

C-Based VLSI Design
Dr. Chandan Karfa
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module - 12
Recent Advances in C-Based VLSI Design
Lecture - 42
Introduction to Circuit optimizations

Welcome everyone, in today's class we are going to discuss various digital circuit optimization techniques.

(Refer Slide Time: 00:59)

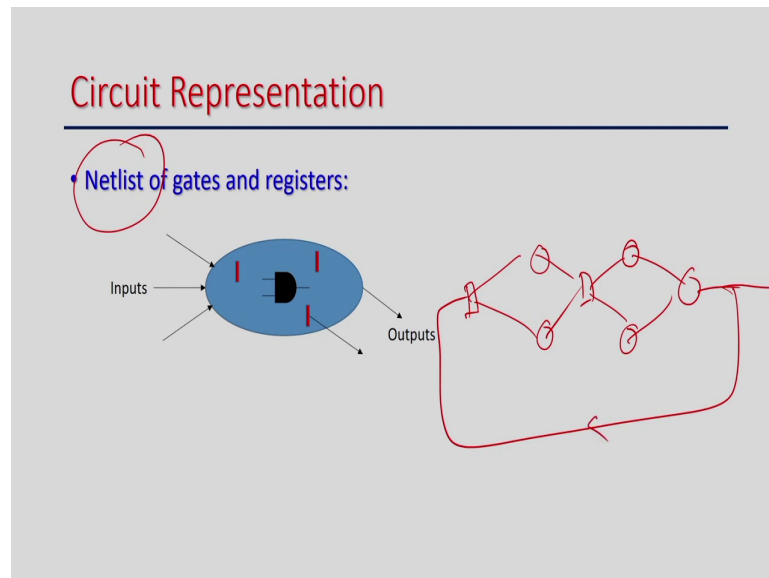


Specifically, in today's class I am going to talk briefly about the optimizations such as retiming, replication, folding, clock gating and glitch minimization. So, these are the optimizations that are not exhaustive, but they are very common in most of the commercial EDA tools. So, I will just talk about the optimizations and their impact in general in circuits.

So, if you look into this optimization in the EDA tools, specifically they try to optimize either area, power or timing. Timing in the sense, either latency or throughput or the clock period. So, there can be various optimizations. So, let us start with this retiming and before going to that let us see how we represent the circuit. So, circuits in general are

represented as a netlist . So, netlist is basically something you can think about as a graph.

(Refer Slide Time: 02:03)



It is basically a graph where there are a set of inputs there are a set of outputs and inside the netlist what you have? You have basically the gates, you have registers and some interconnection units. So, basically those gates and registers can be considered as the nodes and their data flow among them or the interconnection among them can be considered as edge. And that is how we usually represent the circuit.

So, usually this netlist is a very common terminology in the context of the circuit representations or optimizations we talked about. So, as I mentioned you can have this AND gates OR gates and so on you might have some flip flops; so, and their interconnections. So, we can represent it just like this. So, then this might come back here and so on. So, this is a feedback loop and so on. So, this is how we can represent the circuits.

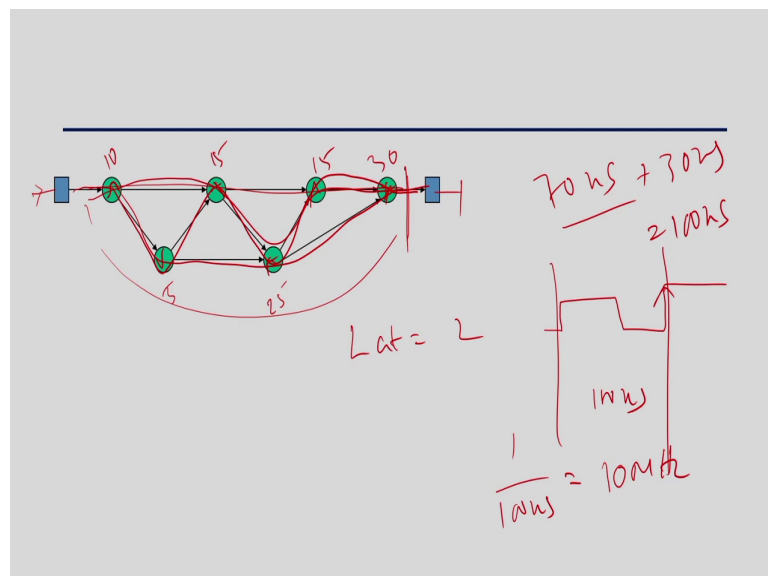
(Refer Slide Time: 02:58)

Retiming – Timing optimization

- **Problem**
 - Pure combinational optimization can be **myopic** since relations across register boundaries are disregarded.
- **Solution**
 - **Retiming:** Move register (s) so that
 - Clock cycle decreases, or number of registers decreases and
 - Input-output behavior is preserved
 - **Peripheral retiming:** Combine retiming with combinational optimization techniques
 - Move latches out of the way temporarily
 - Optimize larger blocks of combinational logic

So, let us talk about retiming. And, it is primarily used for timing optimizations or to try to improve the clock period. So, I want to run the circuit at faster clocks. So, let us try to understand that.

(Refer Slide Time: 03:17)



So, for that let us take this kind of as a simple graph. So, suppose this is the graph and these are the green nodes are basically combinational units, they can be AND gate they

can be OR gate they can be ADD, they can be anything. So, multiply and so on, XOR and so on. So, these are the gates.

And how is this clock period determined? The clock period is determined by the maximum combinational path length so; that means, it is the path between 2 sequential elements or the path from input to some sequential element or path from a sequential element to the outputs . So, basically it is a path consisting of only combinational units. So, there is no sequential element involved in a combinational path.

And how does this determine the clock period? Because say this is 10 nanoseconds say this is 15 nanosecond this is 15 nanosecond and this is a 30 nanoseconds. So, the total delay is basically 30 30 30 60, 70 nanoseconds. So, if this is the longest path. So, if you just assume. So, this is a 5, this is say 25. So, the longest path here you have to understand that this is one this is the longest path. So, it actually involves all the nodes.

So, what would be the total delay of this node? You can actually understand the delay of all the nodes basically here. So, which is basically plus 30 which is basically 100 nanosecond. So, you remember here that there are many paths like this is the longest path which determine the clock period, but there is another path like this, this is one path which is 30, 70 nanosecond this is another path, this is another path there are many paths actually here.

So, among all the combinational paths, the path that has maximum delay or the time, I mean time to execute that particular path, is determined by the clock period. Why? Because your clock should be long enough ; so, this clock period will be longer enough to execute that combinational path. You have to complete this execution of this whole combination within the clock period.

Because whenever the next positive edge clock comes by that time your data should be ready at this point otherwise the circuit will be unstable, it will not be in a stable state. So, if I take this particular circuit and since I found that the longest path combinational path is basically of 100 nanosecond.

