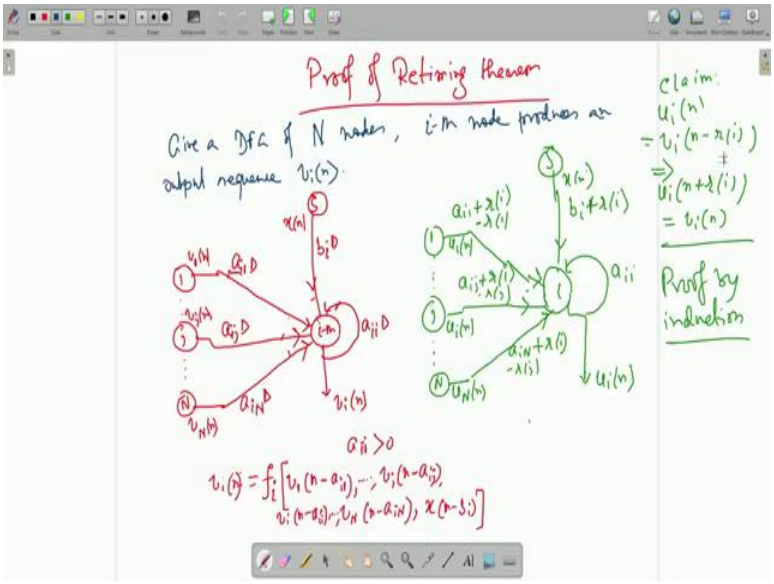
Course on VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 6
Forward Path and Loop Retiming

(Refer Slide Time: 00:30)



Okay, so now we come back to the proof. In the proof I said that I proved by induction, induction what? This is he relation that at any index n we have to show that Vi n is an advanced version of Ui n that is Ui n is the delayed version of Vi conversely if Vi n is at advanced version that is Vi n is nothing but Ui n plus some amount, r of i, r of i is the returning value of i th node which is known to you, I have to prove this.

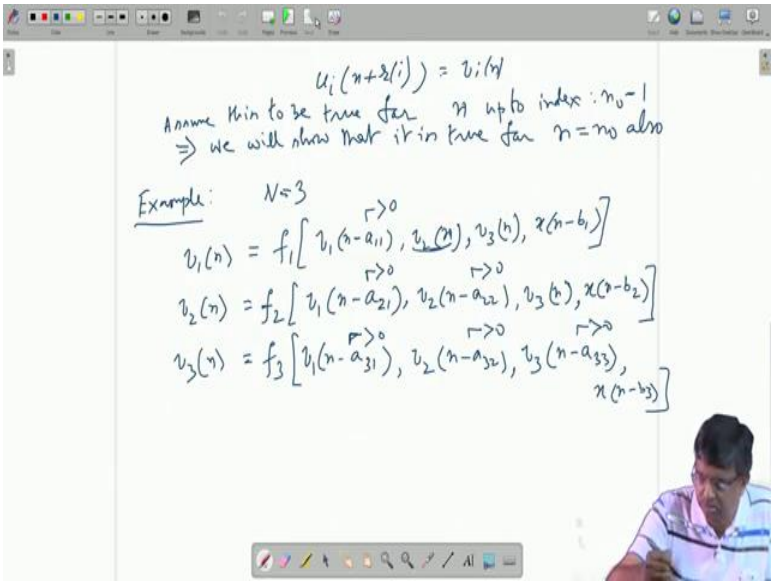
I said I will prove by induction, by that I mean I will assume that this relation is true for all n less than some index n naught, there if it is n naught minus 1 fine, if it is n naught minus 2 fine, it is n naught minus 3 fine, dot-dot-dot it is true. Under that condition I will show it is true for n equal to n naught also and then it will continue.

So I repeat I will assume this to be true for all n less than n naught, n naught is some index, when it is n naught minus 1 true, n naught minus 2 it is true, n naught minus 3 it is true. Now I will show it is true for n equal to n naught also, and one case I will show directly that easily that is simple. Now here what is Ui n?

I will go the new page just to take a look because there the diagram will not be there, Ui n will be nothing but something similar because the node has not changed so it will be function, Ui n will be like Vi n Ui n function fi instead of V1 it will be U1 but the delay will change. Earlier I had ai 1, so ai 1 has been changed to ai 1 plus r of i minus r of 1, ai j has been changed to ai j plus r of I minus r of j dot-dot-dot, only delay values will be changed.

