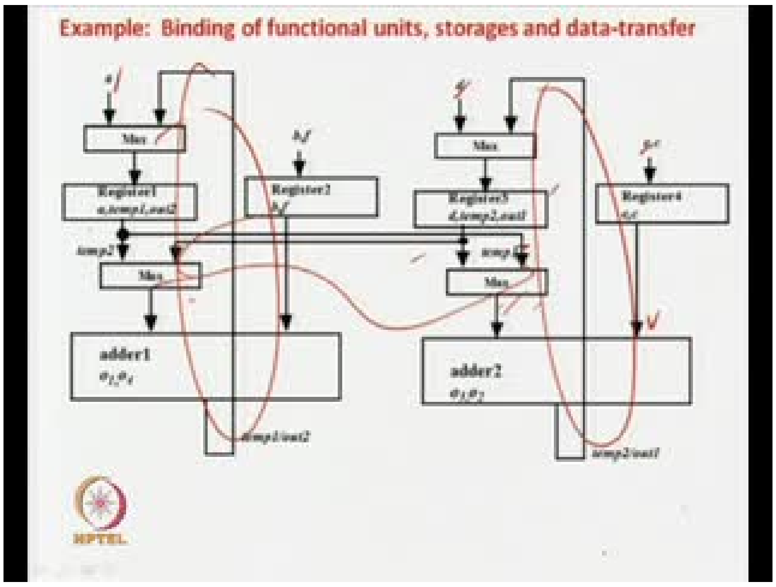
(Refer Slide Time: 35:02)



Example: Binding of functional units, storages and data-transfer

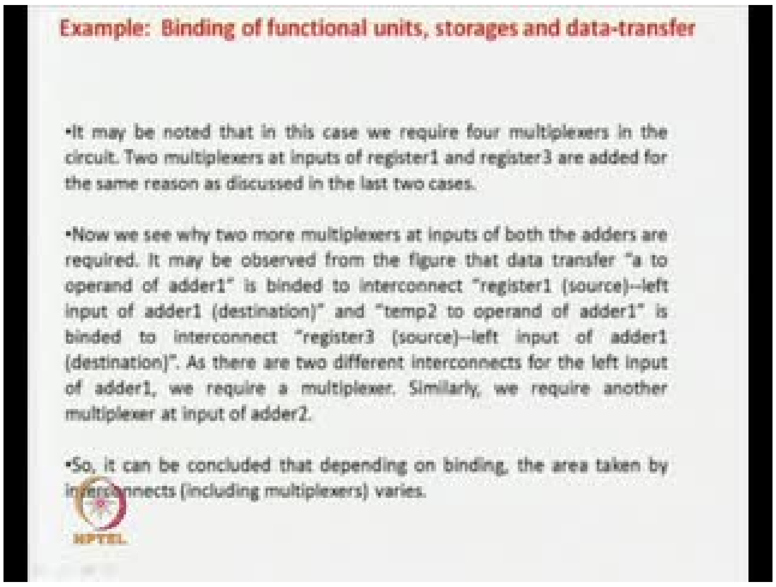
O 3 actually o 2 require temp 1 which has been brought by this pin and o 4 actually require o 2 which has to be again brought by this 1 so you actually require two additional multiplexer to do this kind of interconnection binding so if you take this type of a combination so multiplexing arrangement requires 1 2 3 4 multiplex are required and along with that you require see the complexity of wirings over here so that means what in natural for this type of a binding you get an area over here which is much higher than the first 2 cases so what i mean here we are taking some arbiter some small arbiter example and we have tried the different combinations because that how it has differs but, but i wanted to tell you is that in real case when you have to solve the binding problems so we have to try with there are the different permutation and combinations for these what do you call this register storage unit binding as well as adder binding kind of that is the adder of the function binding kind of a case now we have to find out the most optimal 1 so that interconnect binding.

(Refer Slide Time: 36:04)



Example: Binding of functional units, storages and data-transfer

(Refer Slide Time: 36:19)



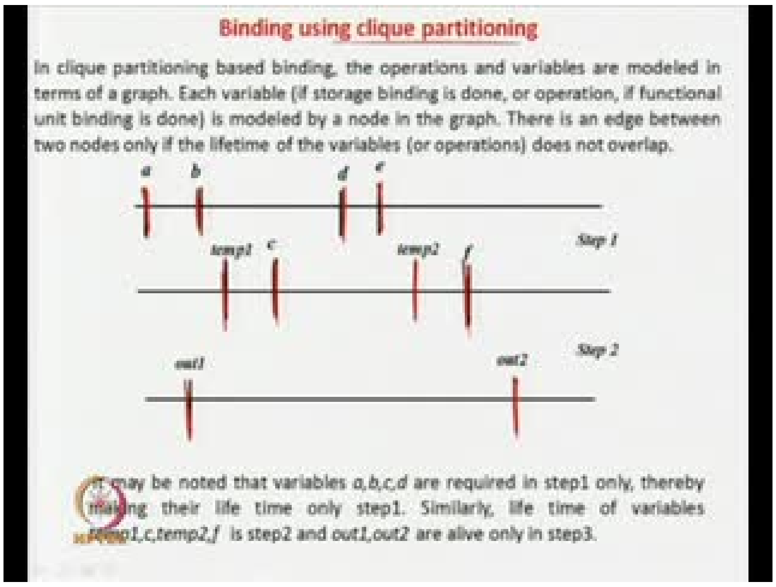
Example: Binding of functional units, storages and data-transfer

•It may be noted that in this case we require four multiplexers in the circuit. Two multiplexers at inputs of register1 and register3 are added for the same reason as discussed in the last two cases.

•Now we see why two more multiplexers at inputs of both the adders are required. It may be observed from the figure that data transfer "a to operand of adder1" is binded to interconnect "register1 (source)--left input of adder1 (destination)" and "temp2 to operand of adder1" is binded to interconnect "register3 (source)--left input of adder1 (destination)". As there are two different interconnects for the left input of adder1, we require a multiplexer. Similarly, we require another multiplexer at input of adder2.

•So, it can be concluded that depending on binding, the area taken by interconnects (including multiplexers) varies.

(Refer Slide Time: 36:52)



Binding using clique partitioning

In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

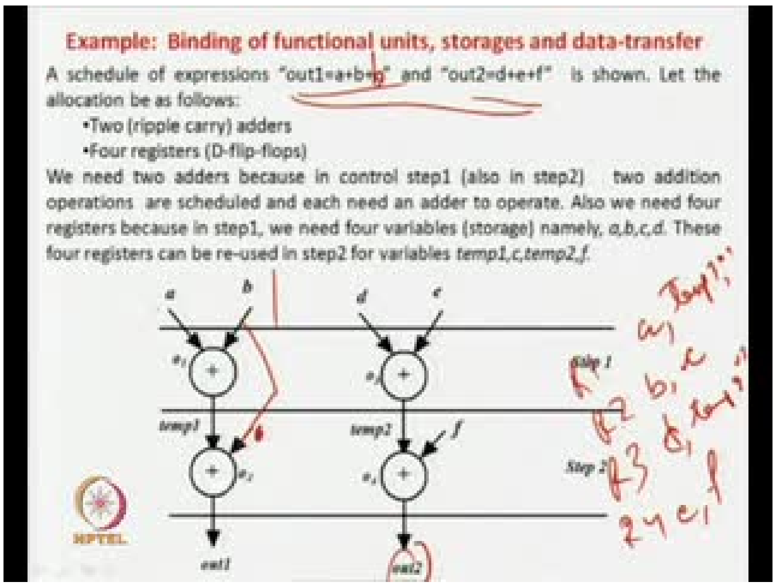
It may be noted that variables *a,b,c,d* are required in step1 only, thereby making their life time only step1. Similarly, life time of variables *temp1,c,temp2,f* is step2 and *out1,out2* are alive only in step3.

In terms of this wire and this multiplexer and the multiplexer and this type of wire is you can see that has wire is and that has to minimal so that is why if you now if find out the algorithm then which will do in example which is do it existed search for you and then tell you which is best 1 is going to be a very very difficult problem then we will take amount of time so you will not be able to do it so we will some kind of heuristics first will try to put formally that is this is the n p complete problem that is very difficult it is not well known polynomial time solve the problem then we will find out some heuristics to do that so. I mean whatever i told you that why do you require 2 multiplex in 1 case and why do you require multiplexer in the other case and so for written over in this slides so you can go through this slide to find out whatever i have told you that why this is they actually taking 4 the other 2 cases where have you taking 2 and so for so the whatever i told you are and whatever i explained you in the over slide so you can go through this now we are going through algorithm so till now again we are reemphasis the problem of binding so in the first.

Lecture of this module we have given you the idea of scheduling allocation and binding problems now again in this case you are again reemphasis the binding problem and we have shown that for different types of what you say different type of storage you need into and function unit binding to make a different area in terms of interconnect binding so you have to take the best 1 way the interconnect binding is least. So that is will be the solution now we are going to see algorithm which can automatically do it for you. So again as in seen in last case scheduling case like in the scheduled case we have mapped and problem to 1 you mapped or problem to a integer linear program zero 1 i help to know this n p complete problem so here we are trying try to match our problem to map out problem to very well known np complete problem the clique partitioning to so what do mean by clique partitioning i mean if there has to under graduate graph theory because course even in graph you have to find out sub graph in which all the nodes from each node to each node there is a path. So that is actually called a clique so if do not take very formal definition the formal definition go to under graduate that you recourse what the basic idea is.

Graph if you can that is call the max if you go for the maximally so given the graph can you find out a sub graph which is taking the maximal sub graph that sub graph which is having the same number of nodes possible from the graph thus that from the each node to each node of the sub graph that is the clique there is a path or there is the edge. So let us now try to do this so in this case what we do so here we have actually Joe this type of control step and then we draw the variables as some vertical line so, what is the vertical line vertical lines are actually saying that how long is the life time so a is require only 4 time is step 1 so its life time only in step 1 similarly, for b d e temp 1 arising after this step first initiation first initial combination is done that is a plus b temp 1 is generator c is generator and d plus temp 2 and f is the input so in control step after control step 1 we required temp 1 c temp twelve so its life time is actually after time step 1.