So, my clock period should be at least 100 nanoseconds, which is basically 10 megahertz . Basically, it's a nanosecond, 1 by 100 nanoseconds so which is basically 100

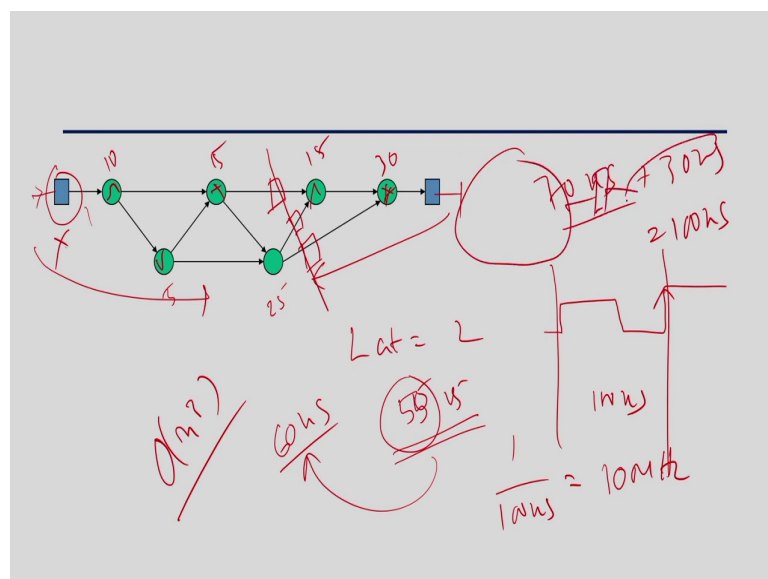
megahertz. This is the clock speed. So, this circuit will not run more than 10 megahertz clock.

So, you might do some kind of combinational optimizations here, but it's not possible to reduce the circuit optimized to under a certain level. So, after certain level combinational optimizations, whatever I talked about in previous 2 classes may not be applicable here or may not be able to optimize this combinational circuit further.

So, in that case the retiming comes into picture. So, what does retiming do? It tries to move the register or the sequential element of the design such a way that it does not change the functionality of the design it does not change the latency of the design, but it actually reduces the combinational path length.

So, I will just give you an example. So, in this particular path here what is the latency? The latency is basically 2 because there are 2 registers involved. So, whatever input you are going to give here after 2 clocks your output will be ready, but I can see that there is a longest path it is basically very long which is basically 100 nanosecond. So, what we can do here I will just clean the circuit a little bit and say, this is 15 this is 15 this is 30. So, I can see that here. So, this is basically 30 plus 45 50 70. So, this path is actually this very long.

(Refer Slide Time: 07:52)



So, if I can put the registers at this place. So, instead of this register, I just do not have this register to say if this register place is here. So, basically I then actually split the paths into two kinds of. So, this path length is 10 plus 5, 15 plus 25, 40 and this path is 15 plus 30, 45.

So, basically I can actually split the combinational path by half. So, I can see here that it will not go beyond 50 basically. So, it is kind of I can say this is basically 30 plus 40 or this is 40 55 basically. So, my clock period can now be 55 nanoseconds, which is almost double.

So, this is what and if I just move this register to this place, you can see for any path from input to output my latency will remain 2 although earlier I had only 2 registers I have now 4 registers. So, the number of registers actually does not determine the latency, rather the latency is determined by the number of registers in any path; so, the maximum number of registers that is the latency.

So, here I can clearly see that if I just remove this register and place it here, my latency will remain 2, but my clock period will almost double. So, this is what retiming does. So, retiming what it does? It is basically as an optimization method which basically undertakes the complete circuit and then it actually models it such a way that I am not going into how this retiming has been implemented. It is a very interesting approach.

But what it does is basically, but you give a certain target clock period, say I say my target clock period is 60 nanosecond. So, what does it do? It tries to model the problem in such a way that once this retiming is done it actually places this register here because if it places the target clock. So, it can achieve the clock period 55 nanoseconds which is the target clock.

So, retiming is an optimization which basically moves the register in a circuit such a way that a given target clock period can be achieved. So, you can understand that combinational optimization sometimes has a limitation because it's just between 2 registers. And you may not be able to do everything.

So, retiming is a very useful optimization and it is actually implementing all EDA tools which basically does this . So, there are three types of retiming, I would say the first one is the global retiming where you actually take the complete circuit and you model it and you give a target clock period.

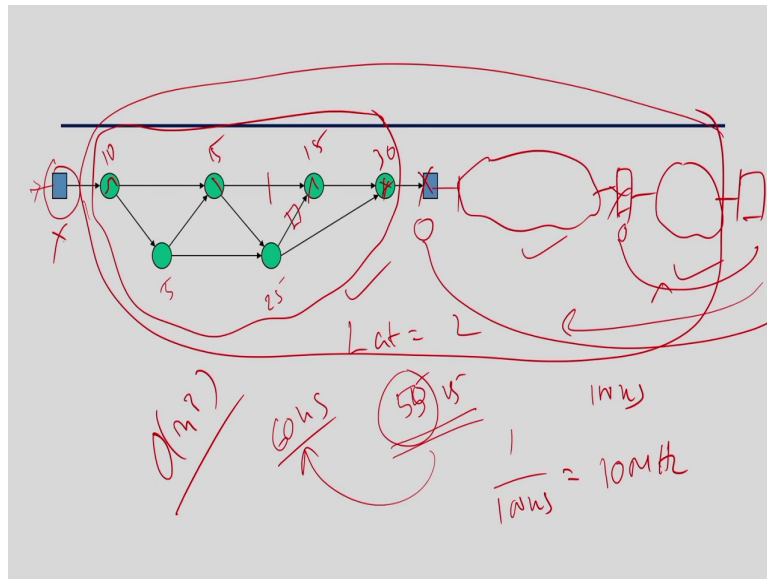
And then it will actually achieve that particular clock period and if it does not you will say I am not able to reach this. Say suppose for example, if you try to achieve here 40 nanosecond probably with two registers, it might be difficult, but actually it is possible because you can move this register backward this register forward and somehow you can manage it.

So, if if your your target lock period is not achievable, then it will say it is not possible to achieve that, but if it is possible let us you give you the retimed circuit or the modified circuit where the circuit functionality remains the same, latency remains the same, but it achieves the target clock period. This is the first one. The second one is called local retiming sometimes because this circuit has a million number of gates and retiming is usually a kind of order of the n cube algorithm.

So, that may be too complex to apply to a complete circuit. But I can always find out a small part of the circuit where it is actually causing a problem and then I can actually apply the retiming locally also. So, you can actually kind of divide and conquer kind of approach also you can apply that you can actually take step by step part of the circuit to and then apply retiming there.

So, that is the local retiming and the third one is called the peripheral retiming. So, the peripheral retiming is also interesting because sometimes you can understand that this combinational optimization cannot apply across registers. There may be some combinational circuit here in this part, then there may be some registers here and then there may be again a set of combinational circuits.

