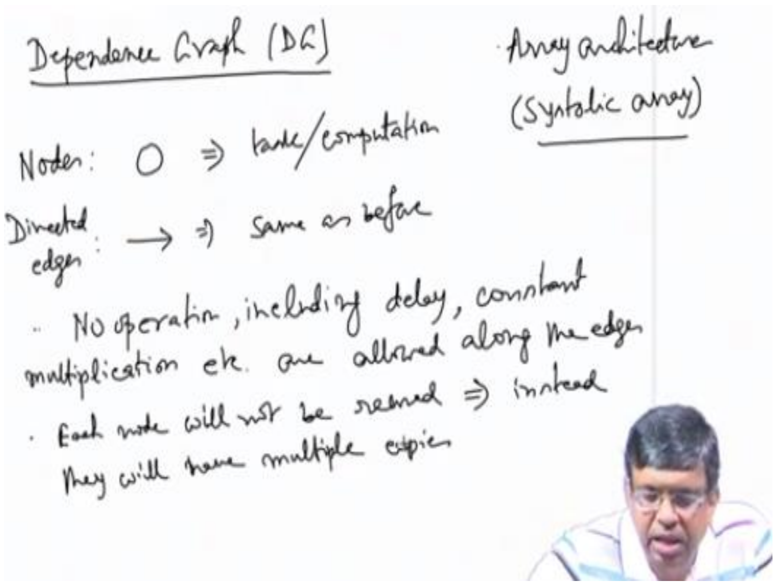
Course on VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 04
Dependence Graph, Basics of Retiming

(Refer Slide Time: 00:27)



Dependence Graph (DG)

· Array architecture (Systolic array)

Nodes: $\bigcirc \Rightarrow$ task/computation

Directed edges: $\longrightarrow \Rightarrow$ same as before

· No operation, including delay, constant multiplication etc. are allowed along the edges

· Each node will not be reused $\Rightarrow$ instead they will have multiple copies

So next graph the third and the final one is dependence Graph. It comes in the context of certain architectures called array architectures. Prominent array architecture is systolic array. Dependence graph like the SFG and also the DFG it also consists of two things only. One is nodes integrated by bubble meaning task computation etc and directed edges same as before, but there are two differences.

One is no operation including delay, constant multiplication etc are allowed along the edge. Remember in the case of SFG we allowed certain operations like delay and also cost and multiplication in the edge. In the case of DFG constant multiplication was ruled out, but delay was allowed here none of them is allowed that is one thing. On the other hand in both SFG and DFG I had nodes.

One node dedicated for one job and that it was doing again, again in all cycles n th n plus 1 th, n plus 2 th same node, but here each node will not be reused and instead they will be copied they will have of multiple copies. Let me explain actually it is dependence graph, graph is the trickiest of the 3 it requires proper understanding. In the case of say SFG or DFG of course we have a particular node. So node was taking some data at the n th cycle current cycle producing something.

Then again in the next cycle it is again taking another data again producing another thing by the same operations and so on and so forth. So iteration after iteration after iteration or clock after clock after clock the same drawing the same node was working again, again, again here it is not. Here I will have several copies of the node drawn in my diagram one copy will work in a particular n th cycle another copy may work in then plus 1 th cycle another copy may work in the n plus 2 th cycle like that so no reusing.

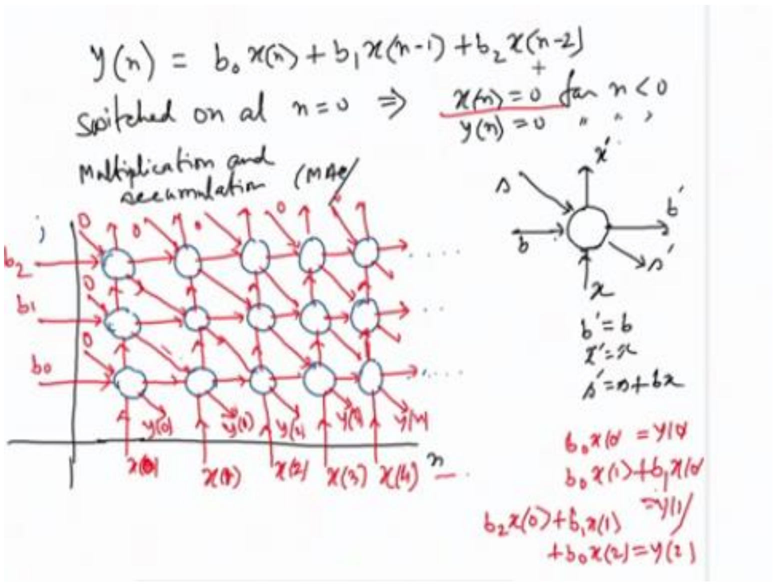
Now unlike the SFG and DFG as I told you dependence graph is but trickiest because it requires certain intuition to draw it. I mean there is no systematic procedure of teaching you how to draw dependence graph. We have to look at the problem and then you know have to use some imagination. It comes just by practice, we will take some example now. Now when you draw dependence graph what is the advantage of dependence graph?

