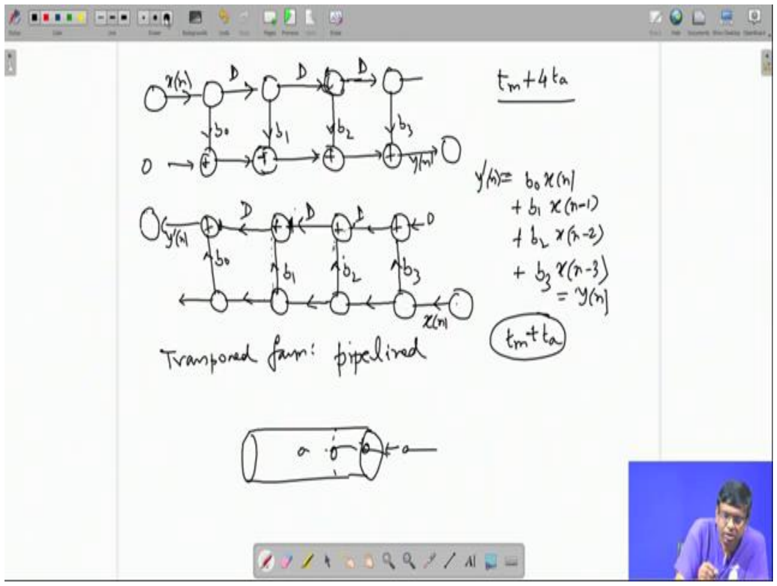
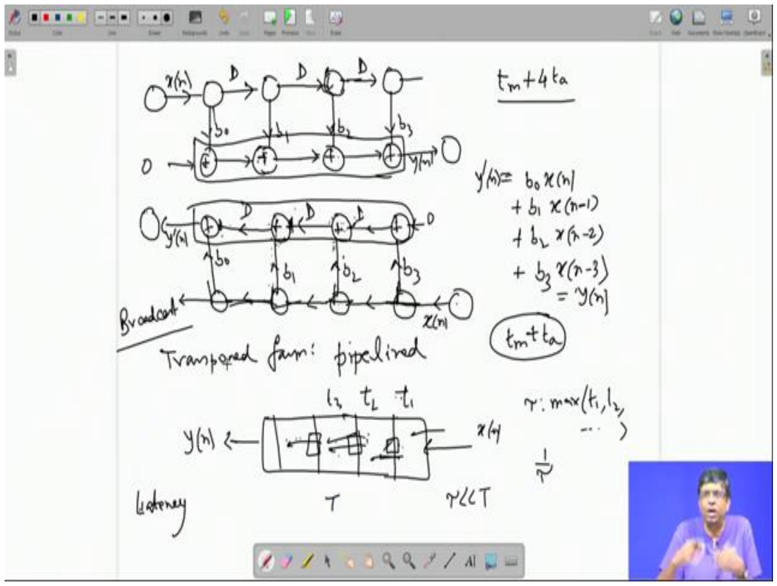
Course on VLSI Signal Processing
Professor Mrityunjoy Chakraborty
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture no 02
Signal Flow Graph

(Refer Slide Time: 00:28)



$t_m + 4t_a$
$x(n)$
D
b_0
b_1
b_2
b_3
0
$y(n)$
$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + b_3 x(n-3) = y(n)$
$t_m + t_a$
Transposed form: pipelined



$t_m + 4t_a$
$x(n)$
D
b_0
b_1
b_2
b_3
$y(n)$
$y(n) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) + b_3 x(n-3) = y(n)$
$t_m + t_a$
Broadcast
Transposed form: pipelined
t_3 t_2 t_1
τ: max(t_1, t_2, ...)
$\frac{1}{\tau}$
Latency
T
$\tau << T$

So, we applied transposition to this circuit. So as I told you all the arrows of the edges have to be reversed. Source and sink do not have to be interchanged and splitter will become adder and adder will become splitter. As that we have yes because splitter it takes one input and gives two outputs right. If the arrows are now changed, here I had 1 incoming arrow and 2 outgoing edges.