(Refer Slide Time: 39:24)



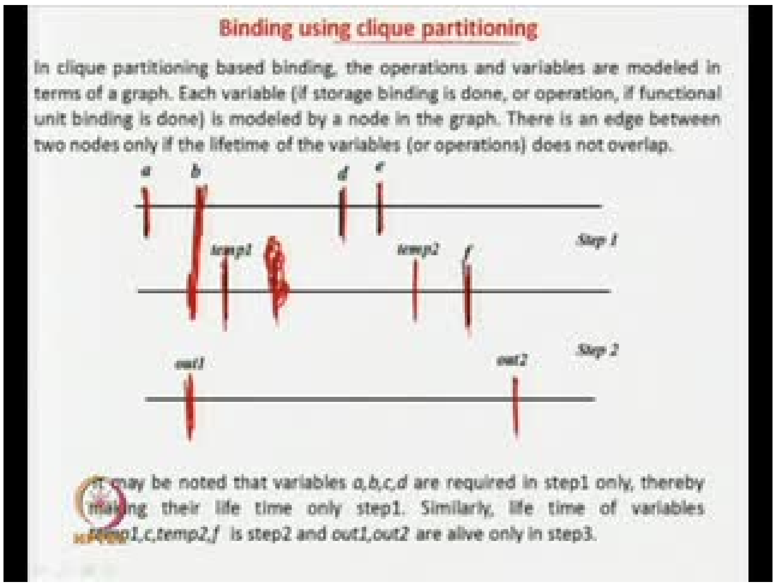
Example: Binding of functional units, storages and data-transfer

A schedule of expressions "out1=a+b+c" and "out2=d+e+f" is shown. Let the allocation be as follows:

- Two (ripple carry) adders
- Four registers (D-flip-flops)

We need two adders because in control step1 (also in step2) two addition operations are scheduled and each need an adder to operate. Also we need four registers because in step1, we need four variables (storage) namely, *a,b,c,d*. These four registers can be re-used in step2 for variables *temp1,c,temp2,f*.

(Refer Slide Time: 39:43)

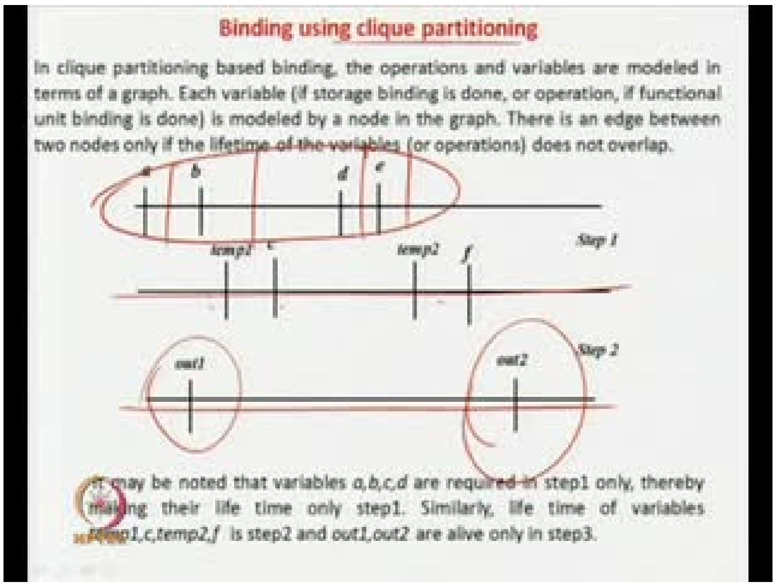

Binding using clique partitioning

In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

It may be noted that variables *a,b,c,d* are required in step1 only, thereby making their life time only step1. Similarly, life time of variables *temp1,c,temp2,f* is step2 and *out1,out2* are alive only in step3.

So after time step 2 generate out 1 and out so when out 1 are generated you do not required temp 1 c or temp 2 f or rather of this so these actually horizon line are representing your life time see for example, if you have something like say just i mean go back say if you could have thought over something like so if you should if you what if you have said that my expression is a plus b plus b so let us think about this a plus b plus b kind of a thing so would have written over this that is a plus b temp 1 then again add with b so in this case b is requires not only control step one but, also in control step two so in this case what would have been you line diagram so in this case your line diagram will be something like this case b is also required over c would not have in this case so b is required both in time temp 1 and 2 and then you actually generate temp 1 if is 5 so first temp 1 will be there then temp 1 again this b is used to generate output so that is b required in 2 time step so.

(Refer Slide Time: 40:21)



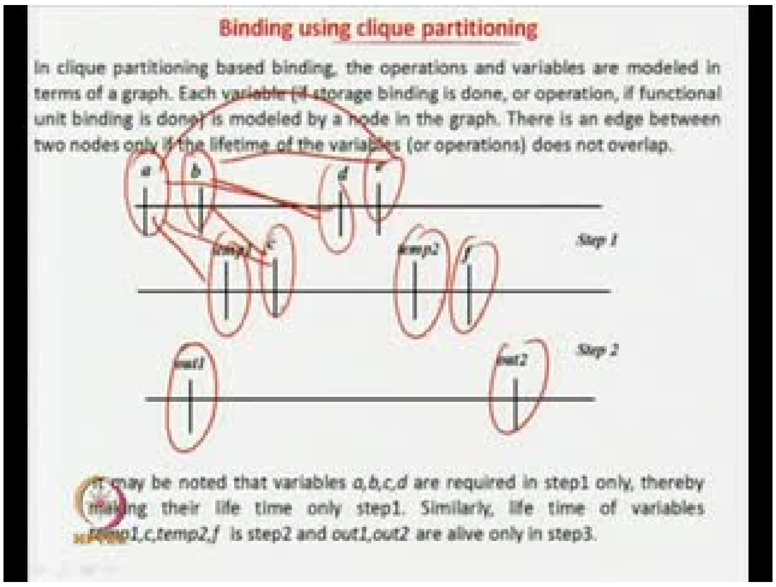
Binding using clique partitioning

In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

a
b
d
e
temp1
c
temp2
f
Step 1
out1
out2
Step 2

It may be noted that variables a,b,c,d are required in step1 only, thereby making their life time only step1. Similarly, life time of variables temp1,c,temp2,f is step2 and out1,out2 are alive only in step3.

(Refer Slide Time: 40:57)



Binding using clique partitioning

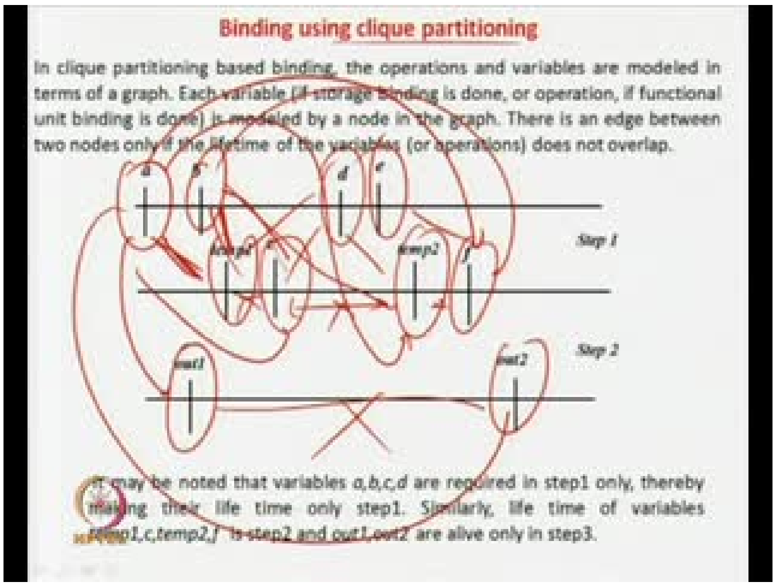
In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

a
b
d
e
temp1
c
temp2
f
Step 1
out1
out2
Step 2

It may be noted that variables a,b,c,d are required in step1 only, thereby making their life time only step1. Similarly, life time of variables temp1,c,temp2,f is step2 and out1,out2 are alive only in step3.

So what in the other way this vertical lines are representing how long how much how many control steps this value will be alive so this is the what is the case now you think how to mapping to quick partition into a problem so you know that a and b cannot share a register because they are all allowing control step 1 similarly, temp 1 c temp 2 f all required in the step so you cannot share their live and similarly, out this 1 up to 1 and up to 2 so they control cannot share a variable they cannot share a register because they all are alive in this out 1 and out 2 are alive in a common time steps but, a and temp 1 can share a register b and c can share a register that is you can think that 4 register are required over here as already mentioned but, this 4 can be shared by any 1 outputs but, again among the temp 1s temp 2 they cannot sharing in between similarly, out 1 and out 2 cannot share anything in between that is what is the now what you have to map to a quick problem so we have we make notes for all a d b e temp this 1 this 1 and this 1 and this 1.

(Refer Slide Time: 41:22)



Binding using clique partitioning

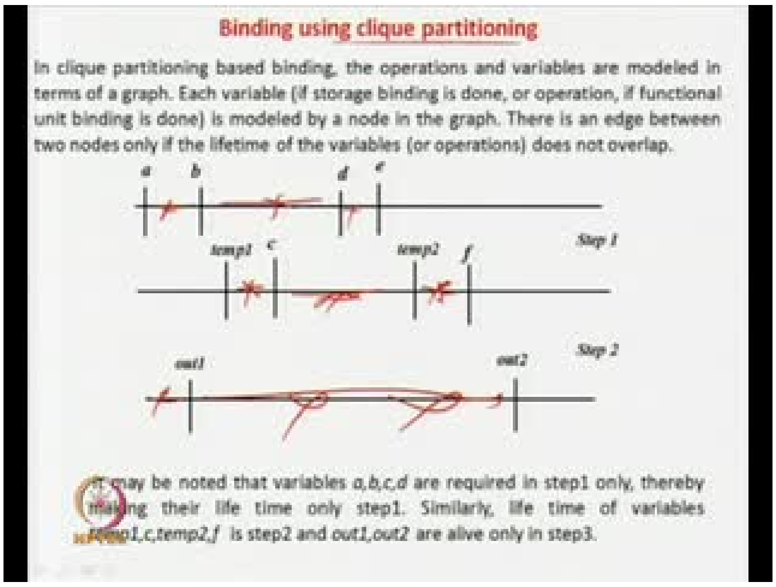
In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

a
b
d
e
temp1
c
temp2
f
Step 1
out1
out2
Step 2

It may be noted that variables *a,b,c,d* are required in step1 only, thereby making their life time only step1. Similarly, life time of variables *temp1,c,temp2,f* is step2 and *out1,out2* are alive only in step3.