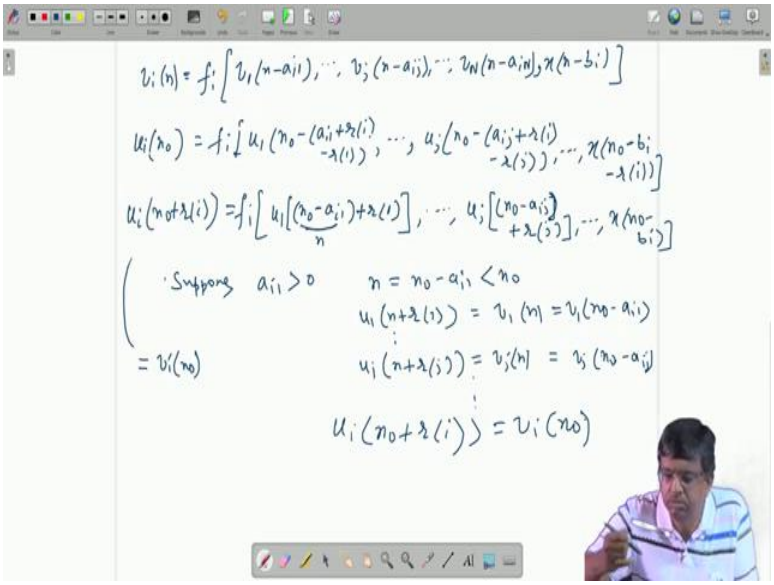
V1 will be replaced by U1 because the new circuit has no V, only Us, Vj has Uj, Vi will be replaced by Ui dot-dot-dot, x will be as it is, bi will be replaced by here it is bi, it is bi plus ri so bi will be replaced by bi plus ri. So that will give you Ui n I will take a special case n naught Ui n naught and I will show that is equal to, I mean if it is Ui n naught then Ui n naught plus ri will be equal to Vi n naught. So this relation will be valid at n equal to n naught also.

(Refer Slide Time: 02:44)



And remember every node is basically a function of all other nodes and itself but they passed values only, even if the delay is shown to me 0, effectively it turns out that every node depends on the passed value of all the nodes.

(Refer Slide Time: 03:03)



So that is for your convenient I rewrite here, I had Vi n was fi just for your convenient so that you do not have to refer to that page it was V1 n minus i yeah destination i1 dot-dot-dot Vj n minus ij dot-dot-dot V capital N because Vi n, Ui n, Ui suppose I take n to be n naught it will be fi instead of V1 it will be as I told you U1 n is replaced by n naught because a particular n I am taking minus the delay.

Delay now is not ai 1 it will be ai 1 plus r of i minus r of 1 dot-dot-dot I will not write all the terms the general term would be enough n naught minus ai j plus dot-dot-dot xn naught minus bi and there is another term r of i. So Ui n naught is such the advanced version Ui n naught if n naught is replaced by n naught plus ri there is returning value of node i, r of i is brought in, so replaced n naught by n naught plus ri. So if you have n naught plus ri here that ri will cancel with minus ri, here also ri will cancel with ri there is a minus i everywhere.

Okay, n naught here, n naught plus ri, ri will cancel with this ri, where r of i is comes with minus i with all the expressions. So will be left with U1 remaining one, remaining part I write this way I put n naught minus ai 1, n naught minus ai 1 in a bracket ri is already cancelled out and minus-minus plus, plus of r of 1, dot-dot-dot U j I get n naught minus ai j this is okay I can make it like this plus r j minus-minus plus rj dot-dot-dot and lastly ri is gone so simply xn naught minus Vi.

This one, but now I assume the I in use the induction hypothesis come here, ai 1 suppose ai 1 the delay between node 1 to node i original circuit, original DFG before retiming there is a single

delay only ai 1 in the path from or in the edge from node 1 to node I. Suppose ai y1 is not 0, it is positive it could be either 0 or positive, suppose it is positive. If this is positive then this is if we call n equal to if we call this n, n say n naught minus ai 1 so this is positive the whole thing is less than I mean is less than n naught.

So n is less than n naught, so I have got U1 n plus r1, by my induction assumption for any n less than n naught the relation is valid that is u1 n plus r1 that is if the U output node 1 output after retiming gives advanced by r1 this will be same as original V1 at that index n, this is what we are trying to prove. Our assumption was that this is true whatever is trying to prove this is true for all n less than n naught that is up to n naught minus 1, n naught minus 2, n naught minus 3 and so and so.

Okay, but not at n naught but if this is positive then this whole index n is obviously less than n naught so i can use my induction hypothesis. So n plus r 1 that is U1 and index n and advanced by r of 1 retiming value of node 1 that by their assumption will be nothing but original V1 at n, that is at that point. Similarly assume that this delay, all the delays are positive this ai j also positive, so n naught minus ai j if you call that to be n, so again by the same token Uj n plus r of j n is here, n naught minus ai j, suppose ai j also positive, so a is less than n naught.

