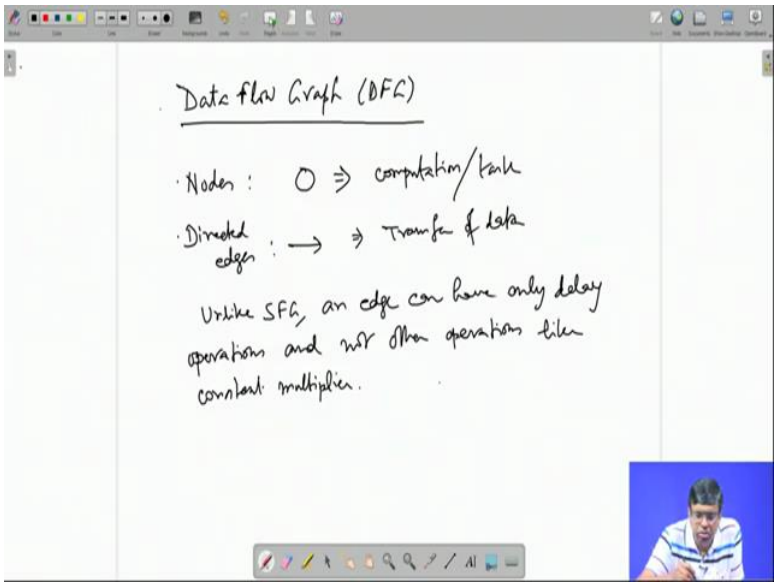
Course on VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture 03: Data Flow Graph, Critical path

(Refer Slide Time: 0:32)



Okay, we will now in the last class we covered signal flow graph. So now today's topic is a next graph, which is data flow graph. Like the SFG, this also consist of two things, one is nodes indicated by bubble which is nothing but what I had earlier computation, task, etc. Another is directed edge, directed edges transfer of data. That is it starts at a node, takes the data output of the node, gives it to the node where the arrow is heading to.

But the basic difference is this, unlike the SFG, unlike SFG, an edge, this directed edge can have only delay operations and not other operations like say constant multiplier. Yesterday we have seen that, in the last class, we have seen that in the case of SFG an edge can have by side either delay or constant multiplier. In the case of DFG constant multiplication is gone. If there is any multiplication, that has to be put in some node as one operation.

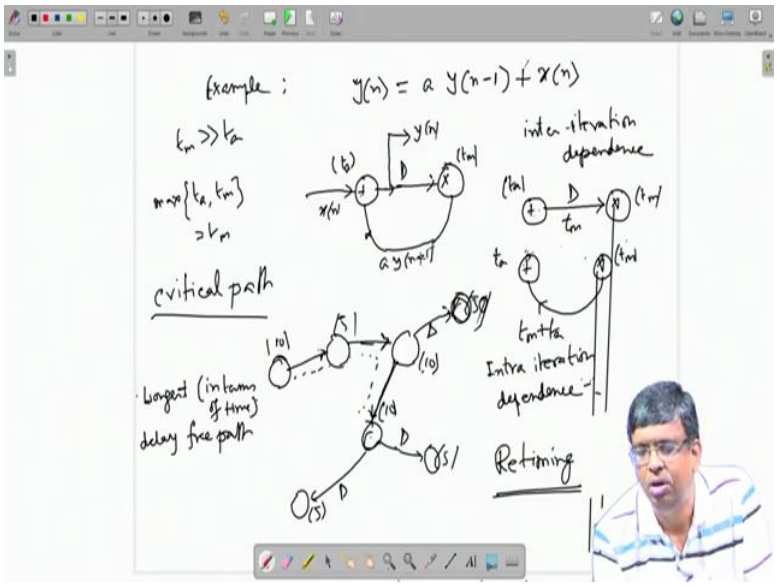
Along the edge you can have some delay only. That is technically the difference, but then the (())(2:48) there is a difference between DFG and SFG. In SFG is more like the circuit like yesterday you have seen the SFG for, FIR filter is more like what we study in DSP books, but when you look at DFG the circuit does not come to your mind first. You look at the entire job to

be done per clock, you divide the job into various sub jobs, sub tasks, and dedicate one node for each sub task and the sub tasks the nodes are dependent on each other.

One mode output is required may be in the same cycle or in the other cycles at the input of other load and vice versa. So kind of network comes up and that is what data flow graph is. So, here you divide the job into various sub jobs, dedicate one node for each sub job and the nodes depend on each other.

So they are by the directed edges form without delay, depending on whether you need this transfer in the same delay or cycle or subsequent cycles. And that internetwork which is called data flow graph, DFG, this comes up. And DFG is most commonly used and this course will be using DFG again and again, every now and then for describing the various optimization techniques.