You will see that a very regular pattern of connection and nodes will emerge. I mean whatever will be the appearance in a particular sector the same thing will repeat here. So this is very good for VLSI realization because in VLSI what they do they have produced a mask of a particular spot and then you keep repeating the mask again, again, again the entire circuit builds up. That is why it is very, very good for VLSI realization.

Also all connections are local you do not have to take a wear and route it through the network all local connections very little propagation delay I mean you can pipeline all the edges putting delays all these things make this dependence graph very useful because it leads to the final architecture called systolic array or array architecture. Now in the heart of the dependence graph is a basic processing cell or processing unit which you define separately.

That ok this is the basic unit I will be using that will be by node it will take this kind of inputs and it will generate this outputs by such and such computation. Once you define that in a diagram then use it in the main diagram again, again, again in multiple places the same copy it is copied here, here everywhere. And you know depending on the problem at hand the various edges will form connecting one node to another node and that entire DG will come up all right.

(Refer Slide Time: 06:28)



y(n) = b₀x(n) + b₁x(n-1) + b₂x(n-2)
switched on at n = 0 ⇒ x(n) = 0 for n < 0
y(n) = 0 " " "
Multiplication and accumulation (MAC)
b₂
b₁
b₀
y(0)
y(1)
y(2)
y(3)
y(4)
x(0)
x(1)
x(2)
x(3)
x(4)
n
b' = b
x' = x
s' = s + bx
b₀x(0) = y(0)
b₀x(1) + b₁x(0) = y(1)
b₂x(0) + b₁x(1) + b₀x(2) = y(2)

Let us take an example consider again the good old FIR filter. Theoretically, it should be from n equal to minus infinity to infinity and this summation also and ideally should be not only just three terms xn n plus 1, n plus 2 like that does not matter if it is non causal or causal n minus 1, n minus 2, n minus 3, but here we are taking only three terms and also for our convenience we assume that it is switched on meaning.

And of course if there is no input there is no output so this is also 0 for this. Then we will consider the implication of the actual equation where xn is we are starting from n equal to minus infinity and going on and now also we have got infinite such terms not only just three terms xn minus 1 xn minus 2, but that is later. Now let us look at this problem, what is the basic operation we are doing or say this there is a build in constant that times and incoming data output so multiplication then adding with something.

So this we do first adding with 0 so multiplication into I mean b0 into this or multiplication plus 0 this are result. Next time again I mean same job multiplication with a built in constant multiplication of a data and then add with a data earlier it was 0 now it is data and then whatever is generated now you come here again a built in constant time a data plus whatever is generated.

So multiplication and then multiplication output with plus, plus with something already generated, so multiplier and accumulator multiplication and accumulation MAC. So we define a processor first here a processor node, processor node how to define it, how to orient the various lines. See orientation of the lines like this is vertically up and (())(09:28) horizontal all this come out a practice orientations we choose matter a lot.

But they come out a practice there is no systematic way of explaining how to do it just comes out a practice. So I define a node processing node basic processing node that has three input incoming edges one is this, one is this, one is this and three outgoing edges one is this, one is this, one is this. So it works on the incoming data this, this, this does something and generates one output here, one output here, one output here.

How to explain, how to define, we just give a name to this variables. So let us call it b or let it be b prime let us call it x and let it be x prime let us call it s s prime then we will give a definition whatever b comes in the same b is pushed up is moved to the right. So b prime is same as b that is b just passes through very sorry for this there is a problem of hi-tech even if I do not touch the board if I go close to the board it just gets erased.

Similarly whatever x comes in that passes through and you got x prime so x prime is x and in the case of s prime is the actual s for summation. So incoming thing plus b into x. Like b into x plus incoming thing this is like s and b into something. So b into x so s prime is the result of this together is s plus b into x this is how I define our basic processor then I define then I draw the array like this is my 0 0 location a (())(11:59) axes though I am writing some line here I am actually axis actual axis will be here and here this is my 0 0 location I put a bubble here.

