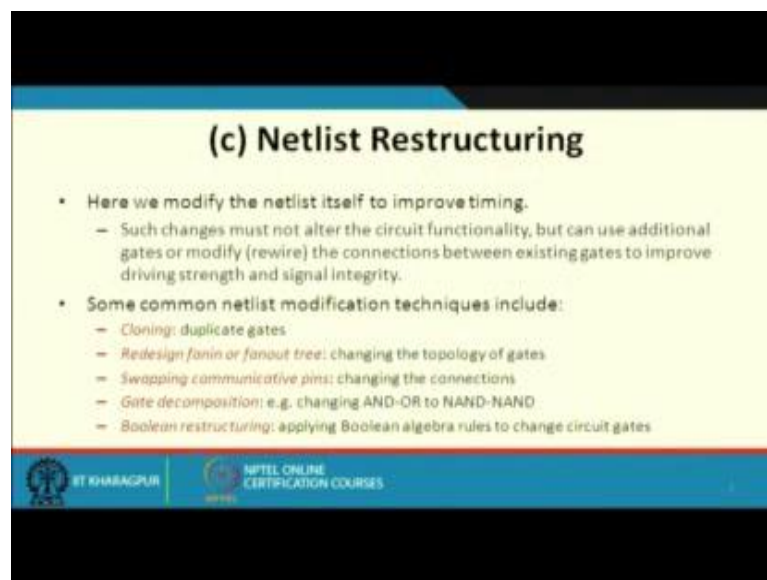


VLSI Physical Design
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institution of Technology, Kharagpur

Lecture - 40
Physical Synthesis (Part 2)

So, we continue with our discussion on physical synthesis. In our last lecture if we recall we talked about 2 techniques; gate sizing and buffering introducing buffering interconnection lines. So, we continue with our discussion.

(Refer Slide Time: 00:38)



(c) Netlist Restructuring

- Here we modify the netlist itself to improve timing.
 - Such changes must not alter the circuit functionality, but can use additional gates or modify (rewire) the connections between existing gates to improve driving strength and signal integrity.
- Some common netlist modification techniques include:
 - Cloning: duplicate gates
 - Redesign fanin or fanout tree: changing the topology of gates
 - Swapping commutative pins: changing the connections
 - Gate decomposition: e.g. changing AND-OR to NAND-NAND
 - Boolean restructuring: applying Boolean algebra rules to change circuit gates

IIIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we look at the third broad approach in this lecture netlist restructuring. So, as the name implies we are trying to modify circuit netlist in some way, and there are many ways in which you can do that. Now as I said that we are modifying the netlist in some way our objective is to improve the timing, but quite naturally we do not want that by this modification, this circuit functionality should change that should not be modified.

This circuit functionality should not be modified, but we can use some additional hardware like some additional gates may be required, some additional connections may need to be added. So, we shall see a number of various techniques here. This list is not exhaustive some of the very common netlist modification techniques as follows; cloning which as the name implies we do some kind of cloning or duplication here we duplicate gates, we can redesign fanin or fanout tree networks. We can swap commutative pins of

some of the gates, we can decompose some gates using some simple Boolean algebra rules, and here again using Boolean algebra rules we can restructure some circuits. So, we can implement them in different way so that the timing characteristics are improved; let us see one by one.

(Refer Slide Time: 02:27)