So again my assumption works, induction assumption Uj n plus r j that is advance by rj that will be same as vj at n and n means here, what is n? I replace n by what I took dot-dot-dot, so if all these delays are strictly positive then obviously I have got this transfer to be fi V1 n naught minus ai 1, V1 and instead of n, n naught minus ai 1 dot-dot-dot Vj, n naught minus ai j, x n naught minus Vi, x n naught minus Vi it is nothing but this transfer to be equal to Vi n naught, if you put n naught here fi. V1 n naught minus ai 1, Vj n naught minus ai j dot-dot-dot xn naught minus Vi, x already n naught minus Vi.

So this relation is valid at n equal to n naught also, that is Ui if from n naught it is advance by ri, Ui of that will be same as Vi at n naught only. So that relationship continues at n naught but this also that proves the validity of the induction but this assumes that this relation positive, strictly positive not 0, at this 1, 2, 3 so that n is less than n naught not equal to n naught then you can ask me a question whatever those DFGs were suppose ai 1 is 0, if ai 1 is 0, n and n naught at same so is not less than, it is equal to so you cannot use the induction hypothesis.

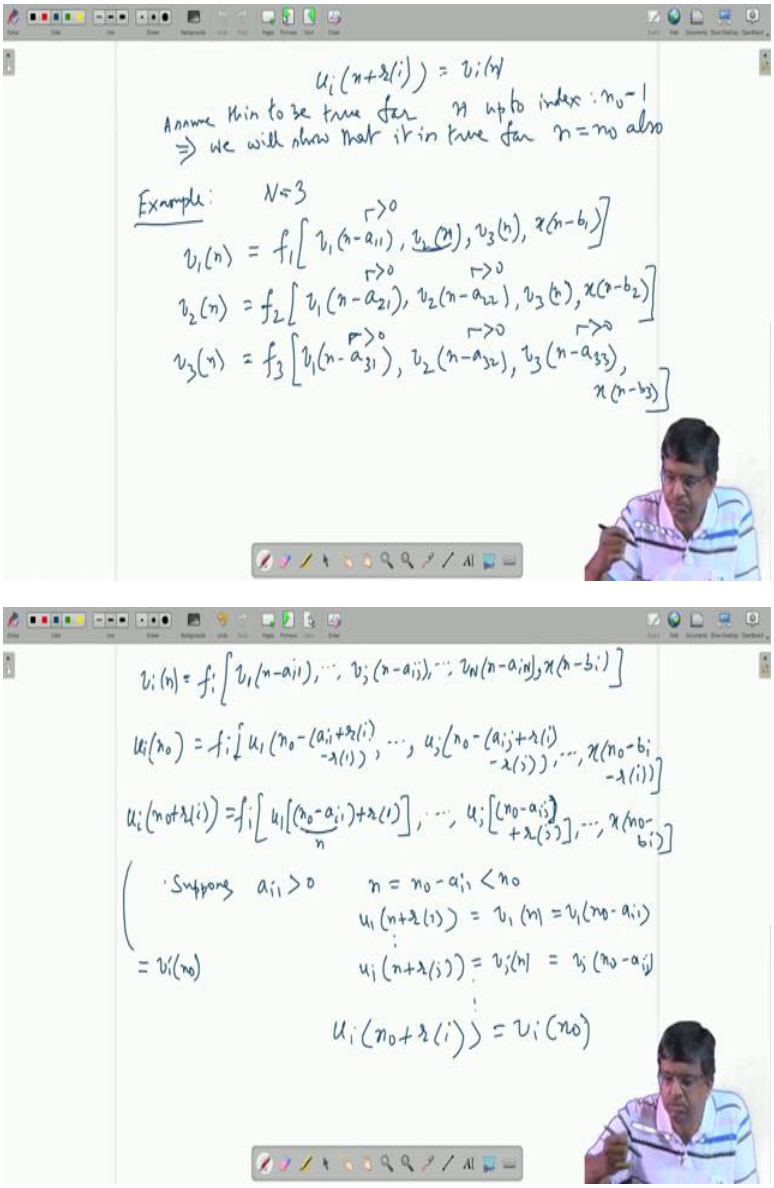
Because induction hypothesis says that for n less than n naught the relationship which I am trying to prove is valid, this is valid but if n is n naught then I cannot say that this equal to this, if n is less than n naught then U1 n plus or r 1 will be V1 n. But if n is equal to n naught then what happens? Then I tell you the moment I come across this that okay, the ai 1 is 0 in my original expression okay what I do is this, I will have U I mean in the original expression it would be Ui n from here only you can say Ui n equal to suppose is the original expression.