Now you say that temp 1 share that register that you may join that similarly, this 1 can share register and this 1 and this 1 similarly, a and temp 1 can share similarly, a and c can share a and e can share again a and d can also share similarly, now if you look at d and all these stuffs so again d can share between of these so we have this in between now you also know that b can share b cannot share in between in this 2 so you do not have any ages so over here so again so this a can share with the this a cannot share with this thing there will be no line from this 1 to this 1 only we have some lines from here to here because as a and b cannot share a variable so there is no ages in between in this similarly, a and d cannot share so there will be no ages in between similarly, d and a cannot share registers so there is no need connection between similarly, a and b there is no connection but, a and temp 1 can share so there will be hyphen b and temp 1 can share so there will be a age a and c b and c can share

(Refer Slide Time: 42:39)



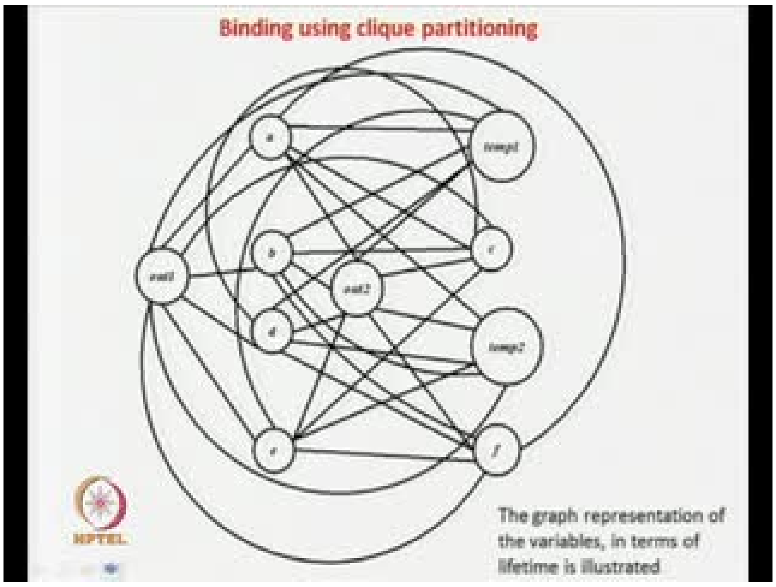
Binding using clique partitioning

In clique partitioning based binding, the operations and variables are modeled in terms of a graph. Each variable (if storage binding is done, or operation, if functional unit binding is done) is modeled by a node in the graph. There is an edge between two nodes only if the lifetime of the variables (or operations) does not overlap.

a
b
d
e
Step 1
temp1
c
temp2
f
Step 2
out1
out2

It may be noted that variables a,b,c,d are required in step1 only, thereby making their life time only step1. Similarly, life time of variables temp1,c,temp2,f is step2 and out1,out2 are alive only in step3.

(Refer Slide Time: 42:50)



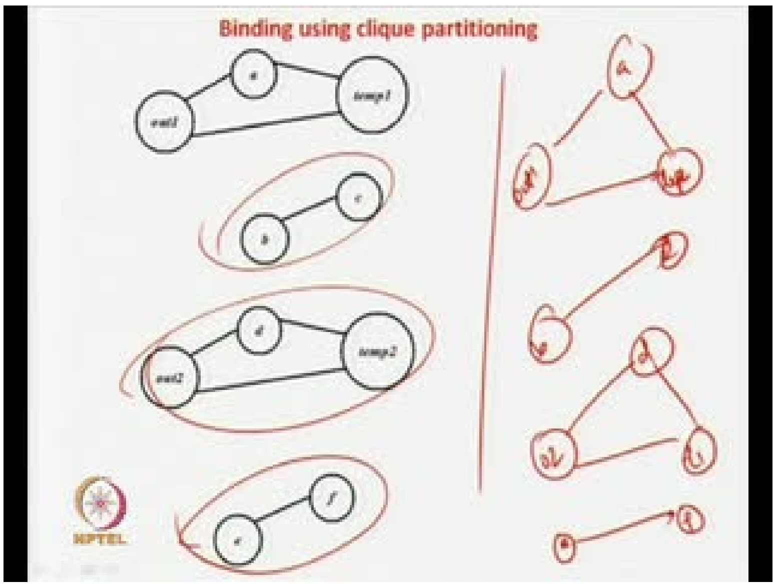
Binding using clique partitioning

The graph representation of the variables, in terms of lifetime is illustrated

So there will be an age b and temp 1 can share so there will be a age b and f can share so there will be am age similarly, for this cases there will be ages so for but, there will be no ages in between similarly, a and out 1 can share an age share a variable register so there will be a array ageing between a and out can also share something so there will be no age there will be no age between this similarly, there will be no ages in between this .so in other words make it a big need so if you can see so there be ages between all the nodes and accepting there will be no ages from this 2 no ages in between no ages in from this 1 accept in this there will be ages in all between now this is what your graph will look like see a b c d there is no ages in between there is no ages but, from out 1 you have connection to a b c d then from out 1 you have to temp c and this 1 so this is what is your graph.

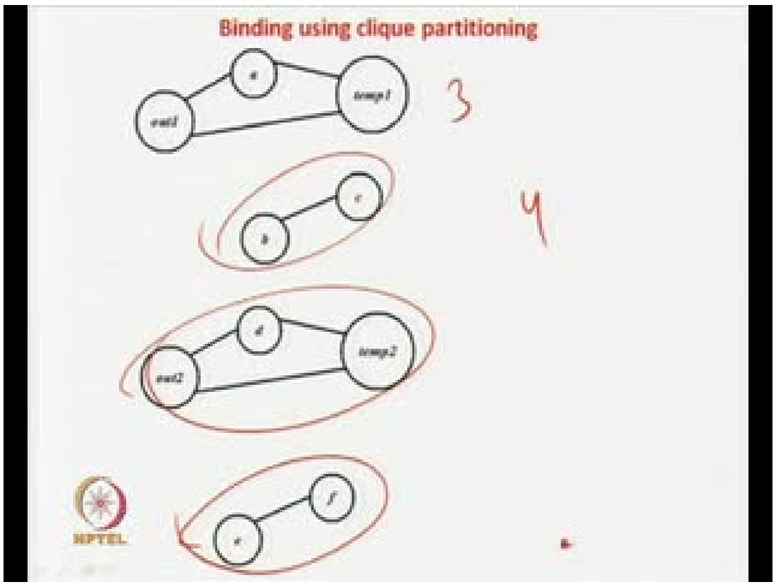
So your graph connection is having between this 2 if and only if they do not life style or they can be shared a now what you have to do now clip partition will tell you how you divided into maximum clips that is you have divided into several some maximum size sub graph such that they are the clip the clip means so from this node you can go here from this node you can go here from this node you can go here so from each node you can go to any other node all the other node in the graph so that is actually a clip now in this case we have to find out the largest possible sub graph of this so directly now how it is actually solving your problem how it is actually solving your binding problem because say we find out if you can solve the problem that is largest size sub graph from where you can reach your node.

(Refer Slide Time: 43:53)

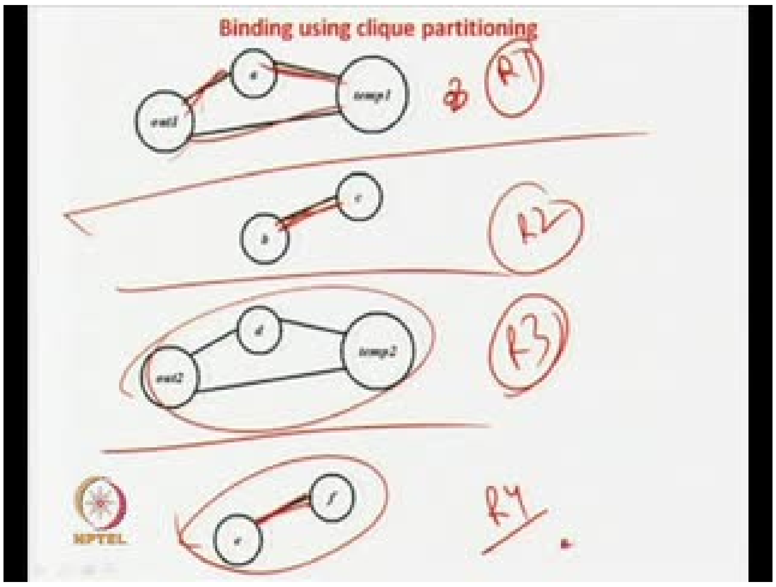


So actually you are getting 1 solution so that can be multiple solution to this what is mean by multiple solution? solution that can be multiple different permutation and combinations of this nodes like For example here it is finding out 1 and so this 1 b and c 1 clip this is 1 clip so you can also find out the another possibility will be that is a in this case it will be say b in this case it will be say a out 1 draw in another can solution like a this out 1 in this case instead of temp 1 you can have temp 2 and you can have this 1 so let this be b c in this case then it will be out 2 in this case it till be d in this case temp 1 joining and in this case e and f so this is one another solution so there can be lot of the solution for this clip there is a unique solution there is 1 solution but, what you mean by maximum clicks? That is maximum number of nodes possible in a sub graph is 3.

(Refer Slide Time: 44:56)


Binding using clique partitioning

(Refer Slide Time: 45:20)

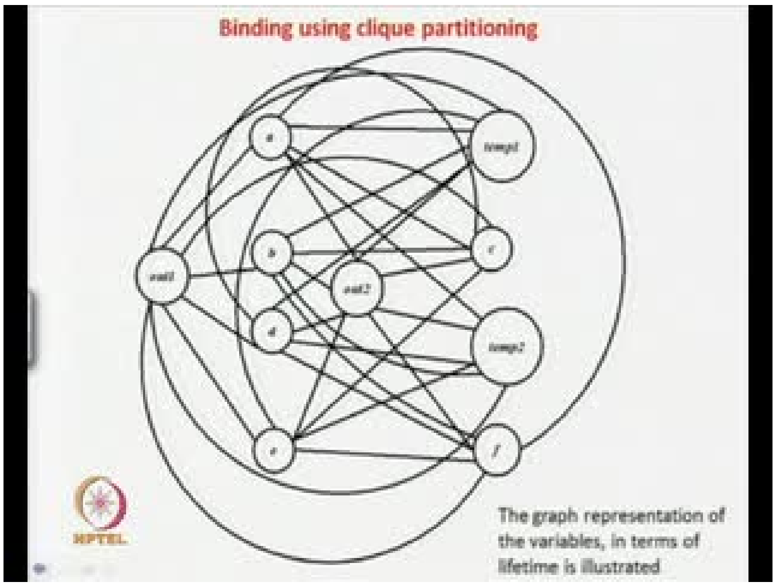

Binding using clique partitioning

So we cannot have another sub graph having 4 node that you can reach node to from here we can go here you can go hero and you can go here so what we have found out we have found out there is a maximum of this 3 nodes can be possible in a sub graph that is actually where exactly that is going from each node 1 another now the algorithm which can find out given a graph like this any arbitrary graph we can find out where is a maximum click of 3 whether what are the maximum clicks that exist in the graph that n p control problem that is very well knows n p complete problem clip partition that whether given graph with k n nodes k maximum clicks possible is whether k clips possible is also n p control problem there is n number of Solution for this the most these n p completeness and definition and keeping a bit super facial or you can call not formal quote on very form in the lecture because this is the course on cad so and just dealing you what is the problem and how it is map to this 1 but, for very formal motion and how the click max click problem is a a n p control problem and the other wise it is.