(Refer Slide Time: 4:04)



An example, consider IIF filter equation. First of all IIF filter, all of you know, xn is the input yn is the output. So what are the jobs involved for output calculation? That is for every yn calculation, if there is any cycle you are calculating what are the jobs you need to do in that n th cycle? One addition between one component and another component and before that one scalar multiplication.

There is constant multiplication of some data, some data into a constant multiplication that means the multiplier with a built in constant a will be required, the output of that multiplier to be added with another data. So, two operations multiplier with a built in constant a and a single adder. One signal, signal is anything function of n is a signal. So signal plus signal. So, we dedicate two nodes for it.

This is maybe the adder node and this is the multiplier node. You have xn coming if you want you can put a source node here but not required, that does not serve any purpose. Now after addition, so adder will take xn and another component which is this. So, this component may come from this side, a into yn minus 1. Suppose it is generated, so this output should be yn. So, you take it out suppose it is generated as yn.

Now, we assume that till the previous cycle n minus 1 cycle, we could generate here correct output y. There is still yn minus 1 or you could generate correct output here. So, suppose I put a delay down and I am in the n th cycle. So, what will come here is what was generated in the previous cycle? n minus 1, n minus 1th cycle and by my assumption, till that time whatever was generated that is correct yn minus 1.

If that be then that correct yn minus 1 is now coming here, getting multiplied by a, the built in constant and coming as the output. So, what you get here is a into correct yn minus 1. And if this correct, a into that loss xn will give you correct yn and the cycle is probably goes on correctly. Okay that is how it works. So we assume that in previous cycle yn I mean yn minus 1 will united correctly. In the n th cycle this comes here because of a delay.

So, yn minus 1, the correct value comes here, gets multiplied by a, comes out here, gets added with xn. So, by this equation whatever comes out that is the correct yn so correct yn comes out and this goes on from n to (())(07:32) n plus 1 to n plus 2 to n plus 3, I mean cycle onwards. This is the DFG. Now in this DFG, you see one thing. Just consider this part only, xn coming in or you are typing something for the output that is immaterial.

Consider this part, look at this multiplier node and look at this node. If you see this part, the upper part, and there is a delay. So here this mode depends on this node output and a node output, but not in the current cycle. So we say this node, a multiplier node is dependent on adder

node output. So there is a dependence relation between multiplier node and adder, multiplier depending on adder. But it is not in the same clock cycle, same iteration cycle.

Whatever this fellow generated in the previous cycle that only I need here as a multiplier, not what this fellow is generating in the current nth cycle, which means that the dependence relation is of the type inter iteration, that is inter clock, not in the same clock, inter clock.

Dependence, inter iteration dependence. That we have seen one cycle it does a job. May be taking amount of time ta. In fact, it is preferred that you know in a DFG, by the side of each node you write within bracket sometime like ta, the time ticket by the adder, whatever it is maybe be 1 microsecond or 5 microseconds whatever you put here and that you put under bracket.

So, it will indicate that this adder takes so much time to carry out one addition. Similarly, here you should write tm. So, this means this fellow definitely needs the output of the adder, but not in the same cycle. So, in one cycle the adder generates an output, it is stored in the flip flop, in the delay and this goes in the next cycle here. So, they are dependence, dependence of multiplier on the adder is inter iteration type that is internal clock type.

What is the implication of that? In one cycle it does a job that is not required here in the same cycle. It goes here in the next cycle. And in the next cycle if there is n th cycle, that time it does a job simultaneously for transfer here is the n plus 1 th cycle. So it does not depend on these output in the same cycle which means they work simultaneously and the two times do not have to be get. It’s like a pipeline that two times tm and ta do not have to be added because as I told you and tm is as you know, multiplier time greater than ta.

So in one time period of say duration tm, it does a job of addition, gets a result, puts it here and sits idle. That time it is doing some multiplication on what came from the previous output. So this also takes tm time and it takes ta and then waits for the remaining part of the tm amount of time. So, I required just tm amount of time to generate the result here. Then the next cycle whatever it generated previously that will come here.

This will work on that and that time it will work on new data and again take ta amount of time and then produce the result, put in the flip flop, put in the delay and sit idle. So, since tm greater than or equal to ta, it is much greater than ta, together I mean they simultaneously work on tm

amount of time here, tm amount of time here, that is your pipelined, the two times do not get added.

And that is why it will take tm amount of time and that because the dependency is inter iteration, but see the lower half. Here, arrow direction is from this side to this side and there is no delay here. So the adder in this case, earlier the multiplier depended on the adder and now it's the reverse. Adder depends on multiplier output. And since there is no delay, that means in the same cycle first I have to carry out the multiplication job taking time tm, then on that only, in the same cycle, I have to carry out the job of addition, that is ta.