Now if the arrows are reversed there will be 2 incoming 1 outgoing that cannot be a splitter that can be adder. Adder takes 2 input produces 1 output.

Splitter takes 1 input produces 2 output right. Similarly take an adder, it takes 2 input 1 output. If I reverse the arrows then this will be again a single output and there will be 2 output. So, obviously it cannot be an adder, adder cannot take 1 input and give 2 output, it will be a splitter.

So, this was my source, this was my sink. Now, this will be my sink this will be a source this becomes a splitter this becomes an adder, splitter arrow reversed but b 3 multiplier will remain in the same place. Only it was a splitter now adder this arrow reversed but D will remain here same place. Then again a splitter previously it was adder now a splitter, arrow reversed, I will have b 2 arrow reversed I put the b in the same place, the delays is not multiplied they do not change.

These are goanna splitter arrow reversed here arrow reversed I got b 1 then again a splitter these arrow this D b0, let it go somewhere I do not care or this is buying sink node. Now question is I give xn as input here, this are source. Suppose I get y prime n. question is, is y prime n is same as yn, if they are then the two circuits are same because equivalent because for the same input I get same yn equal to y prime n.

So, that we can easily see look at this edge, these are your splitters. So, xn comes here goes to this direction, goes to this direction, same xn goes to this direction foes to this direction again same xn goes to this direction this direction and b three times extend here, b 2 times extend here, b 1 times the same extend here, b 0 times extend here.

So, is b 3 xn if I take that component only here b 3 xn with 0 so is a b 3 xn, after 1 cycle delay it will become b 3 xn minus 1, it will get additive something, forget that something, I am only considering only that component. So b 3 xn only that component I focus on for the time b. So, if I take out that component that will go here with b 3 xn minus 1 get added with something but forget that something. This will be b 3 xn minus 1.

Another cycle delay, b 3 xn minus 2 another cycle delay b 3 xn minus 3, so that will come as b 3 just a minute give me a minute, so that will come as b 3 xn minus 3, then here xn, same xn came multiplied by b 2 that was getting added now I take that separately, this b 2 xn part will get will pass through these it will become b 2xn minus 1 again pass through these it will become b2 xn minus 2.

Then the extend b 1, beyond xn it will go through these, b 1 xn minus 1 and then xn goes to this b 0, b uxn, why plus because all are getting added, these two are getting added here together they moved it was getting added with third component, plus three things gets finally added with fourth component b 0 xn. So next thing comes out here and you see the same as, so this is y prime n. But if you remember the previous circuit it is same as original if I filter output will be yn. That is why they are same they are equivalent.

But now here asking the same question, if an extend data comes, if the data comes that is x of n how much time will it take to generate y prime n, that it puts input how much competition time this circuit requires to generate an output after which I can give the next input not before. Let us see one thing now, this is bit tricky so you have to be careful here. See same extend is available on this places. So this four multipliers are working parallelly on the same extend so they take the same time tm. That is common between these two circuits.

Now look at this adders, this adders is adding b 0 xn plus some signal here. This signal is coming after a delay so this was generated by this adder in the previous cycle. So, I do not have to calculate it in the current cycle. In the previous cycle this guy already calculated for B. So in the current cycle I need only to add these two. So, I will take a time t a and that much time will be required to generate these. So, what is this guy doing now in the current cycle, it is adding this component and this component.

But again this component one minute this component is not required to be generated by this adder in the same cycle because there is a delay. This component was generated by this in the previous cycle now it has come up. So, this addition will require only a t a but that result is not required here currently, that is it is doing something at the current n th cycle. That will be must on (())(08:12) next n plus 1 as cycle. So when it comes the task y prime n plus 1 computation that time only the current output of this will come here and will be required, not in the current n th cycle.