Suppose Vi n equal to fi you start back with Vi n, at Vi n fi b1 n only no delay, if ai 1 is 0 it will be V1 n only. The moment I get V1 n I will replace it okay or say start with Vj because it is a self-okay V1 n. Suppose fine, Vi n is a function of suppose V1 n only no ai1, the moment I come across this I will replace V1 n by its own function, I told you at example of node 3 nodes, the moment I come across a term here inside with no delay I will replace by own function.

So function within a function and this way if I proceed finally I will get a huge expression where all the terms will be strictly delayed, right, no 0 delay term like ai 1 0 or will come there all strictly delayed terms will come and therefore this will not come up that okay I mean what happens if the particular delay is 0? Because the moment I have say ai 1 0, Vi will be replace by f1 something and so on and so forth.

So will have fi then using that f1 maybe f2, maybe within f1, f2 within that f3 so on, so the huge in static expression eventually all the terms will be strictly delayed. Strictly delayed means with n there will be a positive term, n minus a positive term strictly delayed term.

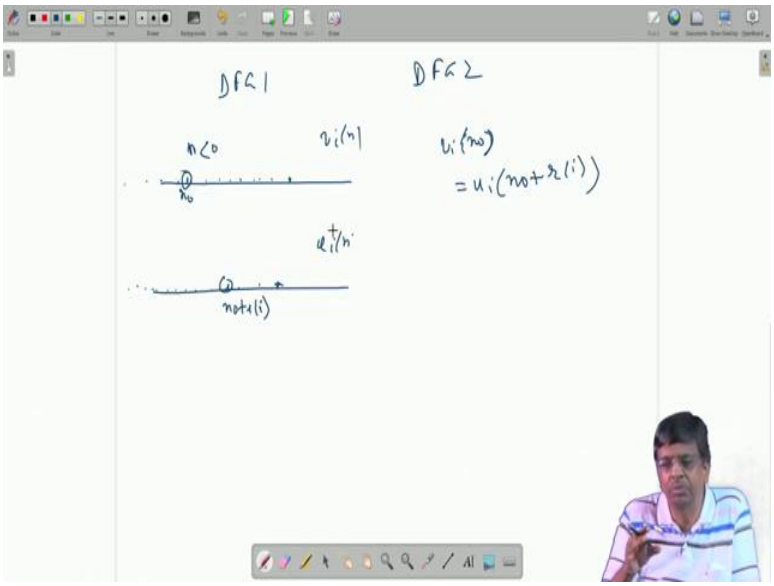
(Refer Slide Time: 13:41)



Like as I told you here okay even if there is delay v2 n, v3 n the moment I get v2 n I replace v2 n by this function. Within the function I get v3 n no delay replace by the function. Then v2 n itself, v3 n I replace by the function, so eventually where v1 n will be a function, within that another function, within that, but all terms will be strictly delayed, this is positive, this is positive so that situation will not arise, because I will can work on this delayed terms if n is n naught, so n naught minus a delay which is constant which is positive so overall is less than n naught so induction hypothesis will work I replace my path back and I get that relation, understood?

So this shows that Ui this relation is valid here also, earlier we thought we said that it is valid for all n less than n naught then if you put such an Ui n plus ri equal to Vi n then when n equal to n naught now it is shown this is also valid and therefore it is valid for n naught plus 1, n naught plus 2 etc. But now I have to show a direct case that at least one direct case where you know left hand side equal to right hand side that is very easy.

(Refer Slide Time: 15:18)



On one hand I have got DFG 1, another DFG 2, DFG 1 before retiming, DFG 2 after retiming. Suppose this is the time axis for V any node, Vi n and this is the time axis for same node but for Ui n and this is the time origin. I assume that left, for all n less than 0 this DFG it is cleared that is it is at rest that is all the node outputs are 0, 0, 0, 0, 0, 0, 0. Here also all the node outputs are 0, therefore you can assume you take an index n may be n naught.