So we required to go for any theoretical complete science book or any complex analysis book any way we have to find out then there is a idea here than so the any algorithm that gives a graph of this can tell you that there is only 3 nodes possible in max click is n p complete we do not have any permanent algorithm for them so say i have a very long time possible with me so what i can do is that i can pump in as many times as possible and then i can say that this 1 is i can available how does it solve binding problem? the answer is that it is saying that this is the number of clips possible.

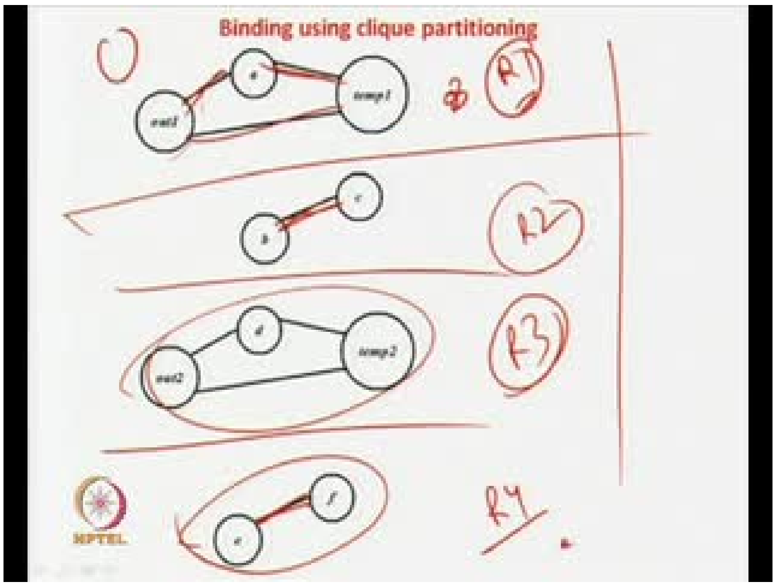
Now you say that this is for register 1 this 1 for register 2 this is for register 3 and this is for register 4 how is that possible? because you see we have said that we have connection between these 2 if and only if the variable of output 1 variable of this life time of this variable this variable and this variable do not overlap if the variable life time between a and b overlap we do not put age over here so there is no age between a and b so that is why we do not have any connection in between a and b that is the reason why They will never come in 1 click so we have an age only in between the variables whose lifetime do not overlap that means this is finding one click right that means from each age node to each node we can traverse that is life time of all of them are not common that is why we can easily have one register which can share this similarly, they can have the age i mean what their lifetime do not overlap.

(Refer Slide Time: 47:52)


Binding using clique partitioning

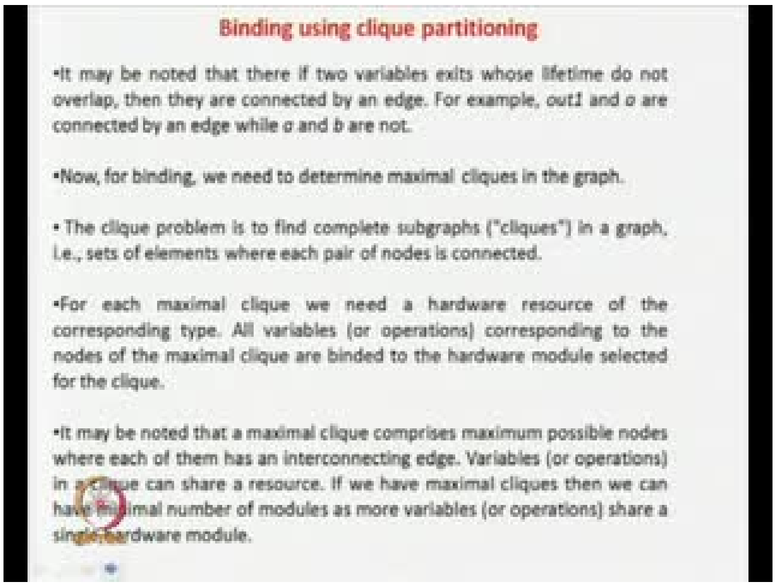
The graph representation of the variables, in terms of lifetime is illustrated

(Refer Slide Time: 47:59)


Binding using clique partitioning

So we can have 1 register here similarly, how to be an this 1 and this 1 to go for which node to a each node that means there is an age that means their life time not overlap so there is a register for that similarly, for the last case there is age between e and f that is it so there lifestyle not overlap you can have register no 4 so that is how if you can have exponential time algorithm and there are lots of heuristics algorithm sub click partitioning problem so in this case you get the answer now you saw that the problem therefore, the clip partitioning problem is a n p complete problem there is no polynomial time solve version for this but, then what you have then what is there idea is here that you can look to any theoretical computer science book here are lot of heuristics to solve this quick partition problem that is they will take much less than exponential time they will try to generate maximum size clips but, here the solution may not be optimal always that is many time be may not be getting the largest possible clips out of it.

(Refer Slide Time: 48:59)



Binding using clique partitioning

•It may be noted that there if two variables exits whose lifetime do not overlap, then they are connected by an edge. For example, *out1* and *a* are connected by an edge while *a* and *b* are not.

•Now, for binding, we need to determine maximal cliques in the graph.

• The clique problem is to find complete subgraphs ("cliques") in a graph, i.e., sets of elements where each pair of nodes is connected.

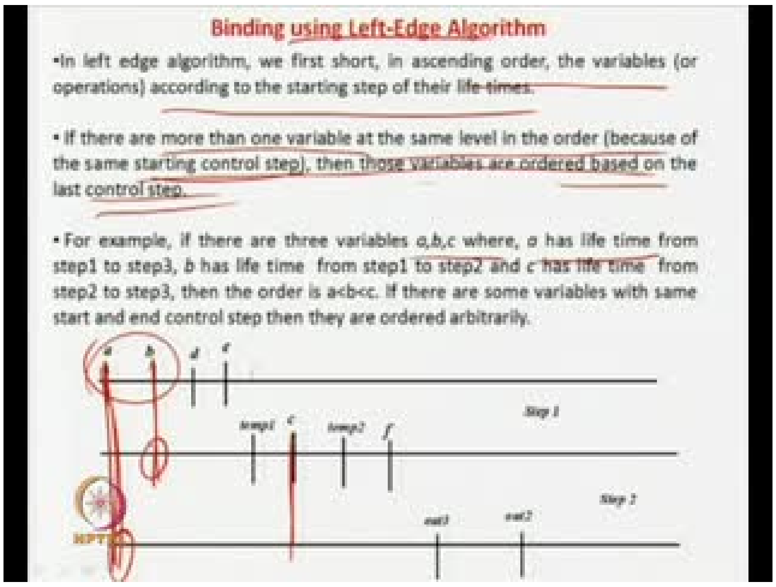
•For each maximal clique we need a hardware resource of the corresponding type. All variables (or operations) corresponding to the nodes of the maximal clique are binded to the hardware module selected for the clique.

•It may be noted that a maximal clique comprises maximum possible nodes where each of them has an interconnecting edge. Variables (or operations) in a clique can share a resource. If we have maximal cliques then we can have minimal number of modules as more variables (or operations) share a single hardware module.

So what you can get here is you can get some clips out of it that is sizes of the clips may not be maximum if that is not the case now the solution is optimal wise because say the maximum clips 4 nodes could have been added to 1 actually then 4 variables could have been shared registers 1 but, the solution will given in very near optimum that is it may say that will give you some 3 nodes which can be possible in a click now this 3 shared by the stuff 1 but in the true case or in the best case 4 variable could have been shared so that is sometimes you may get such types of non optimum solutions so but, any way but, that heuristics algorithm will be much faster than the exponential time algorithm for solving the clips algorithm partitioning problem.

So i am not going to tell you about the heuristics that are available for the this clip partitioning problem because they are very well known in computer science so we can see any heuristics which is available for quick partitioning and any of them size so what you have done in this class is we mapped the well known problem of clips available in the binding problem to clip and then we have shown that it is a n p control problem and there are very largest well known heuristics solve the clip partitioning problem so we can apply anyone of them at least all the variable in the scenario click and share a register but, the taken that problem is that if a solution is sub optimal in most case of heuristics then you may not have all maximum possible nodes actual possible variables in a clip for were in the place of n variable could have been shared where registers and minus case variable should be shared by the register that is actually will be happening so whatever i told you now in my lecture is actually written over in this slide that how the clip partitioning problem is or how the scheduling problem is by binding problem is mapped to the clip partitioning problem and what are the optimization and what are the max clips etc.

(Refer Slide Time: 50:11)



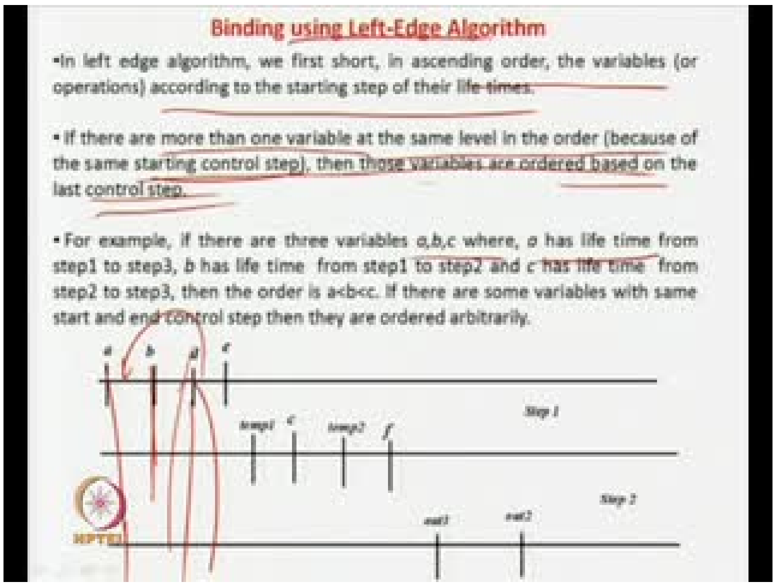
Binding using Left-Edge Algorithm

- In left edge algorithm, we first short, in ascending order, the variables (or operations) according to the starting step of their life times.
- If there are more than one variable at the same level in the order (because of the same starting control step), then those variables are ordered based on the last control step.
- For example, if there are three variables *a,b,c* where, *a* has life time from step1 to step3, *b* has life time from step1 to step2 and *c* has life time from step2 to step3, then the order is a<b<c. If there are some variables with same start and end control step then they are ordered arbitrarily.