So here time taken will be tm plus ta. And here taken time taken will be tm. Why tm? Because maximum between the two node computation time, so one is ta, another is tm which is of course is tm, maximum between the time between this and this is tm because tm is a multiplier.

So I just need tm amount of time. In one period of tm duration it will work on what was generated from previous cycle here. At that time it will work on new data and take tm amount of time only, put it in the delay and sit idle for the next remaining part of tm. So in tm amount of time only both the jobs are done. It is preparing an output for usage here in the next cycle, not in the current cycle.

So, I just need tm amount of time to do the job, but here it is not. In the same cycle first you dedicate tm amount of time to get this output, then this goes here, then another ta amount of time is spent, so total is tm plus ta. Because there is no delay here it is called inter intra iteration. That is within the same iteration, within the same clock, intra means clock intra iteration dependence.

So together, when you put them together in this diagram, I have to consider the largest between tm and tm plus ta because I have to give that much time which is required for completion of all the jobs. So its upper half takes tm but the lower half takes tm plus ta. That means to be on the same side so that both the half's are done I must give tm plus ta amount of time before I give another data. So, I send in one data then give tm plus ta amount of time for this network, this circuit or DFG to do all the job, the upper half an lower half and then you send in another data.

So, that is what I am saying that I have to consider all the paths. Path means path or edge, this edge. This edge connects two nodes, this and this if I have only the upper half, since it is there is

a delay, the two jobs are not to be done simultaneously I mean, sequentially in the same cycle there is first then this. Larger of the two will be enough for me to complete both the jobs. One will take that much of time tm to do the job here, that much time will be more than enough for the adder to finish its job to be used in the next cycle here, not in the current cycle.

So that was tm, so if it is a delayed path, the two times do not get added. It means a non-delayed path, path or edge in this case edge, then the times get added because they come in sequence, in the same time in the same clock, they get added sequentially. First you do this, then in the same cycle take the output here do this.

The two times get added and then I consider the largest of them so that all the paths, all the edges are executed, I mean the upper half, lower half both are executed. Nothing is incomplete. This takes us to a very very important concept in this course called critical path.

First, what is your path? Path means sequence of edges. Like you may have a network, you may have a DFG. One node going here, another node going here, I mean one edge taking from this node to this node then this here, this may be here, this can go somewhere here, this can go somewhere here, this can go somewhere here.

So this is one edge, one path, this is one path. This itself is a path, the two constitute a path, the three constitute a path or this way it constitute a path, these two constitute a path, these constitute a path, these two constitute a path. Path means a sequence of edges a single edge also is a path. So, there are so many possibilities of path.

I told you this can be a path, these two can be a path or this itself is a path or these two can be a path or these three can be a path or this can be a path, this can be a path, this can be a path and like that. All right? Now, certain paths may have delay, certain paths may be delivery, certain paths may have delay. Suppose I start here I find a path edge which is delay free suppose.

That means these two jobs are to be done sequentially in the same cycle, this plus this. Then maybe this also is delay free, but this is delayed. That means in the same cycle I have to do this job, then this job, then this job, whatever is the output that will be required in the next cycle here, not in the same cycle. So the two times will not get added. But suppose I moved to this path, this side, this may be delay free.

With this first we complete, then go here complete this, then go here complete this, then come here complete this, there will be a delay. So this output I mean this much time is fine to do job up to here but output here is not required in the same cycle in the next cycle. So the two times will not get added. This may be also delayed. So the two times will not get added. In this case what we have here?

You find out I am finding out say the longest delay free path. Longest does not mean in terms of length in terms of time. So, this is one delay free path, this is one delay free path, together a delay free path, this is a delay free path, altogether a delay free path but I cannot go further, I cannot go further. I cannot go further because delays are coming. So, I have one delay free path, one delay free path, one delay free path, there is single edge. Two edges forming a delay free path, two edges forming a delay free path, two edges forming a delay free path and like that and three edges.

Obviously here if you call and then find out what is the longest in terms of time? Because in a delay free path all the nodes are to be executed sequentially. There is this out, there is the dependencies intra iteration dependence. That is in the same cycle first you complete this job, then in the same cycle send it here, this works on that, does something, then output in the same cycle further moved up here, it does something then in the same cycle further moved up here. That means the times of all these four nodes will get added.

So, that is our one delay free path, its computation time. Now, in a DFG, you may have several such delay free paths. So, first you find out all possible delay free paths. Even by the single node also is a delay free path. I mean single node time because that time also, if suppose this time takes much much more than all the nodes added together, then this will dominate.

My clocks would be governed by the time taken by this because if I allow this to do its job that will take care of all other nodes together. That is why every, you start with every single node, take its computation time. There is a smallest delay free path and then find out all such delay free paths by following the sequence of edges which are delay free edges. For each delay free path find out the computation time by adding the nodes.