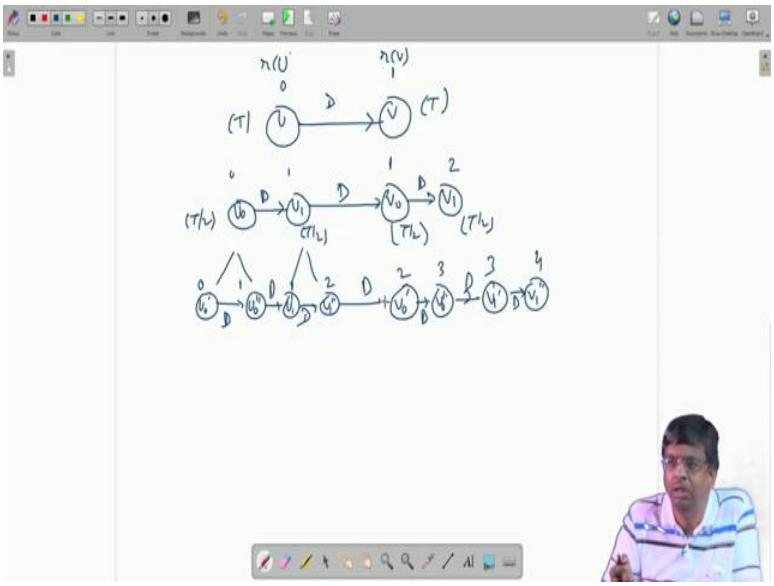
You take an index n naught and go here n naught plus r of i make sure n naught is sufficient negative so that n naught plus ri also to the left of origin. Here Vi at this n naught is 0, Ui at this n naught plus ri is 0, so this is equal to this, so this is the case where I can directly show that Vi n naught equal to ay Vi n naught which is 0 is Ui n naught plus ri. Because both the systems are cleared, original system cleared and so this is 0 at any n naught this is clear so that n naught plus ri is also 0. So 0 equal to 0 is satisfied.

Now this is not just for theory, it is important that both the systems are cleared, they should not start with any you know other non-zero initial condition because then only they will be

equivalent. Otherwise if you start with if it is a flip flop and you know carrying some initial data that will create you know error in the system output. So both have to start from 0s. So this is the proof of the retiming theorem, that every node I mean every edge delay you can changed by adding that if the delay is original delay is w then w plus r of V minus r of U.

V is the destination node of the edge from U to V, U is the source node for that edge, r of v is the retiming value of node V, r of u is the retiming value of the node U. So retime delay will be w original plus of r of v minus r of u and this will do throughout the DFG, make sure that none of the new delays becomes negative and you get an equivalent DFG but with new delays. This is proof for the retiming theorem, now we will work on its implications.

(Refer Slide Time: 18:20)



Start with this, suppose the DFG does this U, V there is no delay it takes time T, it takes time T. So critical part is T plus T to T because first in the one cycle in every cycle you have to first carry out this job take T time, in the same cycle you have to send it here and has to work out this job, for another T cycle then total will be 2T then only you can send another data and you know this can take another data and start working.

So that means 2T is the critical path, you cannot have clock period shorter than 2T. Now suppose I choose retiming value r of V, suppose I chose 1 here, 0 here, so new value will be original 0 plus 1 minus 0, so it will be 1, so 1 D will come up. So this will be pipelined because in one

cycle it does the job of T amount of time this is required here only in the next cycle not in the same cycle, so T will be good enough in 1 T amount of time it does the job here.

Then next T amount of time it works on that and that time it prepares output for next cycle operation here, so the two times do not get added the pipeline you have critical path, now is equal to T. Now suppose I become greedy I want to make the critical path still shorter make it the system faster so I get inside the node, see the operations.

Suppose I break the node I am able to break the node as an example only into two nodes, sub-nodes one followed by the other and suppose both take T by 2, T by 2 and then ofcourse this one so this together is U, this also I break into 2 nodes, sub-nodes either T by 2, T by 2 just for example say and I have a delay here, this delay.

Then I chose retiming value 0 1, 1 because in want to bring a delay here otherwise here they are working in the same cycle T by 2 times spend here, form a output then give it to here next T by 2 time it dedicates to carry out job on that so total it takes T by 2 plus T by 2 T, same here. So if I can bring a delay here, how to bring a delay here, if I chose 0 and 1? Then originally 0 plus retiming value 1 then 0 so which is 1, so delay comes. I want to retain this delay, so I chose 1 here.

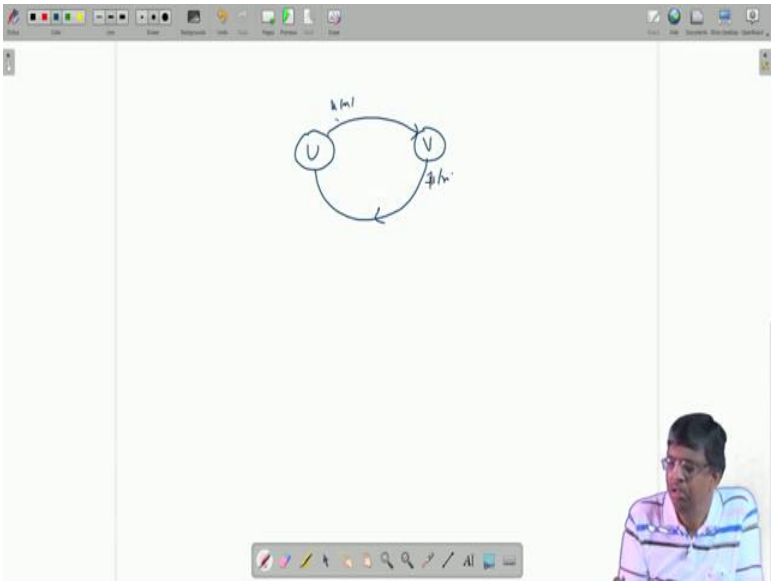
So 1 delay means 1 original plus retiming value 1 minus retiming value 1, so 1 delayed so on. Then I want a have a delay here, so already 1 here so I chose 2, so 0 plus 2 minus 1, so I get D. So now the critical path is T by 2 or T by 2 or T by 2 or T by 2, so T by 2. Then you can go further break it if possible if hardware permits this may be U0 prime, U0 double prime you know U1 prime, U1 double prime, V0 prime, V0 double prime, V1 prime, V1 double prime and this has a path which has this delay, it has delay path, this has delay like this.