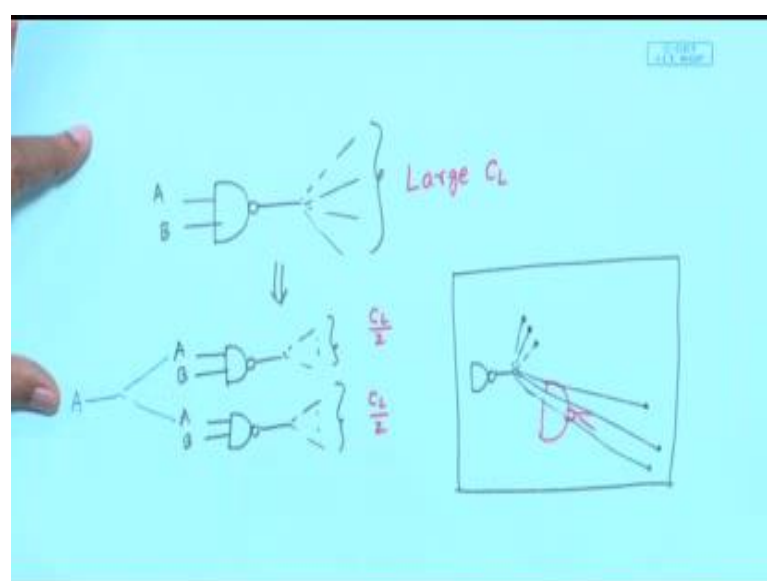
Cloning

- Gate duplication can reduce delay in two situations:
 - When a gate with significant fanout may be slow due to its fanout capacitance.
 - When a gate's output fans out in two different directions, making it impossible to find a good placement for this gate.
- The effect of cloning is to split the driven capacitance between two equivalent gates, at the cost of increasing the fanout of upstream gates.
- The second application of cloning allows the designers to replicate gates and place each clone closer to its downstream logic.

NPTEL ONLINE CERTIFICATION COURSES

Cloning; cloning as it implies that we sometimes may need to duplicate a gate. So, what we are actually talking about let me first intuitively tell you then I shall talk about that.

(Refer Slide Time: 02:42)



Let us say we have a gate, this gate output maybe going to more than one place. So, by cloning what we are doing let us say the inputs are A and B. So, we are saying that we are replicating this gate, same gate we are replicating twice. So, here it maybe going to many of the fanout points, but here we are sending this output to a subset, and this output to the other subset. So, maybe we are splitting outputs across these.

So, why we may need this cloning? There can be 2 broad reasons you can think of; first is that in this case there can be a large number of fanout connections, which means you may be having a large value of load capacitance, which means this gate will be driving those load capacitance charging and discharging will become slow, so the overall gate delay will be large. But here we have distributing this CL among 2 parts, if there are equal let us say I divided up into CL by 2 and CL by 2. So, my delay will become approximately half right.

So, here my consideration is only with respect to the charging and discharging times; if I can reduce the load capacitance my charging discharging time can also reduce. So, there can be another reason like say let us look at my total layout, let us say my this NAND gate was here and the output was going to so many different place. Let us say so some of the outputs were going here, these were 3 points where it is going; but the other outputs were going somewhere which are quite far apart let us say. Far apart means this each of this long lines will contribute to a long interconnection delay. So, intuitively this gates can be inserted either to handle this or to handle this long interconnection lines, maybe instead driving directly this long line you insert you replicate it and another copy to use which will be used to drive these and these gate you can locate in a suitable position not here may be you can place it here and here fine. So, let us come back.

So, just as I explained gate duplication or cloning can reduce delay for the following 2 situations; first when the gate fanout is large, so we can split the total fanout between these 2 gates, so the fanout capacitance will become less effectively. Suddenly if in the other case I just explained, that will be gates output goes to 2 different regions or 2 different directions, so that you cannot find out a good placement of these gate way to place it, because the outputs are going into 2 different corners of the chip. So, when even if you place in the middles still the length of the wires may longer. So, better to use 2 copies of the gates, one you place closer to that other you place closer to that, and you just clone like that.