So, if it is just a single node, the node competition time. If various nodes are coming in this delay free path, add their times. As I told you, every delay free path gives what? All the node computations to be done in the same cycle, they get added.

So, now identify that delay free path for which the net computation time on that edge, that is all the nodes added together is by a maximum. That will give you the amount of time that the clock should have at its period because if that is given then all that delay free paths computation time will be taken care of nobody will remain incomplete and therefore this can work. Therefore critical path is longest, longest means in terms of time.

Time is computation time, delay free path. Suppose I had 10 here, 5 here, 10 here, 10 here, 5, 5, 5. To each node 10, 5, 10, 10, 5, 5. If you add 10 and 5, 15. So this delay free path, takes 15. 5 and 10, again 15. But if you would add all that three together because their total is delay free, 10 plus 5 plus 10, 25. That may not be enough. Again if you go further, still it is delay free. So you can go on adding. 10 plus 5 plus 10 plus 10, so it would be 35.

So, you have 35 on one hand, you have 5 one hand. 5 one hand, 5 one hand, 10 one hand, 5 one hand and 10 one, so obviously 35 is the largest. So its critical path is this, you show by a dotted line around that. Suppose instead of 5, this node takes 50 then this delay free path takes 35 I agree. But this node still takes more than that. So my clock should be minimum 50 unit of time so that this job is done and since this takes 35, which is less than 50, this also is done and they are also done.

So, you have to take the longest critical path in terms of time and identify that as critical path. And the time taken by the critical path is called critical path computation time. But you see the sentence, critical path computation time, there are four words. It is a very long word. So, that is why today maybe in a loose way, the critical path computation time is also called critical path. That if I ask you, show me the critical path that is the orientation, you show by dotted lines the critical path.

And if I ask you, how much is the critical path? Then it is not the orientation, it is the amount. Then you add all the node times like on the critical path and that time will be called critical path. So, critical path is today used both to indicate the path’s orientation, its location, and also the

time taken by it depending on I mean as I told you, if I say show me the critical path then you its location, its position. If I ask you how much is the critical path then I am asking for the time taken by the critical path.

Then this question comes that given a DFG of certain critical path can we transform the DFG into another equivalent DFG? Which may have lesser critical path. So then the DFG will be faster and I can use faster clock. This comes under a very very important technique, which is kind of foundation of this entire course and that is called retiming.

I will spend considerable time on retiming. Retiming means, given a DFG with edges, some edges delay free, some edges having certain delays and all that it shows you a technique by which you can change the delay values in various edges, still the resulting DFG will remain equivalent to the original DFG.

But equivalent but then it may have a new critical path because delays around various edges have got changed. So, new delay free edge with longest time, which is called critical path may be defined from what I had earlier. So, retiming is a technique by which you can change the delay distribution on the edges without hampering the equivalence. This has a theory which I will covered and then it leads to a practical technique called (())(25:25) retiming that we will do after some time.

There is a key thing in this course, how to change the delays in a network in a DFG without of course violating the topology, the connections only the delay values of the edges change and change according to some formula given by returning theorem. So, that the equivalence is preserved. And then you try to change the delays of the given DFG so that critical path of the new DFG becomes lesser, shorter than the original DFG so circuit becomes faster.

Now one thing I will tell you. When I say equivalent, sometimes it may not be possible to maintain absolute equivalence like you know, I mean take one node output, I mean before retiming and after retiming the two node outputs maybe same, but maybe dissipated by a delay, that is previously suppose a node, it is generating a sequence say u of n.

If the same node now, after retiming generates u of n minus some small m where small m is known, then they are equivalent because if I know small m, I can just move the sequence u n

minus m forward or backward by m depending on m is positive or negative. And get back my original one because m is known. So, a delayed output may be generated with a known delay but since is a known delay, original output from this can be easily obtained.

Similarly, there may be a scaling with a known scale factor. There is originally or may have u n. Now it may be some 10 times u n. Now 10 is known to me. So if 10 is known, I will divide the output by 10 I get back original event. So there may be little deviation, but then it does not change the equivalence because I can always get back the original output sequence of any node from the new one without any error. I say equivalence in that sense, not in a very rigid sense.

So this is the thing of DFG and the critical path. So, here if you come, come to this DFG, what is the critical path? If you look at the this one, it takes ta. So ta is one component because every single node also is a delay free path. So ta, tm here they are delayed. So in this edge, the two times will not get added, but this is the delay free path. So the two times will get added, ta plus tm. So I have got ta, tm and ta plus tm of which ta plus tm is the largest.

So this is the critical path, I mean critical path time which is also called critical path is ta plus tm, that much is for DFG. From here in the next section, we will go to DG, dependence graph. Thank you very much.