I get through 0 here, 1 here so we have a delay, 1 1 so that 1 delay plus 1 minus 1 so 1 remains 1, so delay remains. Then we want to have a delay here if it is 1 already D choose 2, so 0 plus 2 minus 1 that means here we chose 0 1, so 0 plus 1 minus 0 1 if you choose 1. 1 plus 1 minus 1, so 1 remains 1, here there was nothing so we choose 2, so 0 plus 2 minus 1 a delay comes. Here already there was a delay no need to change, take 2, so 1 plus 2 minus 2, 1 remain.

Here I need a delay so take it to be 3, 0 plus 3 minus 2 a delay comes. Here only we have a delay, so nothing doing, this remains 3, 0 plus 3 minus 3 then it was 3 I need a delay here so 4 and this way you see now the critical path will be T by 4, T by 4, T by 4, so T by 4. So as long as my hard work path needs this kind of breaking of nodes into smaller parts I can go on inserting delays, these delays were not in the paths circuit originally, this delays were not in the circuit I am adding from external world, this called forward path pipelining.

There is only forward direction, in forward direction I can go retiming, I can bring in delays from outside again-again-again from outside at the critical path reduces, this is called forward path retiming. But now I consider another structure that is loop and if I have to do the same thing here we will see what happens.

(Refer Slide Time: 23:56)

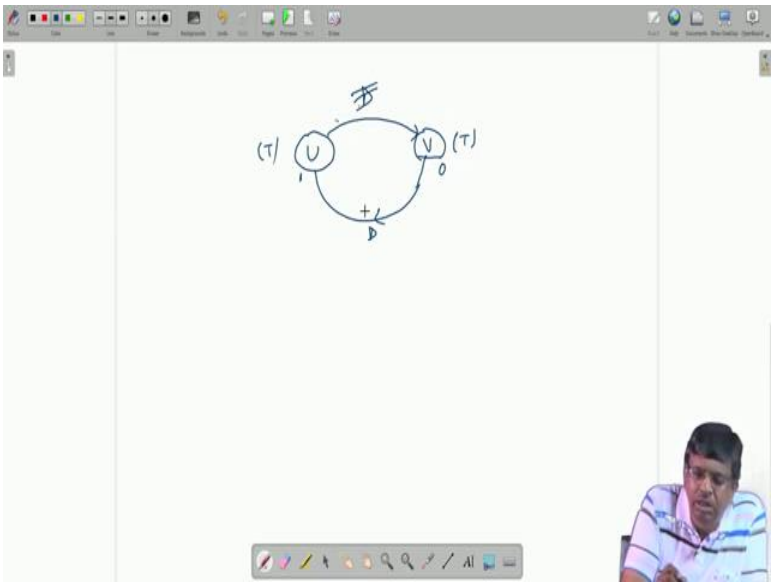


Suppose there is a loop, just 2 nodes U, V there cannot be a delay free loop as I told you because that means if you want this output that will depend on this input but this input depends on that same output, so the same output eventually depends on itself which is not possible. Then if you call U of n, if you V of n U of n depends on V of n and V of n is the same cycle depends on of U of n because there is no delay which means U of n depends on U of n, which I have shown earlier is not possible.

There it should be at least one delay take it from b delay free loops cannot be realized, this is a delay free loop because if it is a delay free loop as I told you if it is generating a signal U of n it

will depend on this V of n but V of n will depend on U of n so U of n will turn out to be function of U of n which is not possible but if there is a delay here then U of n will depend on V of n minus 1, V of n minus 1 will depend on U n minus 1 so U n will depend on U n minus 1 which is fine. So there must be at least 1 delay.