(Refer Slide Time: 12:12)



So, basically what am I going to write? So, there is some combinational circuit here then you have a set of registers, then again some combinational circuit and then set of registers. So, when you try to apply this combinational circuit, I mean optimizations, you have to take each combinational block separately.

You have to optimize this, you have to optimize this, you have to optimize this separately or individually because you cannot do this optimization across the register because the behavior functionality has a boundary this performance of this combinational circuit has a boundary this registers.

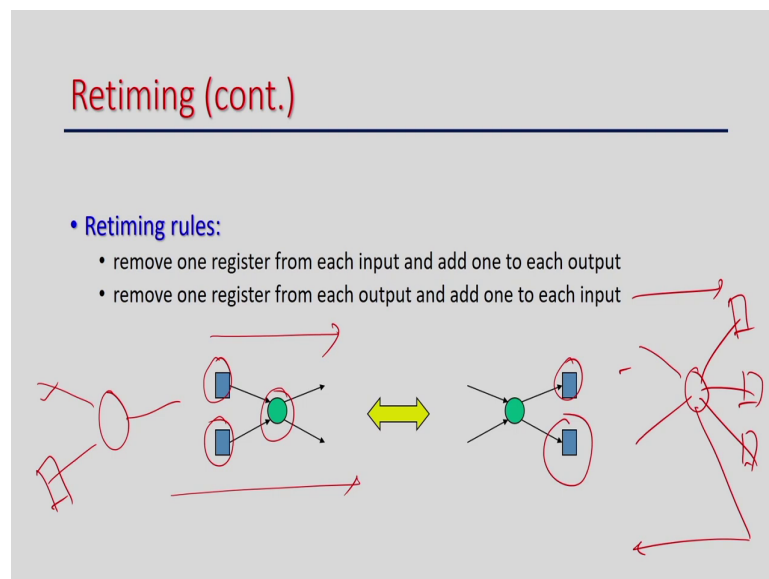
So, what the periphery retiming does is, basically moves all the registers at the periphery or the either to the input side or output side and then then the whole combinational circuit can be considered together and then I can do apply more optimizations and then once it is optimized, I can actually retime back this register to intermediate place so, that I can reach the target.

So, this periphery retiming does two things, it first allows applying this combinational optimization for a larger circuit because otherwise, it is actually a boundary of registers. I cannot apply the component circuit, but if I just apply peripheral retiming my registers will be moved to the outside.

So, basically what I am trying to say is that you move this register here, you move this register here and this is out there. So, now there is no register here, there is no register here, this is a complete circuit which is a combinational unit. So, I can apply the combinational unit in this particular part.

And I can apply more optimization because more opportunities will come because of the bigger circuit. And then you move back this register using the retiming algorithm in the proper place. So, that your target clock is met. So, these are the three kinds of techniques and these are all too done for timing minimization. So, now, I just talked about how these registers have to be moved, but what is the rule? So, how to make sure that we move the register so that the function will remain the same.

(Refer Slide Time: 14:14)



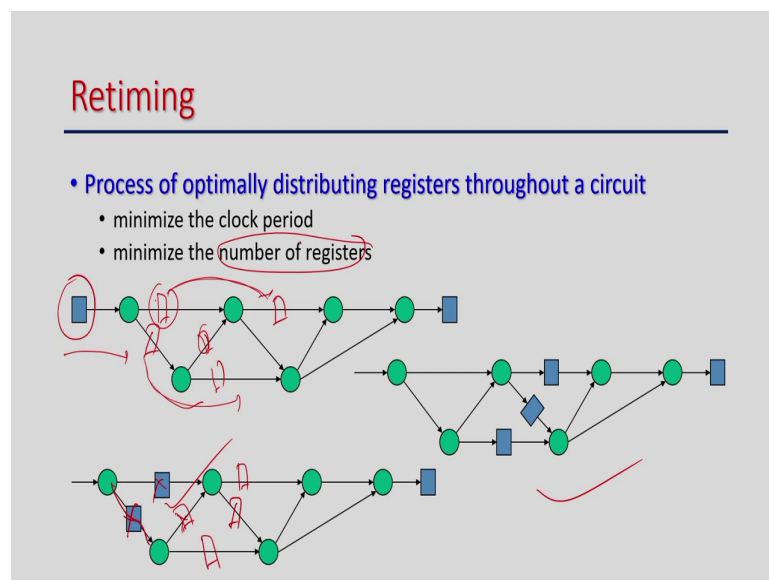
So, for that the basic idea is that whenever there is a node here if you try to move the registers, the register must be present in all input suppose you want to move this this way this direction so; that means, you can move the register only if it is available in all the inputs and once you move you should move them to all outputs. So, this is the retime circuit.

So, for example, if you have a gate like this and if you have a register here, you cannot move it here because there is no register at this part. So, this is the then you cannot this

you cannot apply retiming here. Similarly, if you have two registers at the input, you want to move into output what you are going to do? You have to put them in all outputs, not only one of the output, then only the functionality remains the same.

So, this is how I can move this to the outside. And similarly if I try to move them back, I can actually if it is in all output sides I can move them back into input, but if I see this register is not present here in this particular output then I cannot move them outside. So, that will change the functionality.

(Refer Slide Time: 15:25)



So, this is what this retiming and I have an example here which I already kind of explained. So, now I just try to see how I can apply these rules here. So, suppose I want to move this register as I mentioned, I have to move it with only one input. So, I can move it, but I have to meet moving into two output places. So, this is what is done here.

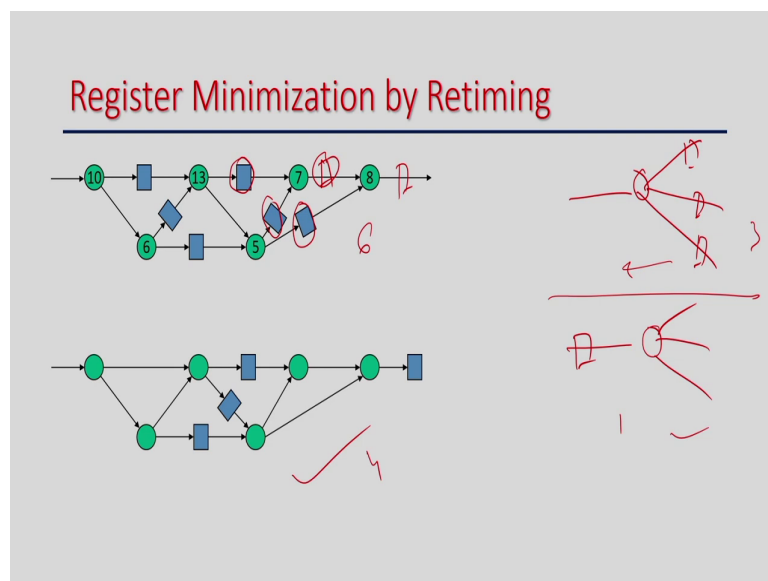
And then say suppose I want to move it further. So, if I try to move it further this one what will happen? So, it seems this is a single input to this node. So, I have to move it to both places. So, I can move it into both places. So, this will not be there now anymore. This is not there, this is a straight line connection.