So this is one node located at 0 0. So here 1 minute I have a node here in 0 0 location then I put a node in 1 0 location one in this direction and let me give a j so 00 there is n and n 0 j 0 this is at origin, this is not the origin this is reference for my drawing it is the actual origin and 1, 0 1 means n goes up this n direction n goes up by 1 this represents 0 1 0 location then I draw here two 0 location, three 0 location goes on.

I would have had on this side also, but that I am coming later that I will share later for that time being we will have here why here that when the array is formed you will see little for the function of this a dot, dot, dot. Similarly, another integer for that coordinate these are all integer coordinate points 0, 0, 1, 0, 2, 0, 3, 0 so the both the n value and j value are taking integers 0, 1, 2, 3 like that.

Similarly this is 0, 1 n is 0, j is 1 this is 1, 1, n is 1, j is 1, 2, 1, 3, 1, 4, 1 dot, dot, dot and then again I have got 0, 2, 1, 2, 2, 2 like this. Then I give input say xn now whatever xn goes in that will by this definition that will go up and there is an incoming edge from bottom to this node and they will join that is from this node there is an outgoing edge and from the node above there is an edge from the bottom those two get join at one common edge forms.

Similarly we will give rise to one edge outgoing and thereby incoming and they get added like that. Similarly here I give x0 here I give x1 I give x2 as which as I told you switched on at n equal to 0 so all data before x0 are 0 that is why the left side is not there, there is no x minus 1, x minus 2, x minus 3. There is only x0, x1, x2 that is why I am getting on this right hand side and j we will very soon show you what's the meaning of j let me complete this.

I hope you are understanding that there is a bottom edge these are edge above. I guess there is a bottom edge and they (())(16:57) this two edges add we join. There is x4 and dot, dot, dot, dot. On the b side there is that edge these are I give the coefficient b0 as you know whatever b comes in that passes through the same thing comes out, same b0 will comes out, come out there is a outgoing edge of this node and there is an incoming edge of this node.

Incoming here and outgoing here and they join so form one common. Same phenomena here and it goes on that means same b0 moves in the same cycles same b0 moves through all of them I mean it goes to all of them then here I will have the same thing but now b1, this is b2. So understand j is nothing, but this index of b. b0, 1, 2 so j0 here j1, j2, 0, 1, 2 I do not have j minus 1.

Because that means that I would have xn plus 1, n plus 2 but those terms are not here because I will consider a causal FIR filters, but theoretically it should go to from b0 to b minus 1, b minus 2, b minus 3 up to b minus infinity and b1, b2, b3 up to b infinity and the same thing and xn also should have been they are from n equal to minus infinity onwards to plus infinity that is n should go from minus infinity to infinity j should go from minus infinity to infinity.

The entire plane I would have had these nodes at all the integer coordinate points remember all are integer coordinate points, but in practice we are not doing it we are making it causal that is why left hand side is not coming all through we are switching on n equal to 0 that is the bottom half is not coming only the first quadrant coming, but ideal array ideal DG would have assumed the same kind of pattern all throughout.

From n minus infinity to infinity j minus infinity to infinity and all integer coordinate points we filled in just by one node and do nothing the various incoming, outgoing edges join and the array is formed. I have not yet done the s so let us start with the s. I will fix the value 0 here I will tell you why. So this is outgoing as the incoming and they join again outgoing, incoming they join so on and so forth.

Let us start at n equal to here this node. What is he doing x of 0 going in b0 going in, b0 into x0 plus incoming 0 that will come here. So what is coming here is b0, x0. Now what is y0, y0 means b0 x0 and then b1 x minus 1, x minus 1 is 0 because it is switched on at n equal to 0, x minus 2 is 0. So that means y0 is nothing but b0 x0 that is why I will give 0 here so previous computation from this side is yielding 0 with that b0 x0 only b0 x0 I am getting. So what I am getting here is y0.

Now the same x0 moves up b1 here so b1 into x0 plus 0 that comes here b1 into x0. Now b0 moves out and x1 so b0 x1 plus b1 x0. This was y0 what is y1, y1 will be b0 x1 b0 x1 and b1 x0 nothing from here because it would be b2 x minus 1 which is 0, so this is equal to y1. So y1 is coming here. Same x0 moves further up here I have got b2 x0 plus 0 so b2 x0 that comes here same b1 is moving b1 end of same x1 moving.