So you can just go throw to the slides now we will take another algorithm which is left stage algorithm to again solve the same problem so in this case what we do so in this case we actually first step is that we arrange the light variable in sub in ascending order of their lifetime so a b c d so they are actually arising in the first step so we put it there then terms 1 temp 1 c temp 2 f they come inside the second stage so we put it over here and then finally, out 1 and out 2 we put it over here we first arrange in this 1 and then 1 more thing that if there are more than 1 variable at the same level and then actually then order in the last control step let me just tell about this in the ascending order say there is some variables like this say let us assume that b is like this it saying if they are more than 1 variable at the same level in the order because of the same starting control step then those variables are ordered based on the last control step that is very important so it is saying that we are arranging all the variables in the order of the start of the life cycle like the left stage algorithm is first sort in ascending order the variables according to the starting steps of their life time.

So a b c d and e they all start their life time in 1 cycle so they are arranging in 1 now if the some variable are that this is actually random in 2 line temp 2 then some variable ordering in the value of ordering in the chain like how for example, if there are 3 variables c and c where a has a life time from step 1 to step 3 let us do that so this is steps 1 to steps 3 kind then b as a life time to step 1 and step 2 and c as a life time from step 2 to 2 step 3 so let us not forget this let us think about this so if step 2 to 3 something like this then the order will be a b and c so in this case what happens to the longest 1 this is second 1 and this 1 is third 1 so we will be arranging like this one so we start the arrangement start the order from the life starting of the life time and then in this same level if there are more than one variables then they are order based on the last control state

(Refer Slide Time: 52:41)

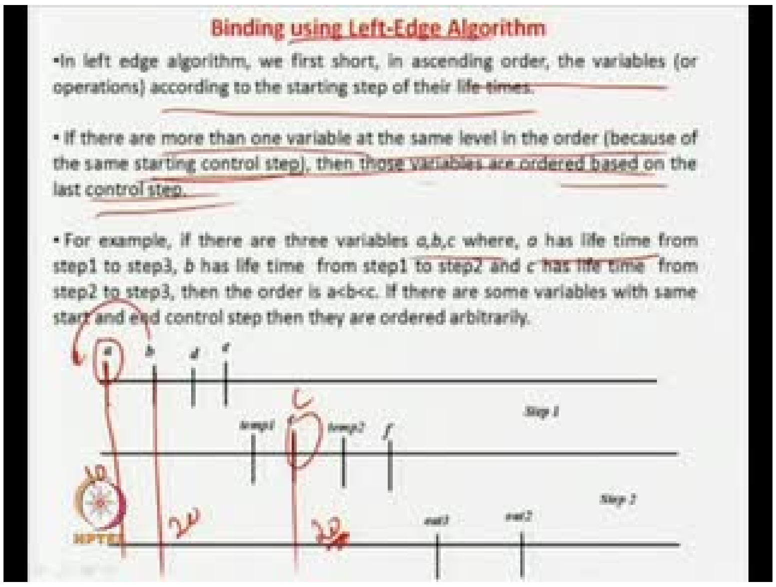


Binding using Left-Edge Algorithm

- In left edge algorithm, we first short, in ascending order, the variables (or operations) according to the starting step of their life times.
- If there are more than one variable at the same level in the order (because of the same starting control step), then those variables are ordered based on the last control step.
- For example, if there are three variables *a,b,c* where, *a* has life time from step1 to step3, *b* has life time from step1 to step2 and *c* has life time from step2 to step3, then the order is a<b<c. If there are some variables with same start and end control step then they are ordered arbitrarily.

So in this case if you see this learning for the maximum this is running for the maximum amount of time that is end life time 3 and in this case 2 so you order a b 4 b that is what being said we say so in other words very simple words you say that we arrange all the variables in you start we arrange all the arrays in some order based on the lifetime so we start with the these starting point of the lifetime and then you arranging and if in the same level say there are this type of staff so we are very long shorter and this may be shorter for so in this case what you do again via longest one will be in the first.

(Refer Slide Time: 53:15)

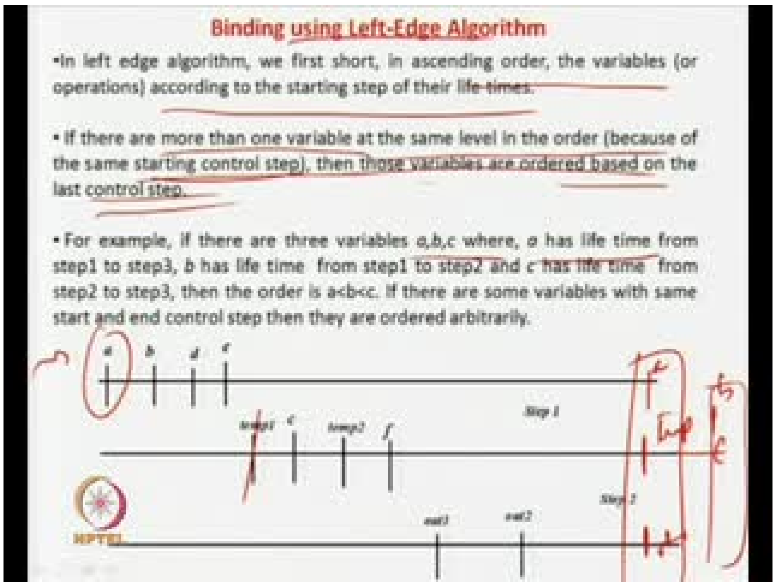


Binding using Left-Edge Algorithm

- In left edge algorithm, we first short, in ascending order, the variables (or operations) according to the starting step of their life times.
- If there are more than one variable at the same level in the order (because of the same starting control step), then those variables are ordered based on the last control step.
- For example, if there are three variables *a,b,c* where, *a* has life time from step1 to step3, *b* has life time from step1 to step2 and *c* has life time from step2 to step3, then the order is a<b<c. If there are some variables with same start and end control step then they are ordered arbitrarily.

So this is going for the maximum length of time so we put it over here then this is going on 2 literal this third this is going to the third taken longest one so you put it over here so maximum is the lifetime so it will come before the 1 this is having the lower life time that is the idea so 1 having the life time of temp starting at point will be higher in the order then the 1 having life time of 3 and also starting from 1 so that is the idea and if c is having lifetime of plenty after a because we start with the starting point of the life time so a is starting at the 1 and running for 10 unit say c is running from 2 and from twenty life times a will be first and c will be next because we start with the life time starting point there is something like a and b but, in this case c will be later

(Refer Slide Time: 53:54)



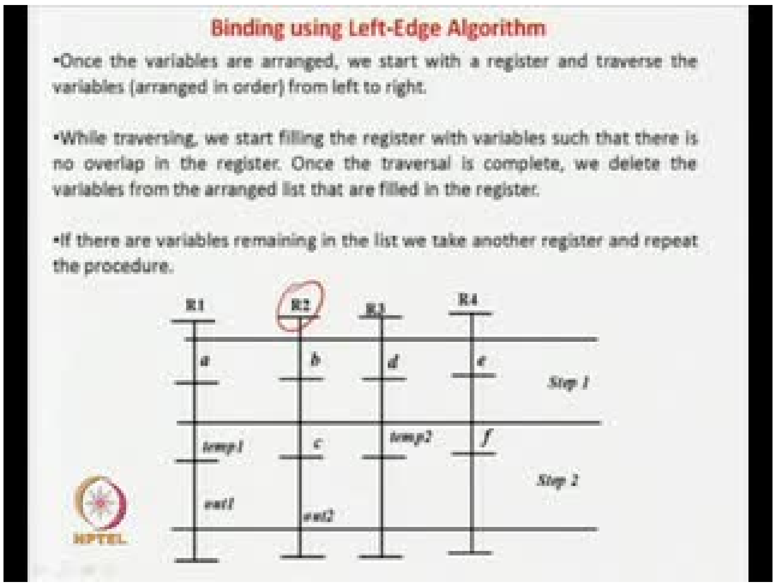
Binding using Left-Edge Algorithm

- In left edge algorithm, we first short, in ascending order, the variables (or operations) according to the starting step of their life times.
- If there are more than one variable at the same level in the order (because of the same starting control step), then those variables are ordered based on the last control step.
- For example, if there are three variables *a,b,c* where, *a* has life time from step1 to step3, *b* has life time from step1 to step2 and *c* has life time from step2 to step3, then the order is a<b<c. If there are some variables with same start and end control step then they are ordered arbitrarily.

c is starting at point 2 but, if there is something like a is running 10 and b is running for twenty then b will come ahead of the a so that is what is being told in this life time now what we have to do now it is very simple now we actually take some buckets and some registers and keep on filling it so this is the very simple algorithm we will see so in this case we have bucket like this empty bucket and then we start from this side then we select a fill it a over.

Here next you go for b cannot filled over here because this part is already allocated so d also cannot be filled over here because this part is already booked e is also filled now we go for temp 1 easily temp 1 can be filled over here because this part of bucket was empty then c temp 2 and f cannot be filled because the part of the bucket is full but, very easily out can be filled over here now this bucket is full now what we do now some iteration is complete that is step 1 step 2 step 3 so now again you take another bucket now a is already gone so b is there so b will be over here d and b cannot be filled because this part of the bucket is full then c temp already gone so you will fill a c over here

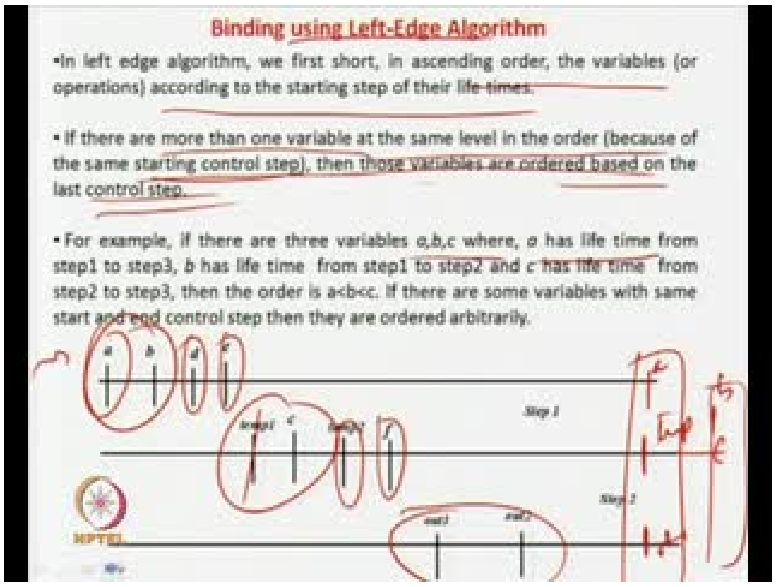
(Refer Slide Time: 54:41)



Binding using Left-Edge Algorithm

- Once the variables are arranged, we start with a register and traverse the variables (arranged in order) from left to right.
- While traversing, we start filling the register with variables such that there is no overlap in the register. Once the traversal is complete, we delete the variables from the arranged list that are filled in the register.
- If there are variables remaining in the list we take another register and repeat the procedure.