And now for this node there are two inputs and both have a register, now I can move this both to the output port. So, if your objective; so, the example I have taken that says suppose I found that moving this register to this place is important, it is something needed. So, to do that first I will do this step.

So, it will come here. Now, I cannot move this register here because there is no register here. So, what do I have to do? Even if I have to move it here I have to move although this move may not be necessary because I would probably say this path is not a critical path.

So, this register is not may not be needed, it is not necessary, but because I need to move this register here first what am I going to do? I am going to move this register to these two places and then I can take these two registers and move to these places and this is the resultant circuit this is what this retiming does. So, the way it is done it actually follows the retiming rules. So, the retiming can also be used for reducing the number of registers as well, though if your objective is not to. So, if such a circuit is very small, it does not matter whether.

(Refer Slide Time: 17:20)

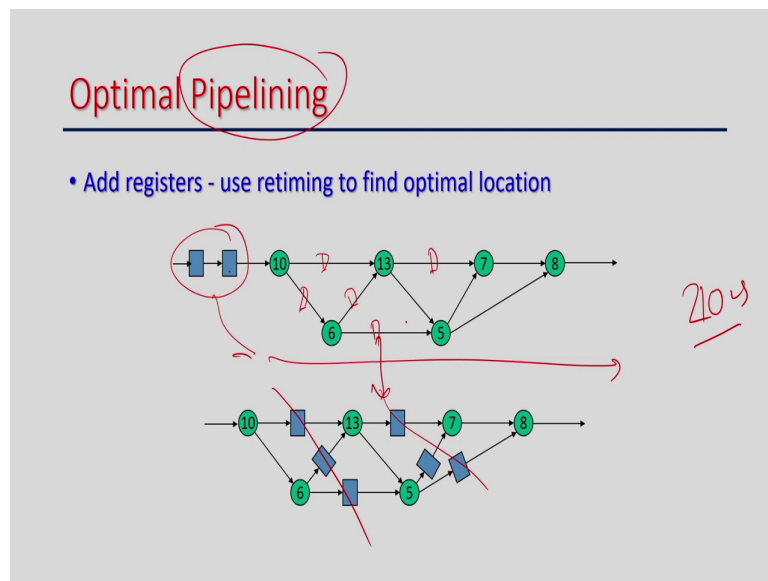


So, basically all combinational paths are actually meeting the target clock, but I want to reduce the number of registers. So, that is also possible because in that case what is basically happening? So, the number of registers I can move and I can actually reduce the register. And the basic rule is that, suppose there is a high fanout and if there are three registers here.

So, the latency of the circuit is one because from the input to output side all paths have one register whereas, if I just move it to the input side. So, this is a correct one because all output fanouts have a register. So, I can move this this way and still this is the resultant circuit has still the latency one, but this circuit is basically better because in terms of number of registers because here I need 3 registers here I need 1 register, but the functionality of the both the circuit same.

So, I can actually utilize retiming to reduce the number of registers as well. So, for example, I have given an example here, suppose I have this case. So, I can actually take these two, I can move them here. So, if one register is reduced then I can take these two and I can move it here. So, this is what is done here. So, here I have 6 registers and here I have 4 registers, but both circuits are actually the same. So, this again can be done using retiming.

(Refer Slide Time: 18:51)

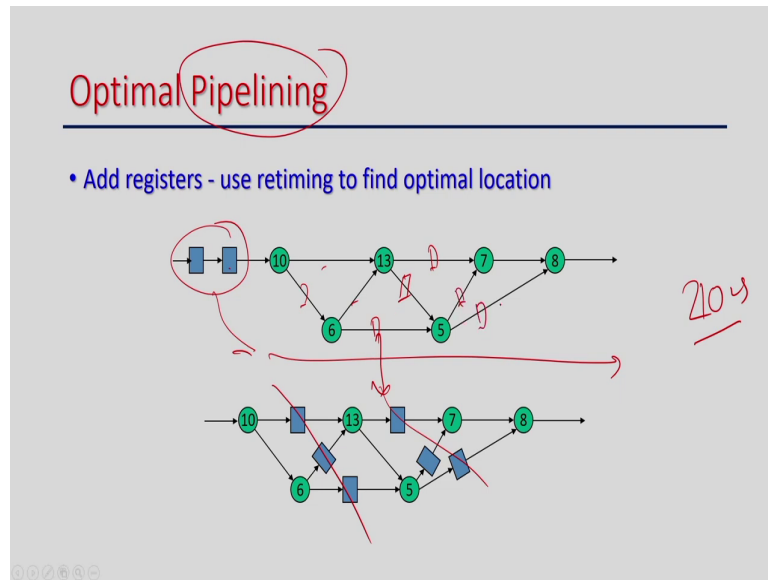


One more application is basically optimal pipelining. So, suppose I want to create a pipeline design. So, I know this is my circuit and then this circuit basically has to be a pipeline; that means, you have to add a register layer. So, you have to add pipeline stages and pipeline stages is nothing, but a kind of set of registers in that particular path and say these are the delays 10, 13, 16 these are the delays.

So, I can put the registers at the input and then I can set a target clock. So, I can set a target clock say, 20 nanosecond and then I can ask the retiming I can apply retiming. So, retiming automatically adds this pipeline in proper places. So, it will do it once one set of registered place here, it will take one set of registered place here.

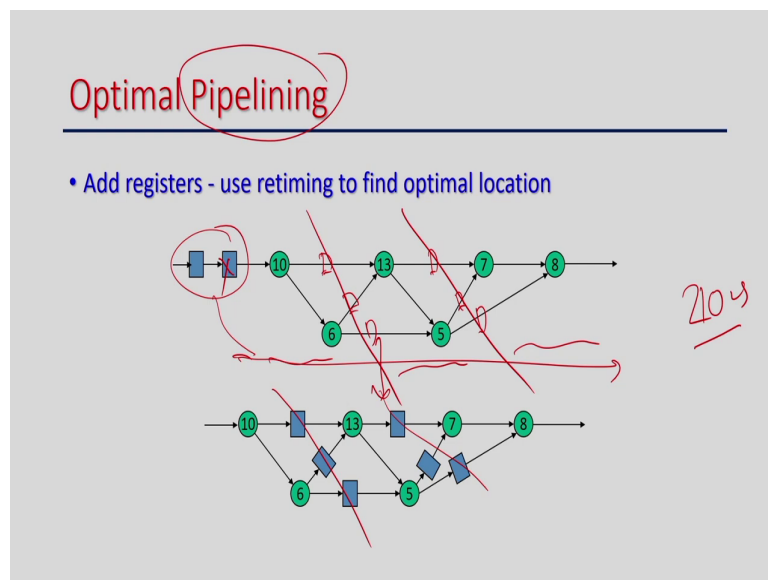
And it is basically you can understand that this is not a single step basically first it will take this register, it will place here the way I mention then it will take it will bring it here, then it will take these two it will bring it here then it will take this it cannot move it now, then what is going to do hm? So, these registers are not there. So, the two registers are like this now.