So, b1 x1 plus that so b1 x1 plus that result is here. Now x2 at same b0 moving so b0 x2 this will be y2 you can easily see if it is y2 b0 x2 then b1 x1 then b2 x0 so it will be y2. In the same way you can verify this will be y3, y4 like that. So this is a DG dependence graph you see all connections are local whatever will be the pattern in one place there is this pattern you design a mask for it.

And when you design the (())(23:45) just put the mask here, put the mask here, put the mask here, put the mask here, mask here and all the incoming, outgoing edges join together and this very regular structure comes. There is nothing happen in this structure, nothing irregular that also is very, very useful very good for VLSI design and all the integer coordinate points 0 0 0, 1 0, 2 0, 0 1, 1 1 all the integer coordinate points you put that processor.

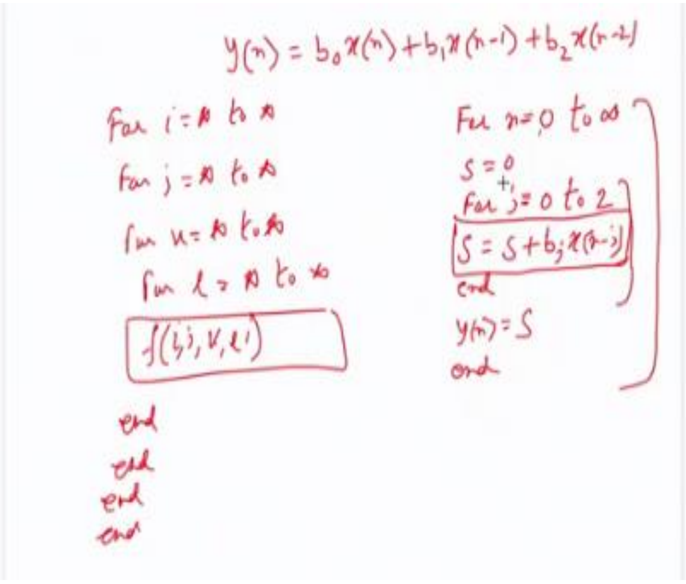
All the edges they get added they get joined they get joined they (())(24:15) depending on the orientation this comes up. Now while doing that ideally I should have that entire plane from n minus infinity to infinity and j minus infinity to infinity should have put this I mean nodes, but here there is a departure I mean this should have come from another processor here, but instead of that I am forcing the value to be 0 this should have come from another processor here.

Instead of that I am forcing the value 0 because I mean there is nothing beyond x0 if there is a processor here that would have taken x minus 1 but x minus 1 is 0. So that is why I am taking forced it get to be 0 force it get to be 0 like that because initial value is 0. Similarly, I am not going below this, but ideally I should have had b minus 1 xn plus 1 plus b minus 2 xn plus 2 like that so I should have more lines x0, x minus 1, x minus 2, x minus 3 up to x minus infinity x1, x2, x3, x4 up to x plus infinity.

And b0, b1, b2 up to b infinity and b minus 1, b minus 2, b minus 3 up to b minus infinity that is the entire diagram entire plane just hit all the coordinate points integer coordinate points put this processor there happily and the edges will join hand in hand at the entire actual DG will come up. It is a truncated DG there is a DG out of which we have done some kind of further restrictions we have enforced the manipulations like 0, (())(25:44).

And then I am not allowing it to go below I am not allowing to go to the left and so on and so forth it is a 2 dimensional DG. As I told you drawing a DG is not an easy job it comes out of practice. What is the meaning of DG actually? I mean I can come back to this page. Suppose I have to write a program computer program to evaluate yn equal to b0 xn plus b1 xn minus 1 we write a program.

(Refer Slide Time: 26:25)



$$y(n) = b_0x(n) + b_1x(n-1) + b_2x(n-2)$$

```
For i = ∞ to ∞
 For j = ∞ to ∞
  For k = ∞ to ∞
   For l = ∞ to ∞
    f(i, j, k, l)
 end
 end
end
end
```

```
For n = 0 to ∞
 s = 0
 For j = 0 to 2
  s = s + b_j x(n-j)
 end
 y(n) = s
end
```

What should be the program that is why I have for your sake I am rewriting that equation your program will be computer program all of you know for n should have been from minus infinity, but as I told you we are enforcing the fact that system is switched only at n equal to 0 and before that everything was 0 that is why we are not going for ideal DG we are going for a truncated DG that starts at n equal to 0.