(Refer Slide Time: 55:05)

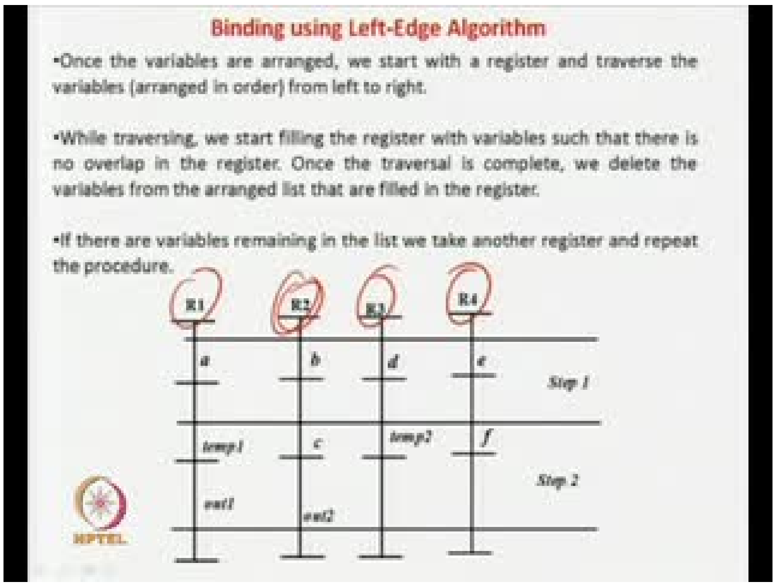


Binding using Left-Edge Algorithm

- In left edge algorithm, we first short, in ascending order, the variables (or operations) according to the starting step of their life times.
- If there are more than one variable at the same level in the order (because of the same starting control step), then those variables are ordered based on the last control step.
- For example, if there are three variables a,b,c where, a has life time from step1 to step3, b has life time from step1 to step2 and c has life time from step2 to step3, then the order is a<b<c. If there are some variables with same start and end control step then they are ordered arbitrarily.

So this will be c and so for and this is how your bucket looks like you start u with a you fill up a then in this bucket i you cannot fill with the a b c d or e b d and e cannot be filled so you can filled only temp 1 can be filled so we have to fill temp 1 over here and then again in this case out 1 will be filled now this bucket is full so you take r 1 r 2 r to be the filled in the first and e cannot be filled c will be filled out will be filled now in this case Now in this case what will going to happen is will be over this 2 as will over in third and 4th bucket you will have 3 and temp 2 and e and f in the 4th bucket in this case you can fill and then you are done.

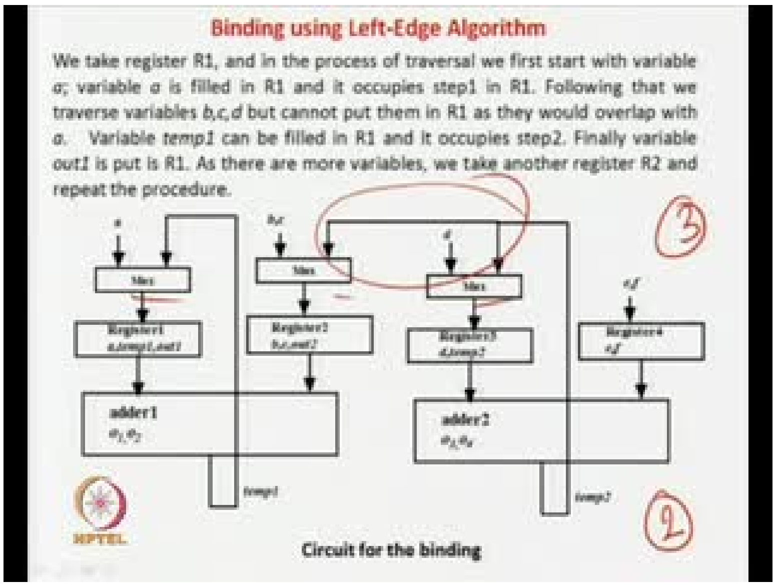
(Refer Slide Time: 55:12)



Binding using Left-Edge Algorithm

- Once the variables are arranged, we start with a register and traverse the variables (arranged in order) from left to right.
- While traversing, we start filling the register with variables such that there is no overlap in the register. Once the traversal is complete, we delete the variables from the arranged list that are filled in the register.
- If there are variables remaining in the list we take another register and repeat the procedure.

(Refer Slide Time: 55:58)



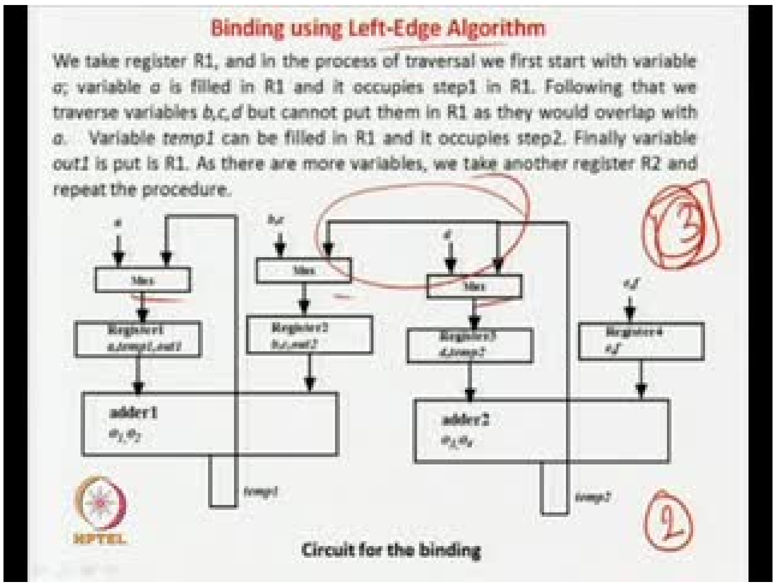
Binding using Left-Edge Algorithm

We take register R1, and in the process of traversal we first start with variable *a*; variable *a* is filled in R1 and it occupies step1 in R1. Following that we traverse variables *b,c,d* but cannot put them in R1 as they would overlap with *a*. Variable *temp1* can be filled in R1 and it occupies step2. Finally variable *out1* is put is R1. As there are more variables, we take another register R2 and repeat the procedure.

Circuit for the binding

Now what are the c all the buckets are nothing but, then you register in this case you have 4 register and the variables u take it so this is very simple algorithm solution these also solving your left algorithm this also solving your binding problem and here you can find out the solution is not at all exponential it is a very polynomial time problem because you have to just take a bucket and everything will be filled up so i am not going to the formal lessons but, you can very easily find out it is much very procedure we just check some wires in the bucket required have to fill up some variable in the buckets so it is very simple problem very simple solution so you can get your solution. If you use this a temp on temp 1 b c out 2 the temp 2 and e and f we take this so you are going to get this type of an arrangement so you can see we require 4 multiplexer so you can just study and find out a temp 1 out 1 b c out 2 d temp 2 a and u this is your binding this 1 so this is going to be your architecture.

(Refer Slide Time: 56:45)



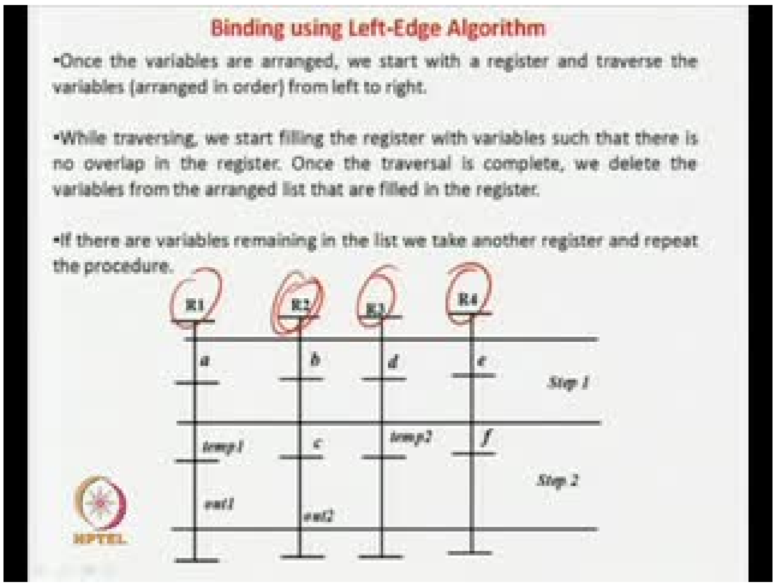
Binding using Left-Edge Algorithm

We take register R1, and in the process of traversal we first start with variable *a*; variable *a* is filled in R1 and it occupies step1 in R1. Following that we traverse variables *b,c,d* but cannot put them in R1 as they would overlap with *a*. Variable *temp1* can be filled in R1 and it occupies step2. Finally variable *out1* is put is R1. As there are more variables, we take another register R2 and repeat the procedure.

Circuit for the binding

So you can see that we require 4 register that is you see you require 3 multiplexers so in the best case fund that we required only 2 multiplexer and adding was very simple in this case there is your actually also crisp cross among the wiring as well as we require 3 multiplex so this solution is not an optimal solution so in other words why do say let us see actually you see so this left is algorithm is the this is taking a simple procedure we just taking a buckets and you are then finding out what is the area required so in this case you find out the area required is 3 what you say is equal to 3 multiplex so that is a big problem for you

(Refer Slide Time: 55:12)

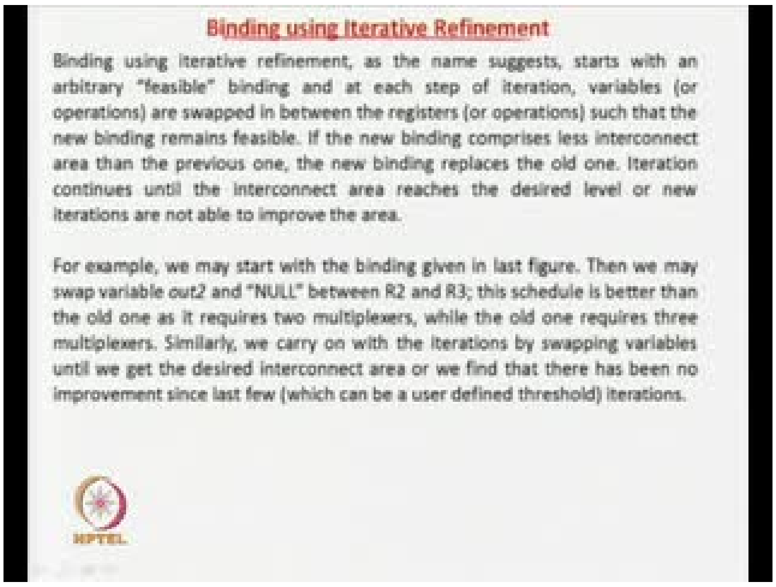


Binding using Left-Edge Algorithm

- Once the variables are arranged, we start with a register and traverse the variables (arranged in order) from left to right.
- While traversing, we start filling the register with variables such that there is no overlap in the register. Once the traversal is complete, we delete the variables from the arranged list that are filled in the register.
- If there are variables remaining in the list we take another register and repeat the procedure.