In the current n th cycle I require output of this for the as generated in the previous cycle, that is the n minus 1 th cycle, n minus 1 nt cycle it took some data, some data added held the result in the delay at n th cycle this came up. I just cool and add them, I get the result. I take t a amount of time. That time that in the n th cycle this is doing some job adding this component with this component but not passing it here just to read. That is keeping it ready for future use here, future means in the n plus 1 th cycle.

And while doing so it requires this data but this data is not required is not generated by this guy in the current cycle, it was generated by this guy in the previous cycle because we are delay. So, I do not have to depend on this adder in the same cycle. That fast complete this job then give me your output, then I complete this job, this I get this output here unlike the previous case where I had to do this addition first, for that I have to first take this job and for this I have to first do this job and for this I have to do this job or converse I mean reverse first you do this then this output goes here then only this can take place then the output generated goes here then only this can be done. Then the output goes here, then only this can be done that is not the case here.

The output here as generated is not required here in the same cycle because of the delay, it will be required in the next cycle. So, in the current cycle whatever is required this for was you already generated previously at m minus one this held in the flip flop.

Similarly this wants to do the addition between these two, when add these two addition. You did this but I do not need to do this first and in the same cycle pass it here because of the delay this fellow whatever he did in the (())(10:11) in the previous cycle that only is coming here now. So, I do not have to complete this addition and this addition together in the same cycle.

So, in the current cycle only these two will be added t a, similarly here this fellow is doing some operation down at the n th cycle adding this with this, and generating this which will be used of you in the next cycle not in the current cycle. In the current cycle whatever this guy needs is already generated in the previous cycle so no need to do that again in the current cycle.

So in the current cycle it is doing some other job what job, adding the current value of this and current value of this. And holding it in the flip flop here in the delay here for use here in the n plus 1 th cycle. At the n plus one th cycle that will take this as this at this, hold it and give it here in the n plus 2 th cycle and so and so forth.

Similarly here this guy it takes this data and this data adds takes t a amount of time, but this is not required to be produced by this guy in the same cycle because of a delay. Whatever it did in the previous cycle that will come of here now, so I do not have to waste some part of the current cycle to do this fast and then take the output and do this job.

So that is how it goes, which means I require only tm plus ta, or as tm plus four ta. And now even if I have to go of the four stages I have got 400 coefficients. Here it would be ta plus 400 ta, but here it is only ta plus ta.

So, this will take much less time and therefore (())(11:41) faster, this called as transposed form of FIR filter which you have studied in your DSP, or you studied DSP in filter architect class that direct from 1 direct from 2 for IR filters. Then for FIR filters you have studied transposed form, so this is a transposed form, and for to this, this books do not mention the you know advantage of transposed form which is basically that it is the faster one. Actually transposed form is what is called pipelined.

What is pipelined processing? Suppose there is a hollow pipe and you are throwing stones into it, so you throw one stone, you wait for the stone to come out of the pipe, then you throw another stone the stone comes out the pipe, then you throw another stone it comes of the pipe, if you go that way you are rid or speed of throwing stones into the pipe will not be high. But if you are smarter you will send a stone let it come up to some stage, may be here then you send another one and let this two move here this move here you send another one.

So, that way your stone throwing capability rate will go up because you are not waiting for the particular stone to come for the enter link of the pipe come out and then only throw the next stone. You need time to only cover some part of the pipe and then move in the next one then let the two move together further then throw in the next stone and so and so forth. So speed goes up, same thing implies in pipeline processing.

In pipeline processing just a minute, suppose you have got a job, total process total competition task job whatever you say. Input is coming here and this your output, now this we take time T. So if you take one data then wait for the entire job to be our taking time T. Then give another data. Let it be processed by this entire job taking time T and so and so forth. Then your input clock that is space between two input data will be T, not less than T, it could have been less than T because we are waiting to the entire job to be done. They only you are giving the next data.