And can go up to infinity for n equal to 0 to infinity have a value S equal to 0 they have an inner loop this is a computer program I think this does not require any explanation or j equal to here 0 to 2 new S will be old S this is the MAC operation old S plus bj xn minus j end then yn will be this S end. We can easily see, we can easily verify this start at n equal to 0 and S is 0, so if n is 0.

What is happening at j equal to 0, 0 minus 0 so b0 or maybe for any general n yn equal to 0 taken general n S is 0. So now get into this slope start with 0 here S is 0 so new S will be b j equal to 0. Now b0 xn so b0 xn next time j equal to 1 so it will be b0 xn j is 1 so b1 xn minus 1 so b0 xn plus b1 xn minus 1 together will be new S next time that plus b2 xn minus 2 that will be a final S that I take to be yn so this is the program.

So how many loops are there at two loops? One is inner loop another is outer loop. There is one loop index n another loop index j and this is the basic processing. You will see my DG was 2 dimensional DG one axis is n another j n was taking various integer values, j was taking various integer values and for a particular n, j pair there was a processor of this kind incoming S plus multiplication and accumulation incoming S, incoming b, incoming x.

They are multiplying adding with incoming S new result. So that means typically we have something like this for suppose I take an example for i equal to some value to some value some star to star. Then again for j equal to another star to star, star mean some value to some value I am not bothered about that maybe I have got another one k equal to some value to some value but integer values.

And maybe for l equal to we have the basic operation where we evaluate like here we evaluating S we are evaluating some function we have a function of i j k l that evaluate by a formula by some calculation like this and then end, end, end, end and a example. So here you will have four dimensional DG one axis will be i, another will be j another will be k another will be l.

And they all take integer values as you mean computer program you write for i equal to maybe 1 to 10 you do not have i equal to 1.5, 2.5, 3.6 not like that they take all this loop indices take integer values. So i take integer values j take integer values, k take integer values l takes integer values and for each (())(30:40) i one value j one value k one value l one value we will evaluate this.

So we will have a 4 dimensional DG one x is i, one x is j, one x is k, one x is l you design a basic processing cell that generates this. So depending on the integer value of i, j and k, l you identify that integer coordinate point put that processor and do it for all the i, all the j, all the k and the entire 4 dimensional DG will come up unfortunately we cannot draw it because it was 4 dimensional, but we have a 4 dimensional DG.

This tells you about the dimension of the DG dimension is nothing, but number of loop indices to run the program each loop index is one axis number of loop indices to run the program. Here I had got 2 that is why there is one axis n another is j 2 dimensional, but you know it does not have to be 2 dimensional you can have any dimensional 10 dimensional, 15 dimensional.

Then a question is if it is a 1 dimensional DG I can directly implement it on IC it will be one string one line only that is a called string if it is 2 dimensional it will be plane that will be called a planer IC that is plane up an array on a plane if it is a 3 dimensional it will be called a cube, cubic so that every integer coordinate points in the cube we will have some processors cube, but if the dimensions goes beyond 3, there is 4, 5, 6, 7, 8, 9.,10 you cannot realize in IC.

Because the whole world real world is 3 dimensional. So you cannot realize a 4 dimensional DG on a IC or a 5 dimensional DG on a IC then what we do is we map a higher dimensional DG to a lower dimensional DG, lower dimension can be 3 or 2 or 1 that mapping is called systolic transformation which we will consider I mean if the time permits we can consider later.

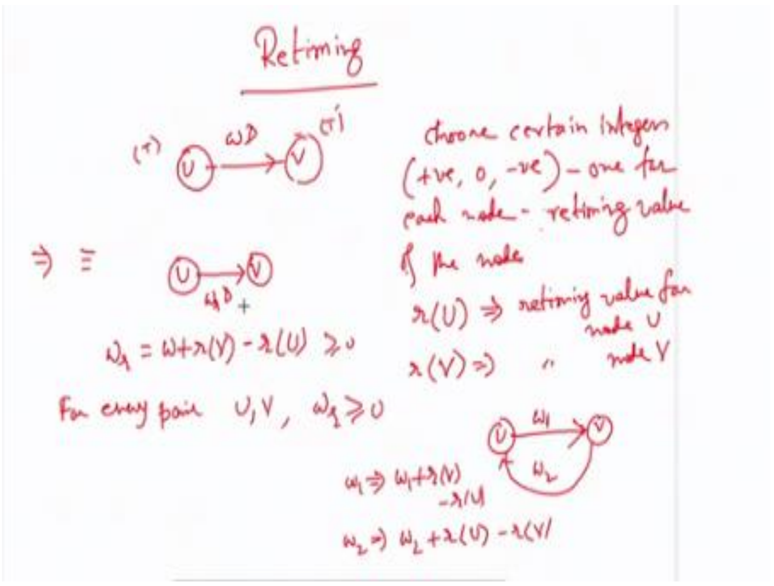
That systolic transformation by which we map a high dimensional DG to a lower dimensional DG and so that lower dimensional DG as you know is realize because its dimension is 3, 2 or 1 realizable that is a entire philosophy of array architecture DI. So this much we have for the, this thing for this graphical representation of DSP algorithms. SFG we started with then we covered DFG we cover a very important topic called critical path there and also told you about retiming.