(Refer Slide Time: 25:33)



So let the delay be here, let the delay be here okay and this is taking time so T, this is taking time T, so now I become greedy like before I break it into 2 sub nodes T by 2, T by 2. Now fine, so this now let us see the critical path here first, this path, this edge if you forget the lower edge just take the upper edge there is a delay supporting U and V, so in 1 cycle U is taking the some job doing it in time T, this is used in next in this node V only on the next cycle because of the delay.

So the 2 types do not get added in 1 cycle you do the job, next cycle it is that works of that and at that time this node I mean take some new data and prepares the output for you use here in a sub sequence cycle so on and so forth. They are pipelined, so if I had only the upper path, upper edge or upper path, critical path would have been just T but there is a lower path, lower edge, here there is no delay. So first you have to carry out a job at V taking T amount of time in the same cycle give it to U, it has to spend another T amount of time in the same cycle.

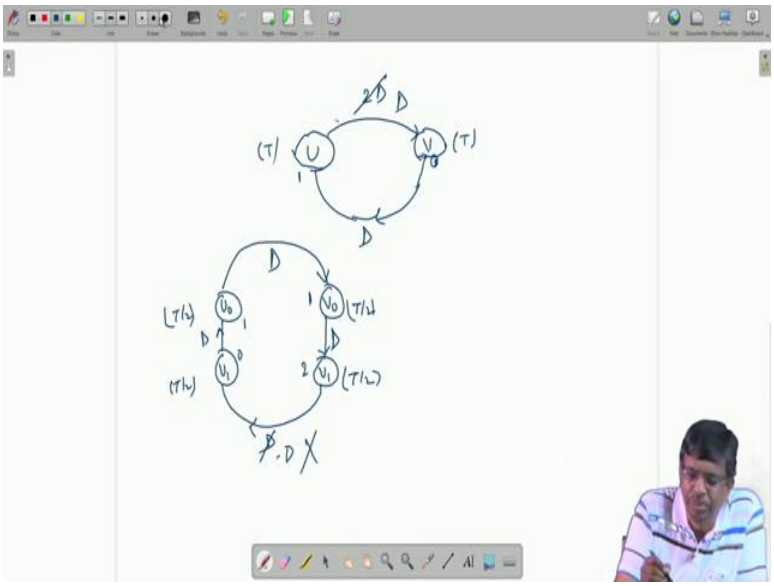
So the 2 times got added got 2T, so from above half, upper half critical part is T from lower half critical path is 2T you have to take the larger one, largest one, so critical path here is 2T. So to avoid that I want a delay here, to avoid that I want delay here, how do I have a delay here? If I

have a retiming value 1 and 0 then what will happen 0 delay plus 1 because this is the destination now, arrow is pointing here, so 0 delay in this edge plus 1 minus 0, so 1 so I will have a D.

But then what happens upstairs in this edge? I have 1, now 1 plus 0 because this is the arrow is pointing here now for this edge, so 1 plus 0 minus 1, so this will go. So lower half is now pipeline because of the delay, the 2 time, the 2 node times do not get added because there is separate cycle but upper half no delay so what happen in first you take time T, do this job in the same cycle give it here it takes another time T, so T plus T 2T.

So critical path remains 2T, delay only has come down from upper half edge to the lower edge, so it does not you know I mean help me, the retiming here does help me reducing critical path at all.

(Refer Slide Time: 28:37)



But you see there will be a change now, I will go back to original, suppose instead of 1 D here I had more delay I started with I told you delay free loop is not possible so I took 1 D but suppose I had 2 D that is the total loop delay or delay on edge increased now the delay in this loop is in edge is 2 D, total delay is the loop also therefore 2D.

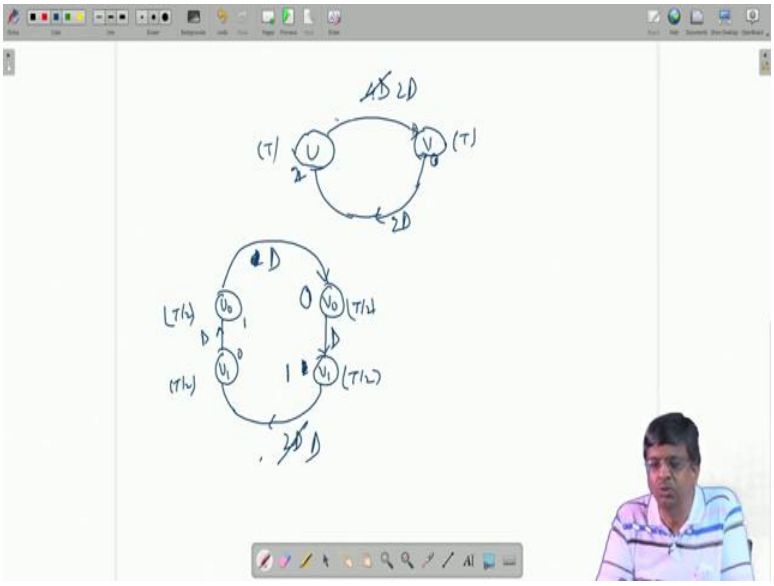
Now if I choose retiming value 1 here sorry 1 here and 0 here as before, so here I will have 0 plus 1 minus 0, D will come but up stair what will happen, now I have got 2 plus 0 minus 1, so 2 D will not turn out to be 0 it will turn out to be D. So now this is the happy situation, this edge