So the area is not optimal so we can see that binding a what you call say left algorithm is also heuristic algorithm kind of thing again come in more elaborated say in question and answer session that why it happened that you are getting a 3 variables in this case that what is left algorithm is not given the very optimal solution now we will see what happens but, you have to understand here that the algorithm we are using to solve left problem is not a heuristic it is a very exact because nothing to do we are not taking any kind of randomization we are not taking anything that we are not trying such whole substance or so for what we are doing is that we taking some buckets we are trying to fill it out.

(Refer Slide Time: 57:42)

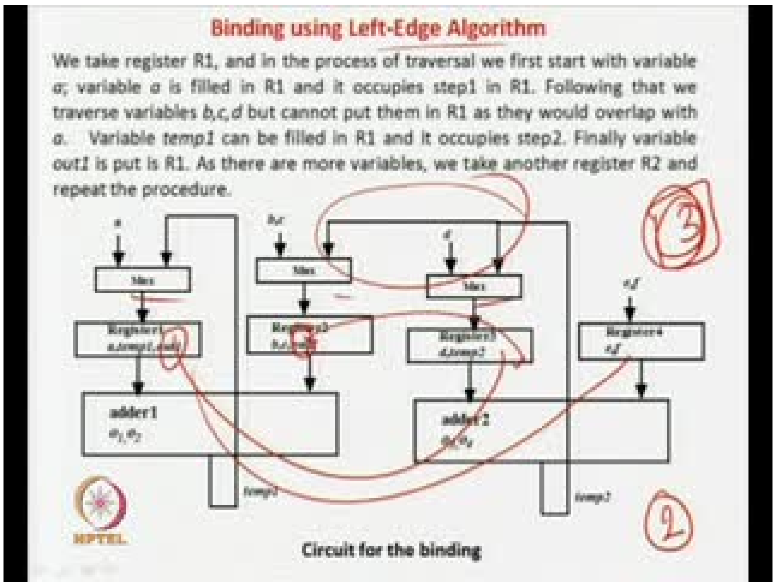


Binding using Iterative Refinement

Binding using iterative refinement, as the name suggests, starts with an arbitrary "feasible" binding and at each step of iteration, variables (or operations) are swapped in between the registers (or operations) such that the new binding remains feasible. If the new binding comprises less interconnect area than the previous one, the new binding replaces the old one. Iteration continues until the interconnect area reaches the desired level or new iterations are not able to improve the area.

For example, we may start with the binding given in last figure. Then we may swap variable *out2* and "NULL" between R2 and R3; this schedule is better than the old one as it requires two multiplexers, while the old one requires three multiplexers. Similarly, we carry on with the iterations by swapping variables until we get the desired interconnect area or we find that there has been no improvement since last few (which can be a user defined threshold) iterations.

(Refer Slide Time: 57:47)



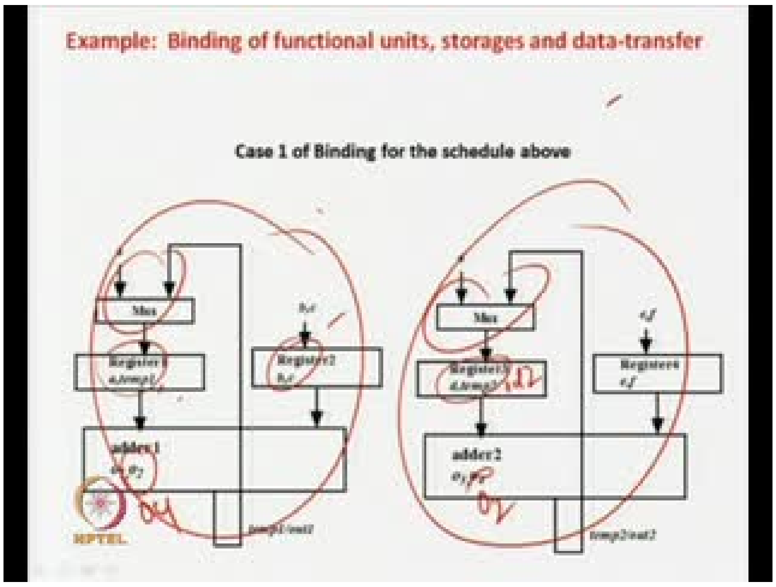
Binding using Left-Edge Algorithm

We take register R1, and in the process of traversal we first start with variable *a*; variable *a* is filled in R1 and it occupies step1 in R1. Following that we traverse variables *b,c,d* but cannot put them in R1 as they would overlap with *a*. Variable *temp1* can be filled in R1 and it occupies step2. Finally variable *out1* is put is R1. As there are more variables, we take another register R2 and repeat the procedure.

Circuit for the binding

So this filling of buckets but, you are not giving optimal solution so why is that because the left age algorithm is design is a actually not very powerful algorithm for solving the problem in most optimal today so what we will do today sometimes people will do that type of algorithm is iteratively so iterative actually merge with left algorithm and then try to solve the problem so what they will do now you find that multiplexers for this now they will iterate is what they will do they will they try to shuffle out from here to here and then they may try to say that i will bring out this algorithm out to from here and put it over here these one you are doing here so that another combination try they know try to say that i may try to put b and c over here and e and f try to re shuffle.

(Refer Slide Time: 58:53)


Example: Binding of functional units, storages and data-transfer

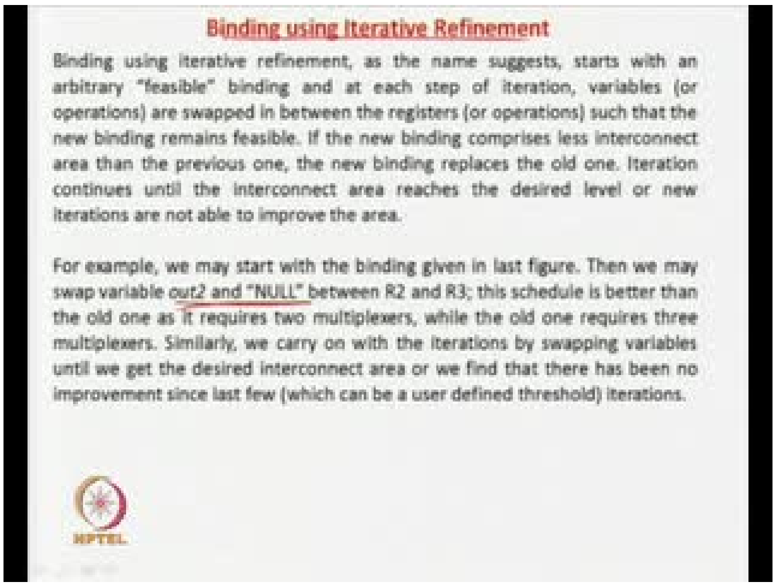
Case 1 of Binding for the schedule above

So in this case we can try to 1 with variable we can try with 2 variable all the re shuffle we can try over here you have temp a 1 out 1 b c how taken move over from here to here this is 1 we can do or wecantry out also with the also take out from here to here out 1 we can take it from here to and try some re shuffles we can do then you can find out what is the area over it because of a multiplex arrangement and keep on doing it and if you find out that if i you can easily find out and easily out 1 from here to here then actually we take go to the very previous i mean this stuff will go up to if you do that you will go to a first this thing we will go to this thing the first thing so in this case a temp 1 out 1 we see here we have out 2 and this is the first very simple structure you will get multiplex and original example start example if you say that will be landing up to we just re shuffle the from here to here.

So the idea is iterative refinement is that you get the solution from the left edge and then keep on iterating with these values by reshuffling the different permutation and combinations of this allocation binding and then if you find that new shuffle resulted over the computer previous 1 you retaining it we keep on doing it as long as you and i want to try with thirty deviations whatever is the best solution we will take you.

So those things are actually heuristics that is iterative refinement along with left edge with mix up you will get a heuristic solution and then what is the idea you keep on doing as we as i said lot of time i will give 5 days to solve the problem and it will be explore or very large number stage space of this binding problem and then try to give a better solution but, you have very less time then also you will get a solution in case of iterative refinement class left stage but, the solution may not be optimal if you try with the first step.

(Refer Slide Time: 60:04)



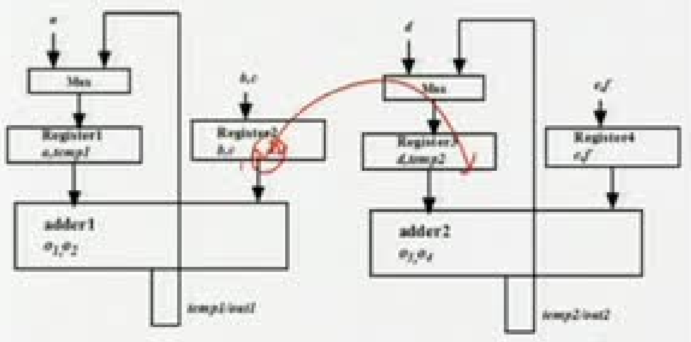
Binding using Iterative Refinement

Binding using iterative refinement, as the name suggests, starts with an arbitrary "feasible" binding and at each step of iteration, variables (or operations) are swapped in between the registers (or operations) such that the new binding remains feasible. If the new binding comprises less interconnect area than the previous one, the new binding replaces the old one. Iteration continues until the interconnect area reaches the desired level or new iterations are not able to improve the area.

For example, we may start with the binding given in last figure. Then we may swap variable out2 and "NULL" between R2 and R3; this schedule is better than the old one as it requires two multiplexers, while the old one requires three multiplexers. Similarly, we carry on with the iterations by swapping variables until we get the desired interconnect area or we find that there has been no improvement since last few (which can be a user defined threshold) iterations.