And then separately we have come to another interesting architecture which is very useful another interesting thing graph which is very useful for a particular kind of architecture called array architecture that is dependence graph so that covers the first topic and then as I mentioned about retiming. This is what I will be spending (())(33:37) time on and there is something called retiming theorem which I could skip but I would not I feel that you know I should prove that theorem in the class.

You have to bear with me because then only we can get an idea about I mean why we follow this particular retiming formula when it comes otherwise it might look little boring as to where from a particular retiming formula is coming. So I will spend some time on retiming theorem proving and of course then from retiming theorem various things follow and slowly we go to a practical way of doing retiming using of course retiming theorem.

We cannot value retiming theorem at that practical way is called cut set retiming and then we take some examples and show you how to do retiming to achieve some particular critical path target that takes me to the next topic.

(Refer Slide Time: 34:34)


Retiming
choose certain integers (+ve, 0, -ve) - one for each node - retiming value of the node
r(U) ⇒ retiming value for node U
r(V) ⇒ " node V
$w_1 = w + r(V) - r(U) \geq 0$
For every pair U, V, $w_1 \geq 0$
$w_1 \Rightarrow w_1 + r(V) - r(U)$
$w_2 \Rightarrow w_2 + r(U) - r(V)$

Retiming means you are giving a data flow graph and there are various edges some edges having certain delays some edges are not having delay free you can calculate a critical path also. Now retiming tells a way by which you can alter the delay values in various edges without violating the functionality of the original graph that is it will be new one will be equivalent to the old ones in terms of the output that it generates for a particular input.

As I told you equivalents maybe little loose here it still equivalent very much equivalent, but it may not be very rigid equivalent that is a particular node before retiming and after retiming may produce a same output but the two outputs can be delayed version of one another with a known delay. So my appropriate shifting for order backward from one you can get back the other because the delay is known.

So it is equivalent in that sense of course it is equivalent because I am not losing any information if I know the current output if I know if it is delayed by say delayed from the original one by say 5 cycle I will move it up by 5 cycle I can get back the original one because 5 is known to me. Similarly something scaled up like a node is generating a sequence xn after retiming if it generates 10 times xn if I know the factor 10.

So from the current output if I divide by 10 I can get the original one so it is equivalent of course very much equivalent, but there is little flexibility there because if you do not allow the flexibility then things becomes very tight and you cannot do many optimization techniques like retiming and all. So with that consider a DFG any part of DFG will be typically of this form it will be one node another node will be at edge.

We have a delay we have not have a delay so I write w actually w times D w can be 0 means 0 delay 1D means 1 delay 2D means 2 delay like me call it U let me call it V. You may write some amount of time T here T so T microsecond spent here T microsecond it means T it can be T prime also, this is given w amount of delay. This is a typical sector of a DFG a DFG is nothing but repetition of these things.

One node and edge comes out of it hits another node with or without delay again from here another edge comes out hits another node with or without delay and goes on everywhere as the entire DFG comes up. So I do sector wise so I take the basic sector. Retiming theorem says that choose integers. It can be positive 0 or negative, but this is purely your choice you are absolutely free.

You can choose certain integers one for each node, one for each node. So assign you choose one integer assign it to one node, you choose another integer assign it to another node choice is yours positive, negative 0 no problem. This will be called retiming value of the node. Like if I choose value 2 and give it to you and node will have retiming value 2 and so on and so forth.

So u denote by r, r for retiming r bracket U as the retiming value there is an integer for node U. Similarly r of V retiming value for node V these are the integers you have chosen then retiming theorem says that this will be equivalent to same U same V same connection, but delay will be w r times r for retiming new values where w r will be original w plus retiming value of the destination node where the arrow is hitting.

That is r of V minus r of U source node source in this sense there are limited context in this for the edge this is the source this is the destination. So retiming value of destination minus retiming value of original source with that original delay that will be wr, only constant is if there is a negative sign you should choose this value such that this does not turn out to the negative that is you must choose rv value such that for every pair the wr value wrote that it should be greater than equal to 0.