(Refer Slide Time: 19:59)



So, this register will also come here because once you try to move it here. So, now, these 2 registers are there. So, I can move it here . So, this is the one set of register movements. So, after moving one set of registers it will look like this and it is not there.

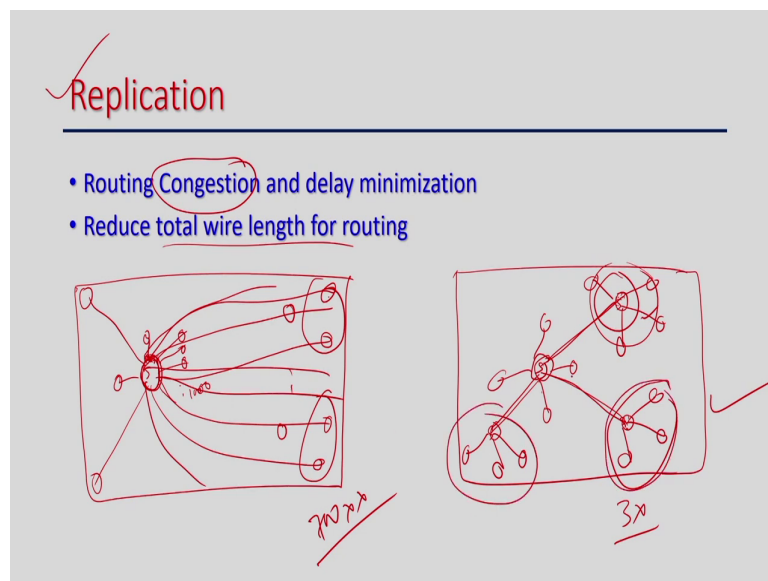
(Refer Slide Time: 20:11)



Then it will take this registers and it can move here and here then it can take this and it will move it here. This is how it can be done . So, this will result in this circle. So, this is one pipeline stage. This is one pipeline stage and the delay will be determined by the

maximum of these three delays. This is how this pipelining can also be implemented using registers. So, these are the many applications of retiming. So, that is why this retiming is so, important algorithm and it is actually available in all kinds of EDA tools. So, let us now move on.

(Refer Slide Time: 20:48)



So, another important optimization is called replication. So, it is a very simple, but very useful optimization. So, it is primarily used for routing congestion. I mean reducing the routing congestion and delay. So, combinational delay or routing delay rather and also it basically tries to reduce the total wire length. So, let me try to give an example of what replication is.

So, suppose there is a node here which is very high fanout see it has say, 1000 fanout. So, there are so, many 1000 fanouts so; that means, this signal is going to 1000 other nodes, now think about the scenario that. Now I am going to do the routing and I want to create the chip, think about this FPGA or say ASIC. So, now I am going to place this node somewhere.

So, suppose this is your area of the chip. So, since it is going to 1000 different locations, it has 1000 fanouts. So, my objective will be to put 1000 other nodes near to this place.

So, that the wire length will be reduced. But it is not possible because there are I mean I cannot put so many things nearby places. So, what can I do?

So, there may be some nodes that will be placed here some will be placed very far because you cannot place them all in together. So, other nodes will be very far away. So, these three may be nearby this may be nearby this may be nearby, but this may be here and so on you can understand that.

So, now, what is going to happen? You can understand here that it actually creates very long wire. So, then when you do the routing you have to route this particular wire in through the routing area. So, basically you can understand clearly that it actually increases the routing length. It will be more and also the congestion will be more, thus putting pressure on the routing algorithm because you have so many long paths you have to schedule them or route them.

And also the delay of the path will be more. So, if this particular wire length is more; obviously, the signal reach from this place will take more time than its reaching from here to here. So, these are the so, many things will come because of a high fanout node.

So, what I can do now as a solution is this . So, what am I going to do? I have the source node. I will just say place it here and what I can do? Instead of creating so many long wires I will replicate the source and I will create some buffer nodes. So, what am I going to do? I am going to create say another three source node buffer.

So, I just connect them. So, whoever is nearby it and then whoever is long I mean and then I try to place the other nodes near to the source. So, then the wire length will be less you can understand. So, I am going to place them here. I am going to place them here.

So, basically I replicate my source into multiple copies, earlier all such wires are long . So, all such wires are long now. Only a few long wires. So, these are the 3 few wires and because the other node these are the nodes will be placed too near to this source. These are the sources. I mean out nodes will be placed to this source and so on.

So, this is what it does. So, earlier say if there are say 700 long wires it is 700 x wire. Now, I only say three source nodes. So, it is basically three 3 x long nodes and then this

plus this. So, this will always be less than this so, and it. So, basically this is a very simple optimization the replications I am going to replicate a high fanout node and as a result it will help me to reduce the total wire length is very much clear that I do not have many long wires.

I have only very few few long wires and other other nodes which will be nearby to that source as a result total wire length will be reduced my delay will be less because since this path is coming here. So, after that this delay will be lesser than the longer wire length. And the third is basically since the number of long wires is less. So, routing congestion will also be minimized. So, that is the idea of replication. It is a very simple, but very powerful optimization to do so, with many advantages.

So, the question here is that I mean I understand that I have to. So, basically I will create this high fanout node with multiple copies. I will create a buffer for that. But how do I distribute this node? How do I know that? I have to place this node near to this which are the nodes I am going to put to this source, which are the nodes I am going to put under this source which are the nodes I am going to put under this source and so on.

(Refer Slide Time: 25:57)

Replication

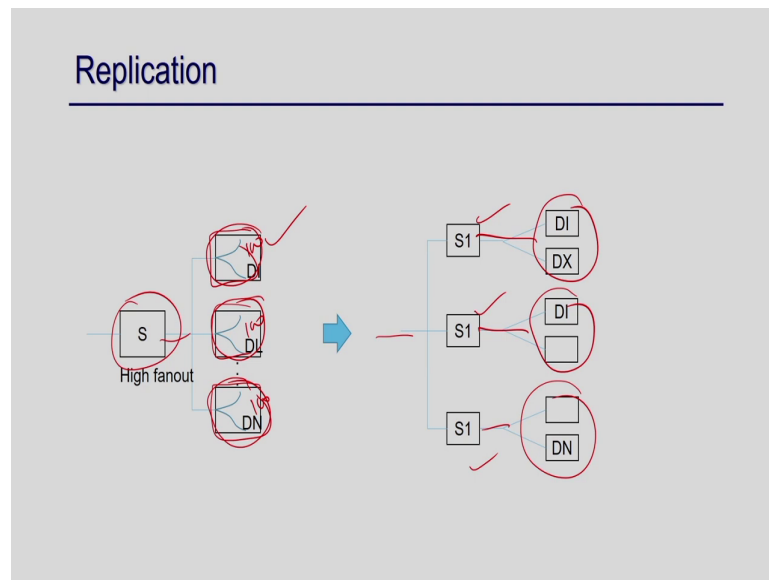
- Replication is used to redistribute the fan-outs of high fan-out source.
- For a high fan-out source, all of its fan-outs can not be placed in near by place.
- Routing delay would be high.