also delayed, this is also delayed, so critical part will be T which means the moment I increased the delay in an edge and therefore total delay in the loop then by retiming I can have better situation I can have delay in both the edges.

Now suppose I mean the critical part goes down, now suppose I become greedy I break U into two parts U0, U1, V also in two parts V0, V1 as before. So I have D here this is D, lower half D and no delays here but T by 2, T by 2, T by 2, T by 2 or I have to bring down the critical path from T to T by 2 as I need that time, can I do that here? If I try I want a delay here so start with 0 and 1 retiming values.

So 0 plus 1 because the arrow is pointing to this, so 0 plus 1 minus 0 so it will become D. Then this D remain as it is, I do not want any additional D so if it is 1 let it be 1 so 1 plus 1 minus 1, so 1 remains 1. Then I want a delay here that means if it is 1 make it 2, so 0 plus 2 minus 1 D, but then what happens here, I have 1 plus 0 minus 2 so what turns out to be minus D which is not possible. So I cannot do that but suppose I had more delay in the loop.

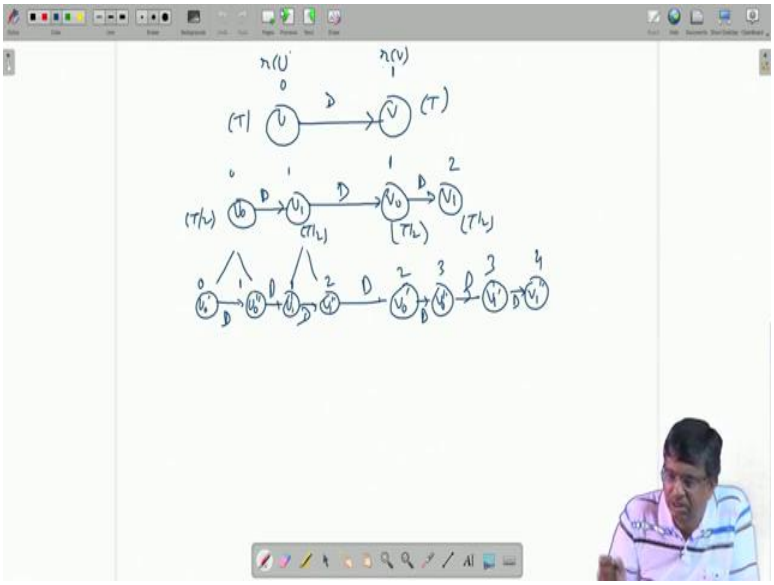
(Refer Slide Time: 31:38)



Suppose originally instead of 2 D I had more delay, I had say 4 D, then by choosing maybe 2 here and 0 here I can bring in 0 plus 2 minus 0, so 2 D here and 4 plus 0 minus 2, so 2 D here. So 2 D, 2 D if I do like that then you see you choose 0 and 1 here you get 1 then originally I had 2 D but 2 D is not required 1 D is enough to be make a delay but pipelining 1 D is enough.

So suppose I may get 1 D, what to do from 2 D? If I choose 1 and if I choose 0 here, then 2 plus 0 minus 1 so it will become just 1 D, even that is enough for B then I need a delay here, so it is 0 so 2 is not required 1 is enough for me, where originally 0 plus 1 minus 0D and now you see I had this 2D originally here, originally I had 2 D so now what is happen 2 plus 0 minus 1, so 2 minus 1 it will become D but still I am happy every edge is delayed and processor takes is T by 2, T by 2, T by 2, T by 2. So critical path is T by 2 but for this I had to increase the delay loop delay or rather delay in one edge and therefore total delay in the loop from 2 D to 4 D, otherwise I could not.

(Refer Slide Time: 33:26)



But if you go to the previous page you see originally I had nothing then I could happily put a delay here then I broke it in two parts, I could happily put new delay from outside world and I could go on doing it I had never had this problem of you know changing the delay value of the original edge.

So this is the problem of retiming a loop, retiming a loop your freedom is limited. Depending on the original loop delay you can go only up to some extent not beyond that. So this has a fundamental region which will discuss in the next class, thank you very much.