Because negative number of delay has no meaning. It might appear to be something easy to you that why not increase rv takes less value of this so that this one is positive, but if you have a loop suppose this has w1 this has w2 so now this will become w1 will become equal U V w1 will become w1 plus because this arrow is here r of V minus r of U, but in case of w2 arrow is hitting this side.

So this is the destination this is the source should be reverse of this w2 plus r of U because this is the destination minus r of V so if you want to make r of V very high r of U less so that you are happily you know in a situation where this is positive you are very happy (())(41:26) this is positive because you have taken r of V to be high needs to be low very good so this is positive, but then there will reverse effect here, r V very high means this will be highly negative r U very less means very less positive.

So this by trying to make this positive you may end up by making this negative which is also not allowed so it is not an easy thing for every pair when you apply this inequality then this should be greater than or equal to 0 for every pair of connected nodes not disconnected nodes. Thus for everywhere you pair up nodes that are connected by an edge we sub delay a w.

Every pair of nodes that are connected by an edge there is a delay w you have to ensure this. Then you get a series of inequalities whereas inequalities you can give it to a computer if the inequalities have a common solution then it will give a solution space you can pick up any choice of r U, r V and like that for all the nodes and that changed the value from w to w1 for every edge and you get a retime circuit.

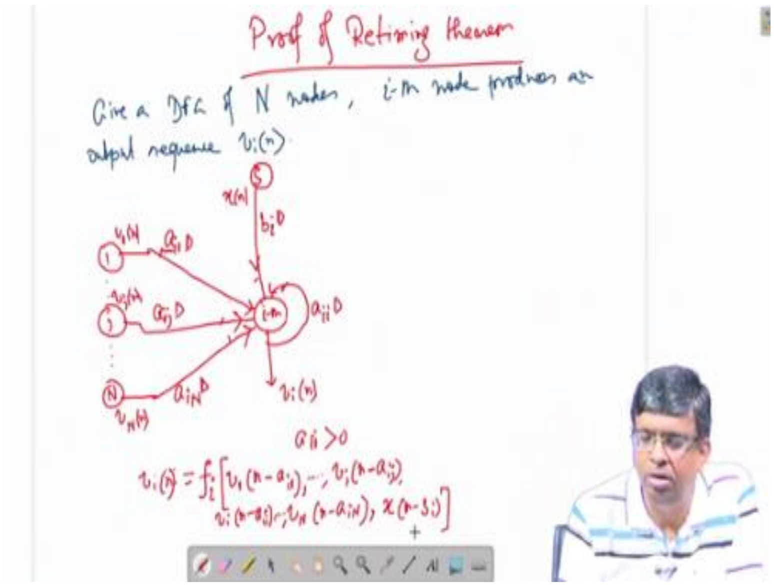
If it is retimed it will be equivalent, but remember node V will produce an output here now that will be not same as the original node V output in the sense this one will be delayed by this will be nothing, but delayed version of the original one that comes out of V here it will be delayed by how much time r of V there is a retiming value chosen here. Similarly here if it is node is generating some output here also it will generate same output.

But it will be delayed version of this delay by r of U So every node output will be delayed by its retiming value, but since retiming value are chosen by you, you know them since retiming values are chosen by you, you know them. And therefore there will be equivalent.

Because from one delayed one you can get back the original one. Now how it comes up first let me given there is a bit lengthy which involved proof it is called retiming theorem I am going to be retiming as proof and it is not given in books I worked for a proof so be with me because I still feel that I mean I should give a proof so that you really understand where from it comes you are convinced that this works.

Then we will start trying to understand the implication of this result it’s you know I mean various possibilities, pros and cons other things and then how to give a particular kind of retiming where we will follow this kind of retiming theorem but choose this retiming values in a particular way leading to what is called cut set retiming and use it to retime certain circuits and I will take up some examples of retiming also. So that will be the proof.

(Refer Slide Time: 44:16)


Proof of Retiming theorem
Give a DFG of N nodes, i-th node produces an output sequence $v_i(n)$
$x(n)$
$b_i D$
$v_1(n)$
$a_{i1} D$
$v_j(n)$
$a_{ij} D$
$a_{ii} D$
$a_{iN} D$
$v_N(n)$
$v_i(n)$
$a_{ii} > 0$
$v_i(n) = f_i[v_1(n-a_{i1}), \dots, v_j(n-a_{ij}), v_i(n-a_{ii}), \dots, v_N(n-a_{iN}), x(n-b_i)]$

Suppose you are given a DFG end nodes node 1, node 2, node 3 up to node capital N, i th node produces a output sequence Vin. So node 1 produces V1n node 2 produces V2n dot, dot, dot, dot node i produces Vin and they are all connected it is DFG of total N nodes. So if you focus on a particular node say i th node it will be like this, I take the i th node as my center of focus it produces an output Vin.