Solution:

- Replicate the source and redistribute the fanout logic.
- How to club the fan-out?
 - Better to keep track as the hierarchy.
- Replication of source may result in another high fan-out source.
 - May need to apply recursively.

So, this is something that is a question and basically we can actually keep track of the hierarchy and the nodes within a hierarchy I can place together I can break them together.

(Refer Slide Time: 26:09)

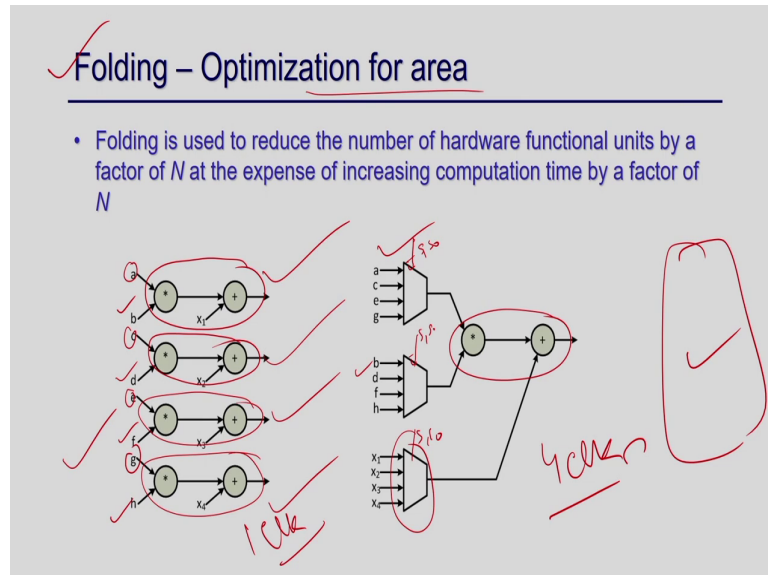


So, the idea is that if you have a this is a high fanout source node and there are. So, this is going to say there are 100 connections here, 100 connections here, 100 connections here, but I know this is another module this is another module and this is another module. So, when I am going to split them I am going to split them into module wise.

So, what I did I just this is my source node and now I created multiple source three copy of the source and I just put whatever it is here I am going to put it near to this whatever the fanouts are here, I am going to put to this source and whatever the fanouts are here I am going to put it here.

So, this will help to keep the nodes which are basically strongly connected. So, whenever they are inside the module they are kind of strongly connected and if I place them together. So, placing them during routing or placement and routing will be easier. So, that is the basic idea of replication. So, let us move on, the next one is another interesting and very important optimization which is called folding.

(Refer Slide Time: 27:09)



So, this is primarily used to optimize the area. So, what is this? Basically if you found in your data path or in your circuit that there are many repetitive kinds of instances for I just took an example here that I found there are 4 instances of 2 add-in series. So, it is basically kind of 4 copies of the same circuit which is basically two adders. So, what does this folding do? It is to fold this 4 into 1.

So, I am not in the modified circuit. I do not anymore have 4 copies of these two adders rather I have only one copy of the adder. And then what am I going to do? I am going to multiplex these inputs to these two adders. So, you can see here this guy is doing a plus b. So, I am going to say this is not a two adder multiply adder. So, this is basically the same thing. So, it does not matter what is inside. So, it is basically multiply and add.

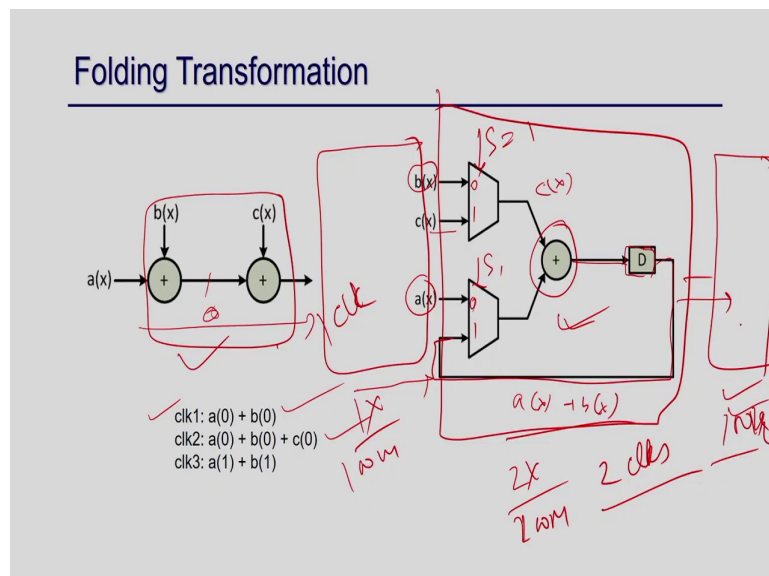
So, I can see here that I can use this. For this multiplication, what is happening? So, it is basically a I can put this a c e and g into this mux because this is the left input to the multiplier and the input is basically b d f and h. So, b d f and h I can put into another multiplexer and I can multiplex this input to this multiplier.

So, I can have a 2 bit signal now so, s 1 s 0 s 1 s 0. So, if it is basically 0 0 then say a into b is going to happen if it is 0 1 then c into d is going to happen if it is 1 0 then e into f is going to happen and if it is 1 1 then g into h is going to happen. And similarly after that this is multiplying with this x 1 x 2 x 3 force. So, I am going to put another MUX here. So, that this is now going to be when it is basically again this s 1 s 0.

So, effectively when this is 1 and s it is 0 0 is basically this circuit will be executed once it is 0 1 this circuit will be executed, when it is 1 0 this circuit will be executed and this is 1 1 then this circuit is going to execute. So, this is what is folding. So, basically what I am doing here is really time division multiplexing. I am going to use the circuit for one copy of the circuit and I am going to multiplex the input, but it is a time division.

So, because one copy of the circuit cannot perform four operations at the same clock. So, I have to distribute it in 4 clocks. So, earlier this circuit was working in 1 clock because they are all parallel. This will take 4 clocks because it is basically time division multiplexing that is happening here. So, I will come back to that and so, if this is for parallel copies.

(Refer Slide Time: 30:09)



If the operations are in series, I want to fold them that is also possible. So, what I am going to do here is that I want to fold these two adders into one. So, I can create that and then what I can do? I can put b here, a here. So, this is say s and this is also s.