So, it cannot be made shorter than T or in terms of speed it is a reciprocal of time period so 1 by T 1 by capital T, it could be made faster it cannot that is 1 by T is the clock input clock it could not have been faster. But suppose I divide the job like this, one job, another job, another job, may be the job takes time t 1, this job takes time t 2, this takes t 3 all at. But let one job 1 data come or I say that clock period which is maximum largest of t1 t2 t3 whatever. So, if the fast data come that will be processed by this guy then I hold it in flip flop delay till the clock current clock is over, when current clock will be maximum all this time.

Then this flip flop there is latch, latch or prints, and these output moves here, I give another data and that new data is processed here and that time the previous output which came here that is processed here. Then again after one clock this latch will (())(15:35), there is latch here this opens so this moves to this side, this moves to this side. Another new data comes that is processed here, this is processed here this is processed, then again another latch this opens this goes to this side, this goes to this side another data comes.

So, this way your speed will be much faster if tau is the max of values you know t1 t2 dot-dot. Then tau is the clock period, so in this case 1 by tau will be your speed, since tau is much smaller than T, your speed will be higher. So this is the pipeline processing. Of course there is one thing, there is a term called latency.

So if you give xn in the current clock cycle of period tau here, output is not coming out in the same clock cycle, 1 tau gone here then the intermediate output comes another tau gone here spent here, then the intermediate output comes here. Another tau so several times tau amount of time tau will be spent.

That is several clock cycles will be required to generate yn. So, there will be a delay but again this is delay is with respect to the faster clock, that is shorter clock period only tau. So, that delay is called latency, these two are very important terms. Now if you look at this transposed form at y sub is this, if you got some of the original one, look at these four adders, this part is this job, the four adders because the data from multipliers are coming parallelly, I am not bothered about it. The four adders together I mean one after another they are added here that is the job.

So, here you are giving 1 input and output was getting from so this we are getting added what's you given input all the four will get added and output will be generated but what is happening here is these multipliers now the same adders chain, these were the adders chain, now same adder chain I put in a box this is my process.

But here between them there is a delay so in one cycle the input whatever it is processed here. This goes to this side, the latch opens the flip flop, the D this opens this goes to this side then new data with this you know is added. That time another data comes here this is added here then again latch opens invest to this side, this goes to this side. New data comes here with them it get added. New data comes here with them it gets added it goes to add this side another data comes it gets added and likewise.

So this pipeline processing, that is why I say that transposed form is a pipeline that is its advantage. There is of course some demerit also otherwise you can always ask your question as to why not use transposed form all the time. Why use the direct form also, and the transposed form you get this line, I am saying splitter but actually they common line because same xn is given to all the fellows, we say it is a broadcast line.

The same xn is broadcast to all these nodes here all the adder nodes are getting multiplied. So, these are very large adder filter throughout your (())(19:22) indicates circuit one particular line has to be created to take your data xn throughout the circuit that is a problem because you have to have a path. So, through the ways of all the devices and all you have to have a path, you have to do routing for that path so that there is enough space. So that routing can create problem and then there will be propagation delay along this path.

I am assuming that the same extend is available together at the same time in all the places but that is not, once you give the xn if you are working at very high speed then that will take time. So, there will be propagation delay they are not available at all the points together because there will be delay there is a propagation delay. For large order circuit that have to be taken into account. Those are at the (())(20:08) side, but last side is it is a faster circuit because it is pipelined.

So, this much for the signal flow graph. As I told you SFD or signal flow graph is not something that we use so frequently. Rather it is a data flow graph DFG which comes next that is used most of the times and then there is another graph called dependence graph which we will consider afterwards. Dependence graph is very useful but only for a specific kind of architectures called array architectures. But we are doing very big competition but competition has lot of symmetries.

Same kind of job is done again-again when a big array forms. So for this big array based competitions the dependence graph is the key for architectures optimization. For SFG as I told you this is one high leveled transposition technique called transposition. For the other graphs also there are high level transposition techniques. Which we will consider in the next class, next so this is next class we will take up DFG, thank you very much