Other nodes are here I write separately node 1, dot, dot, dot node j dot, dot, dot node N alright. There may be or may not be a connection from node 1 to node i to be general I make provision for a connection edge again if there is an edge there maybe 0 delay or some delay. So I keep a provision of some delay a since it is going from node 1 and going to i I want to bring subscript where it should be both i and 1 should be present and I write this way ai 1 number of delay.

So ai 1 is an integer either 0 or positive that many number of delay ai 1 which goes from node 1 which is present in the edge that goes from node 1 to node i. If there is a 0 delay we just happily take this to be 0. So 0 into D 0 so this will be delay free edge otherwise if it is 1 delay this will be 1 this will be 1 D, 2 D, 3 D if there is no connection from node 1 output to i th input in that case you laid this ai 1 go to infinity.

Infinite delay means signal from this side can never reach this side so if you want to make it I mean you want to make sure that there is no connection in all the math's we do we will let this a fellow ai 1 go to infinity alright. Now similarly from everyone I am taking a possibility this will be aij D again this can be 0, 1, 2, 3 or infinity, infinity means no connection. This will be aiN so i is the first index in all of them that many delay alright.

So this is this fellow is node 1 so it is also generating v1 N because if node i generates Vi N node 1 V1 N this generates vj N this generates v Nn. This itself can depend on there can be a feedback from its output to itself because it not only may depend on other node output it may depends on its own output and in this case the delay will be ai the destination node again i there is a source node destination i here source from here i aii D, aii D can never be it has be greater than or equal to it has be greater than 0 it cannot be 0 has to be greater than 0 it cannot be 0 simply because suppose it is 0 because it is a delay free path.

As from Vi N I want to find out Vi N that will depend on Vi N itself because there is no delay path. So output to calculate output by this node I need the same output at the input so that is not possible because output has to be calculated first.

That is why if it is one delay no problem Vi N depends on Vi N minus 1 which was calculated here in the previous cycle and now made to appear here. If it is Vi N minus 2 here no problem it is too delay, but if it is 0 delay it is a delay free loop is absurdity because your output goes back to the input without delay so output depends on output. How to calculate the output we did the same output as input.

So if output is not known I cannot calculate so it is a contradiction that is why this must be greater than 0. If there is a (())(49:44) the corresponding I mean delay must be greater than 0 and then there may be an external signal source I call it S. So there may be an edge coming from there also with some delay bi. Here I am writing only i I am not putting any second index because here the node is fixed in a particular node specific node signal source node.

It does not have any index like 1, 2, 3 so but when I look at the figure b, b means edge in the path in the edge from source signal source to i th node that denoted by b, others are by a, a for small a coming from node to node edges. Small b coming from external signal source to the node that edge i because it is going to i th from external source S bi times D, if bi is 0, 0 delay if b1 is infinity no connection.

That means this particular node does not take anything from external signal source. What it does is i th node takes all this fellows all the information including this and does some operation to calculate Vi N that entire operation it can be smaller operation it will be large complicate operation that I am not going into that is all inside the node that entire thing I am repeating as a function that is Vi N it evaluates the function for the i th node.

Some function which you know very well what it does inside because you design the node function it takes all this fellows at input what is this fellow if it is V1 N it will be a1 i1 cycle delayed version of that. So V1 n minus ai1 dot, dot, dot, Vj n minus aij dot, dot, dot v capital N small n minus aiN, if want you can write this one also Vi itself n minus aii dot, dot, dot and lastly if it is generating as sequence xn, xn comes with a delay here.

This is some particular node i that is the function, but then I could be 1 so node 1 would come here i could be 2 node 2 would come here I am just showing at the center of focus so I am just concentrating general case. So these are the delayed things that are coming a delayed signal and then node this node does some operation on that which I mean I include in this function fi if I just to indicate that net operation done on that and output comes.

We will then apply retaining formula on this change the delay values by the formula after choosing some retiming values for this nodes. We will get another structure like this and will show both are equivalent that will be done in a next class, so please come prepare with this. Thank you very much.