So, in the first clock when it is 0, it is going to do $b \times$ plus $a \times$. So, this is going to happen in the next clock when s is equal to 1. I want to perform the result of this $a \times$ plus $b \times$ with $c \times$. So, what am I going to do? I am going to give $c \times$ here. So, the next clock 1 s is equal to 1 $c \times$ will come here and I need a plus b which is computing the previous clock because it is called calculating the previous clock.

So, I need to store it in some register. So, that is why I put a register here and then I put a feedback loop here. So, this is actually my $a \times$ plus $b \times$. So, when s is equal to 1, then it is actually calculated this $a \times$ plus $b \times$ plus $c \times$ this is what is going to happen; that means, in clock 1 a_0 plus b_0 is going to happen so x is something the clock. Next clock a_0 , b_0 plus c_0 is going to happen. In the third clock a_1 plus b_1 is going to happen; this is how I can actually alternate or time division multiplexer things.

So, again you can understand that this circuit is going to work. Only thing is that earlier it was taking 1 clock to execute this now it is taking 2 clocks earlier it is 1 clock, now it is 2 clocks. So, this is what is folding transformation. So, EDA tools usually identify such instances and try to fold them. But one thing is something the question might arise to you about how I am going to manage this 1 clock versus 2 clock. Because you can see it here.

So, this is say one module and this input is coming from another module. So, earlier it was giving in every clock and similarly this output is going to say some other module. So, these modules have to be synchronized because the initial specification if you think about this is the module implementation. So, if I implement this here.

So, this module is going to give you the input in every clock and in the next clock the corresponding output will go to this module. This is how those things, the

synchronization or hand shaking, happen when they know that this is that every clock I am going to get the next input. And this module also knows every clock will give me the correct output.

But now in the modified or the folded circuit I can see here that this is actually now taking 2 clocks. So, it actually changes the kind of the interfacing it also creates the problem to the interface because now this you have to do some kind of things here also. So, you have to make this circuit wait for 2 cycles to get the output and input in every 2 clock cycle.

Similarly, for this case the corresponding output circuit has to wait 4 cycles to get the output, so; that means something has to be changed here. So, which is something very complex we should avoid. What can I do? I can run this module now say if it is a 2 clock I can actually run in 2 x clock this is a normal clock 1 x clock.

So, suppose this is running in 100 megahertz this is earlier all modules were running in 100 megahertz now I am going to run it in 200 megahertz. I am assuming this circuit is replaced by this folded circuit and this is also running in 100 megahertz. What is going to happen?

Now, it is basically it it is giving the input in 100 megahertz speed and when the next clock come because this is a 200 megahertz, its 2 clock of this circuit will be done so; that means, it will actually can produce the output and for this circuit or this circuit because they are running in 100 megahertz and this is running in double speed they are going to get this output in every clock.

So, that means, whenever I fold a circuit or the module that I have to run the circuit in, that folding factors more speed so that the interface will be unchanged. So, they do not understand whether this circuit was running. I mean for them it is a still running kind of 100 megahertz.

Because earlier also they are sending the input in every clock in the modified circuit also they are giving they are they should be able to give the input or the output to this module

in every clock because this circuit is running faster. So, when 1 clock, one clock of this circuit means 2 clocks of this circuit. So, basically the interface will remain unchanged.

(Refer Slide Time: 35:05)

Folding Transformation

- Multiple Fus (Functional Units) are time-multiplexed to a single functional unit
- Is useful in DSP application

Basic Steps

- Identify common pattern such as MUL, ADD etc.
- Find the folding factor N
- Replace all the pattern by a single pattern with inputs on time-multiplexed
- Create a faster clock $N * \text{clk}$ to operate the folded clk

So, when you do the folding transformations what do you usually we do? We basically identify that common pattern and then your fold factor identifies the kind of folding factor. How many copies are there? You basically create the folding circuit and then run the folded circuit in N into N clock speed

So, the interface remains unaltered. So, that is the idea. And usually for DSP applications the folding is something very useful because in DSP there are many such repeated instances. So, this is about the folding transformations and it is primarily used to reduce the area of the circuit

(Refer Slide Time: 35:40)

Clock Gating – Dynamic Power consumption

- Dynamically disable the clock in specific regions that do not need to be active at particular stages in the data flow.
- Dynamic power consumption is directly related to the toggling of the system clock, temporarily stopping the clock in inactive regions of the design is the most straightforward method of minimizing this type of power consumption.

17

So, the next one I am going to talk about is clock gating, which is basically used for dynamic power reduction. So, the basic concept of clock gating is very simple in a circuit where you have two kinds of power. So, static power and dynamic power and dynamic power are at a very high level. It basically comes from the toggling of the bit.

So, when your signal goes from 1 to 0 or 0 to 1 the power consumption is more compared to if it remains from 1 to 1 or 0 to 0 in the next clock. So, when you toggle the signals in your circuit usually the power consumption increases. So, now, if you identify so, a circuit is a part of the circuit which you can understand that for say this circuit is running forever I can know that from say 11 cycle to 25 cycle this circuit is not doing anything useful. So, it is basically kind of the output its produce basically remains unused.

So, what I can do is, but since the clock is coming the input register is getting modified and every time you get a new data. So, there is a high chance that some of your signals get toggle and as a result to this combinational input you have a getting a different output. So, there will be a toggle and as a result this circuit is consuming more power.

What can I do? I can just get the clock . So, I can have a clock enable. So, this is my clock signal. What I can do here is I can actually have some controller which will generate this enable signal 0 for the example I have giving that 11 to 25 that particular cycle it will give 0 as a result your clock will remain 0.

As a result, what is going to happen? Your content of this register will not change because you do not get any positive clock . As a result, whatever the previous data it will contain, this register will contain the same register and the same data . So; that means, I am actually stopping the toggling.

So, earlier if it is whatever the value in the next clock also has the same value no toggle. So, the power consumption of this part of the circuit will be less and whenever I want to activate this I will just put the clock element equal to 1 so that this circuit will work as it is.

So, this is basically the very high level concept of clock gating is basically if you identify that a particular part of the circuit which is basically not happening, it's basically useful for a large amount of time of the complete execution time. So, I can just use the a clock enable, basically disable the clock for that part of the circuit for the time when it is basically ideal .And we should have a controller which basically generates this control enable signal. Again this is a very simple, but very powerful technique for reducing dynamic power in a circuit.

(Refer Slide Time: 38:27)

Glitch

- An electronics **glitch** or hazard is an undesired transition that occurs before the signal settles to its intended value.
- A significant portion of the overall power consumption of an RTL design is due to propagation of glitches in both control and data parts of the circuit.

18

So, the last one that I am going to talk about is a glitch. So, let us try to understand what is a glitch? Glitch is basically a hazard or undesired transition in a circuit. So, let me just give the example of how the glitch is generated. It is basically generated when for a gate two inputs are not coming at the same time. So, it's just a some minor timing difference between these two inputs and that then it will create some glitch at the output . So, let me give an example for this.