(Refer Slide Time: 60:16)

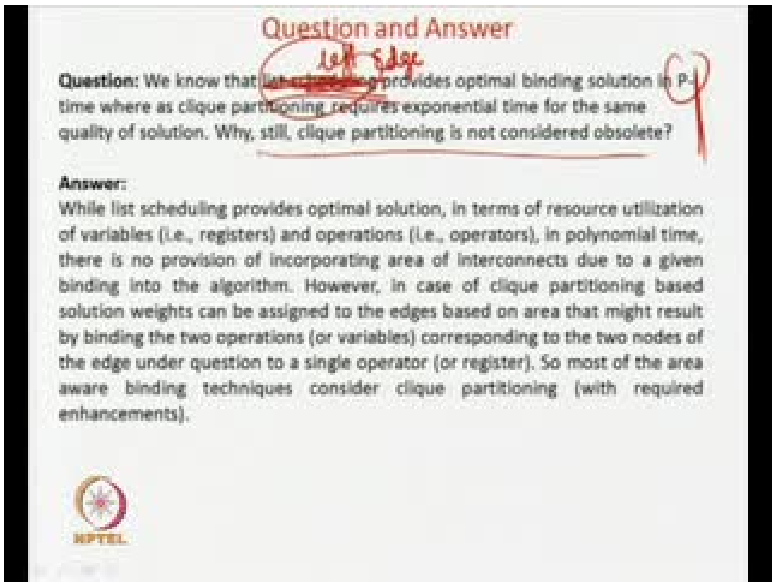
Binding using Iterative Refinement



So without any iterative sediment your answer is 3 which is not iterative optimal solution but some kind of replacement you will get a you will get the most optimal solution in this case but, for a bigger example it may happen that you may take n number of time steps required or n number of iterative the required to go optimal solution or near optimal solution what in case of that what you call the clique partitioning the problem is that the problems in clique partitioning in the whatever you do mean if you are going to get an exact solution for clique partitioning this going to be exp1ntial time requirement and you will get the best solution but, the time required will be very high.

So what you will do if you will go for a heuristic functions of clique partitioning to get the solution so that is 1 area so you can solve this binding problem in two way so 1 is you can do in mapping in the clique problem clique partitioning problem and do not use a exact algorithm use some heuristics for solving the clique partitioning problem so we get a solution which will be the near optimal solution otherwise what you can you go for iterative refinement then keep you go for left as and keep on doing it iterative refinement as long as we have time you also get a solution in a.

(Refer Slide Time: 61:30)



Question and Answer

left Edge

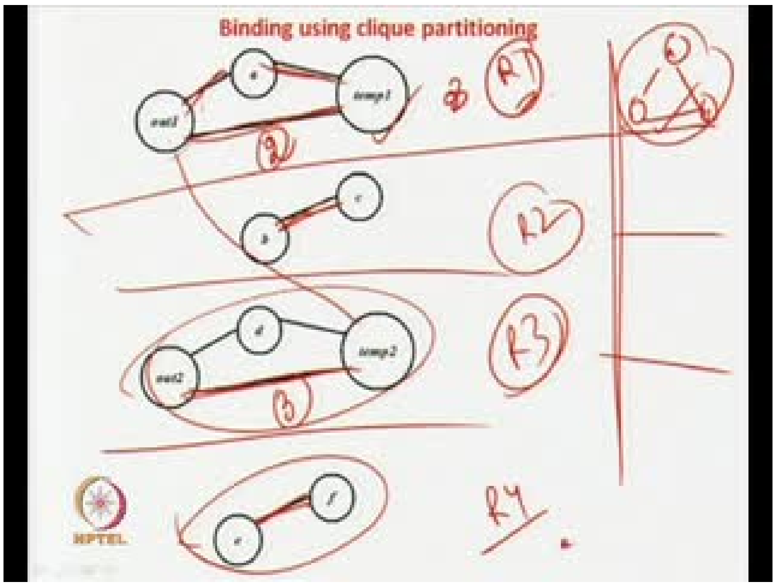
Question: We know that ~~list scheduling~~ provides optimal binding solution in P-time where as clique partitioning requires exponential time for the same quality of solution. Why, still, clique partitioning is not considered obsolete?

Answer:
While list scheduling provides optimal solution, in terms of resource utilization of variables (i.e., registers) and operations (i.e., operators), in polynomial time, there is no provision of incorporating area of interconnects due to a given binding into the algorithm. However, in case of clique partitioning based solution weights can be assigned to the edges based on area that might result by binding the two operations (or variables) corresponding to the two nodes of the edge under question to a single operator (or register). So most of the area aware binding techniques consider clique partitioning (with required enhancements).

Reasonable amount of which may not be the a best one but, it will be near optimal 1 so we have see to i mean 2 different type of solutions i mean 2 different algorithms to solve the binding problem so as i told you if you go for this refreshment that out 2 with null that is where we have actually in this we have out 2 over here so this out 2 you are moving it over here so it is a very simple structure you are going to get in this is the small examples so in the first itself in will refinement itself so getting the best solution so all as may not be the true 1 before you closed over so we are going to the question answer session now we know that list scheduling provide optimal binding solution and in time as clique partitioning requires exponential time for the same point quality of a solution why then clique portioning is not considered obsolete now we have to understand what is the minimum so the clique partitioning as you already know takes a exponential number of time exponential solution to get the correct answer but for clique but for what you can call is list scheduling the solution is in p so what does it.

mean list what you can call the list what you say list scheduling if you do not merge with this it refinement that itself is algorithm or you say this is a solution it will take some buckets try to fill up the variables over there then it will give you solution so itself be the solution if you merge it with it will refine in a then you can get better and better solution and that in all becomes a heuristics but, if i become iterative refinement list partitioning stands itself is a solution and you can understand that the solution is very simple so what it does it actually does fill up some bucket and gives the solution then it is done in very less amount of time but then why you are going for clique partitioning based 1 or why we are actually merging iterative refinement why do you want to merge iterative refinement and with this least scheduling what do you want to do then the answer is list partitioning list scheduling this is not list scheduling with this is actually not least scheduling that is the actually left edge so the problem is we know that left edge provides optimal binding in p time in clip partitioning the exponential time so it is not solution in this solution this is actually is an issue algorithms so it left edge we know that left edge over optimal binding in p time where as the click partitioning request exponential time then why it is so as i told you left edge is nothing but, it take some variables and put it in some the buckets that is how it is done but, , still we can see that does not give an optimal solution because in all case it given a solution of 3 multiplexers so and if you merge it with the iterative refinement then it is heuristics point but let us look up but, you see why you need clip partitioning so in clip partitioning you can increase that you can and its very popular algorithm and only the one which is mainly used because some weights.

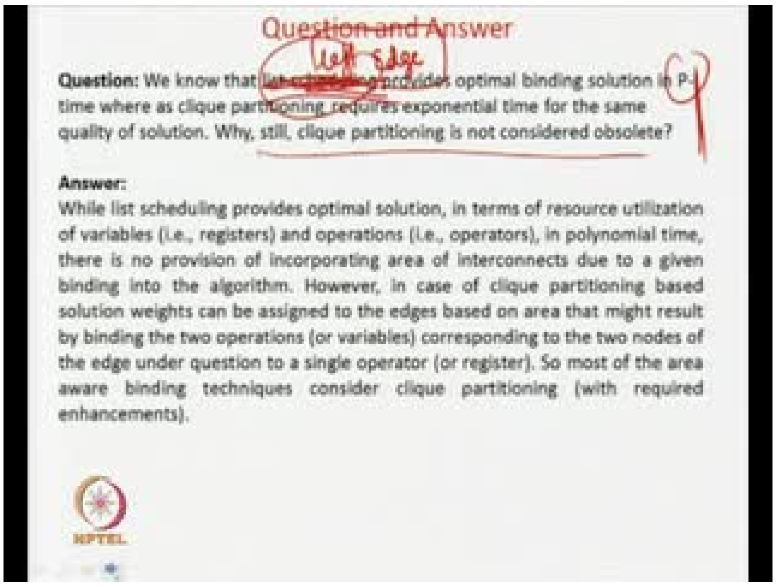
(Refer Slide Time: 64:01)



So what is the weights like we can see here that we having r 1 r 2 and r 3 so in this case also showed about that there can be different other solutions like a then it was out 1 and this temp 2 kind of a thing and collection some other this is the 1 solution we say that another solution whether you take this solution and whether you take this solution that means in case of temp 1 let us see this is temp 2 and temp 1 this is another solution so we can put some weight in the areas like you can say that if i merge out 1 with temp 1 then i can say that put a weight 2 so we are get some so in case i put a out 2 with temp 2 if i merge i say where from this weights will come the weights desired by another heuristics mechanism or another any mechanism saying that if i merge out 2 with temp 2 then what will be the requirement of multiplex or how many more additional interconnect binding interconnect what do you say interconnect area in terms of multiplexers are over here binding this 2 similarly, if i can bind this 2 with another thing that means if i bind out 2 with temp 2 more area will required kind of thing so that type of weight you can put over here whether i merge out 1 with temp 1 or out 1 with temp 2 so interconnection of here also.

So you can thing that whether out 1 with temp 1 is better solution or out 1 with temp 2 is better solution we can put some weights so that is if you merge with out 1 with temp 1 so if you do this which is going to a better solution so those weights that is if you merge these 2 nodes what in that area over it because of interconnect binding so some weights you can put then you can get on the algorithm so it will try to find out the solution which is actually the quick partition solution the weights and edges will merge and will be merge only in the cases only the weight are very less that means you can embed some into your clip partitioning solution that is when you are and how it will do this merging so that you get the more optimum solution compare to some other venue of the solution 1 2 3 so we can put some weights in the edges so it can be automatically guide.

(Refer Slide Time: 66:14)



Question and Answer

Question: We know that ~~list scheduling~~ left edge provides optimal binding solution in P-time where as clique partitioning requires exponential time for the same quality of solution. Why, still, clique partitioning is not considered obsolete?

Answer:
While list scheduling provides optimal solution, in terms of resource utilization of variables (i.e., registers) and operations (i.e., operators), in polynomial time, there is no provision of incorporating area of interconnects due to a given binding into the algorithm. However, in case of clique partitioning based solution weights can be assigned to the edges based on area that might result by binding the two operations (or variables) corresponding to the two nodes of the edge under question to a single operator (or register). So most of the area aware binding techniques consider clique partitioning (with required enhancements).

Some solution in 1 is good whether it solution 3 is good all this weights excetra can put here. But you can think left algorithm is very simple you just you have to fill up the bucket so there is no option and no choice is in the single solution, and that may not be optimal solution and that has been in the case of something so in this case you also put some purity define to make it a better 1. So with this we stop the discussion algorithms for scheduling allocation and binding is high level synthesis in the second module so in the next module. What we are going to discuss that once our high level synthesis is done you are get design or black box architecture design now how we can use them to get level design that is called the get level synthesis so in the next module will be focusing on that.

Thank you.