Suppose your x was coming here and suddenly it became 1 . So, then this 1 will come here and say since there is a gate here let say there is a NOT gate. So, now, earlier this was 0. So, that is why the NOT gate output is 1 and suddenly it becomes 1. So, it will be 0, but it will reach earlier than this one because it will take some amount of time to make it 1 to 0. So, this is the timing change.

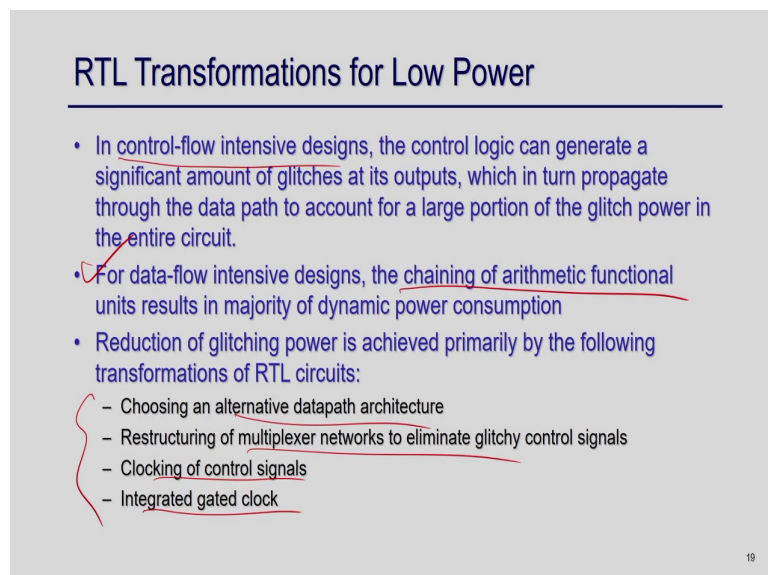
So, when it becomes 1 it should be 0, but at this moment this becomes 1 and at this moment it becomes 0. So, this is the period of time when this signal will remain 1. So, this is 1 and this is also 1 as a result output of the AND gate will be 1. Because 1 and 0 the output of the AND gate will be always 0 because I am giving x and x back to the AND gate, but because for a very fraction of time both inputs remain 1. So, I am going to get a glitch here.

So, my result will be like this whenever I change this value of x_2 to 0 I will always get a kind of sign of a glitch. So, this is something that is not changing the functionality of the circuit, but it is actually the glitch means some power whenever you have a glitch some power consumption going to happen this is it is kind of a toggling of the value.

So, this this is; obviously, if you have a NAND gate you will not feel this glitch, but if you think about a there is a AND gate which input is coming directly and there is a big combinational unit here which is giving the output and the second input the same input is going to this is a big combinational unit and then it is coming to this input. So, obviously, there will be some delay for this compared to this.

As a result, there will be some glitches. So, a glitch is something that happens because of the signal if both the signals do not come from a particular gate at the same time. So, that is something kind of an undesired behavior and this unnecessary consumption power.

(Refer Slide Time: 41:09)



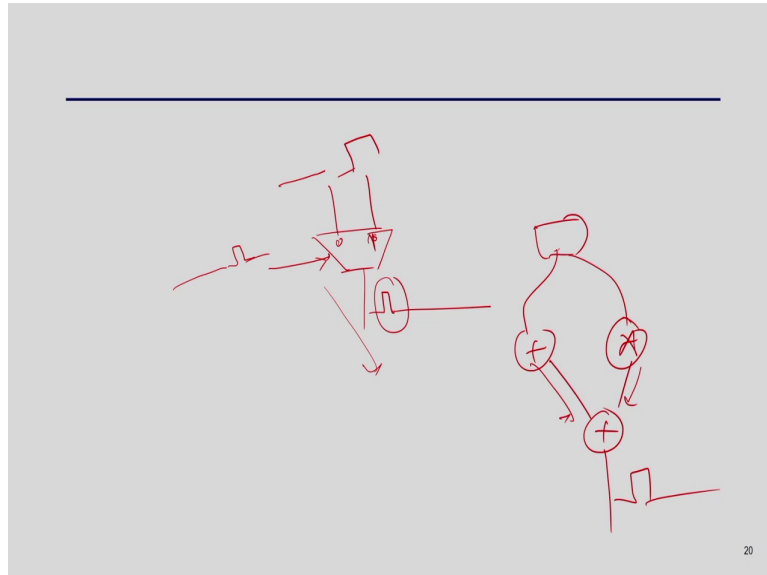
RTL Transformations for Low Power

- In control-flow intensive designs, the control logic can generate a significant amount of glitches at its outputs, which in turn propagate through the data path to account for a large portion of the glitch power in the entire circuit.
- For data-flow intensive designs, the chaining of arithmetic functional units results in majority of dynamic power consumption
- Reduction of glitching power is achieved primarily by the following transformations of RTL circuits:
 - Choosing an alternative datapath architecture
 - Restructuring of multiplexer networks to eliminate glitchy control signals
 - Clocking of control signals
 - Integrated gated clock

19

So, for that usually what we do is basically do similar kind of transformations the first so, for control intensive design if there is a input, say suppose there is a MUX here.

(Refer Slide Time: 41:20)



So, what is going to happen? So, there is a MUX here and so, say suppose this is your select signal and there is a glitch here . So, suppose this is 0 and 1. So, the actual signal is 0, but suddenly it becomes 1. So, suppose this is also 0 and this is 1 actually what is going to happen? Because it suddenly becomes 1 you will get a small one at that point and then it will be 0. So, again it will create a glitch.

So, the point here is that if there is a glitch in the control signal that actually gets propagated into the data path. Because because of the glitch the wrong input would be selected for a transit, very fraction of time and as a result that might create some glitch .So, for the control signal if there is a glitch that gets propagated to the data path and for the data path when you are chaining an arithmetic unit and then also it actually creates some kind of glitch.

Because you can think about that if there is a adder here and it is input coming from adder and a multiplier; obviously, this will come first this will come bit late because it is basically suppose they are actually input is same so; obviously, there will be some delay here as a result there will be glitch at the output of the this output of this adder.

So, because of both the cases you can actually have a glitch and there are similar kinds of optimization that can be done to make sure that this glitch actually can be reduced further. So, one is basically integrated clock gating to reduce the control signal control hazard or the control glitch you can actually clock the control signal so that you do not have glitches. It will be minimized because this control signal will only be 1 when the clock comes.

You can actually to handle this basically you can restructure the multiplexer network or you can actually choose alternate data path basically you restructure data paths so that you have uneven delay for both the inputs . So, these are the kind of techniques that are being applied to reduce the glitch. And again it is targeted to reduce the power consumption of the circuit.

So, with this I conclude today's class, again I just try to emphasize that I have taken very few optimizations. It is very important and it's very well-known and very popular in all EDA tools there may be many more optimizations. And there may be context specific optimizations which we we we should learn in detail, but in the scope of this particular course, I just talk about very few very important optimizations and their impact in digital circuit .

Thank you.