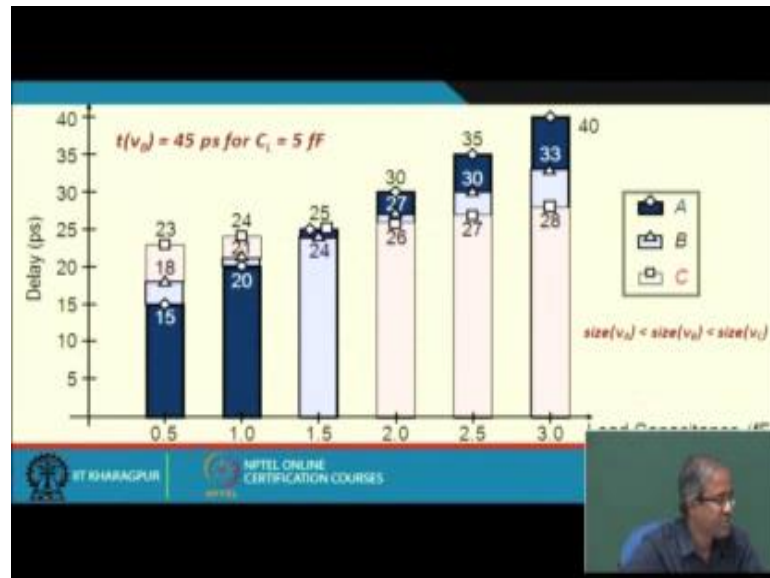
So, the effect of cloning just now we have seen is to split the driven capacitance between 2 equivalent gates; that means, output capacitance CL , was dividing was getting dividing by CL by 2 and this CL by 2, but at the cost of increasing the fanout of upstream gates what does this mean let us see here again. Initially this input A was coming to this single input of NAND gate, but now this input A must come to 2 different places. So, I am effectively increasing the fanout of the line A and also B. So, now, A is having a fanout, this is what we refer to as the upstream gates. So, these inputs which are coming after cloning they have to be driven both of this A and both of this B is to be driven. So, this is what we referred to as increasing the fanout of upstream gates. And the second one you can mentioned if the output going in 2 different directions. So, we can replicate gate, and place the clones closer to the downstream logic. Downstream logic means at the circuits which are towards the fanout side. So, we can place closer to that.

(Refer Slide Time: 08:19)

- Using the same load-delay relation as shown earlier, the initial gate delay $t(v_B) = 45$ ps.
- After cloning, $t(v_A) = 30$ ps and $t(v_B) = 33$ ps.

So, let us take an example we will use that same delay capacitance delay model that shown earlier. So, we take an example like this, where I have a NAND gate which is driving 5 fanout connections with a total load capacitance of 5. So, this is V the gate size B we have taken.

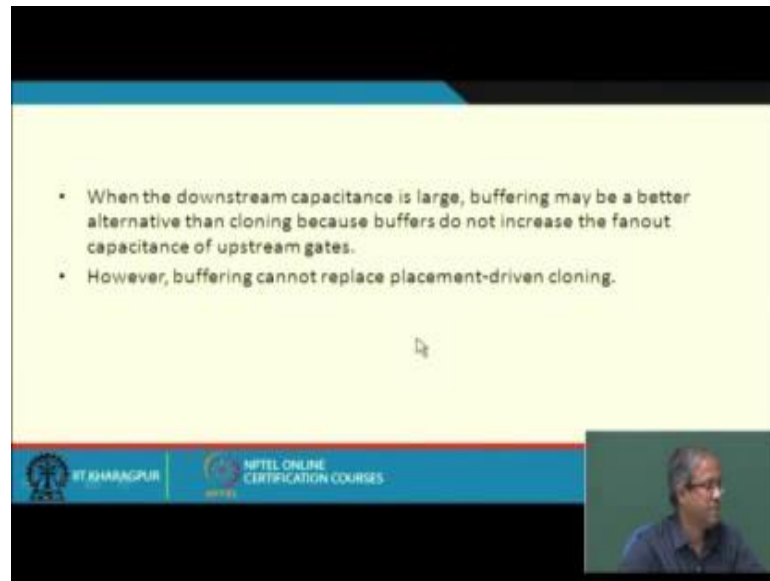
(Refer Slide Time: 08:54)



So, let us look at it once more for gate size V for 5 (Refer Time: 09:01) capacitance the delay was 45 prefer seconds. So, for this case delay is 45 prefer seconds. Now cloning we are doing like this, this we are splitting into. So, let us say one of them is V A and V B it is not necessary to that all these gates are previously same size. So, the 2 is so that we may have to be. So, here for example, here this V A is driving 2 loads V B is driving 3 loads, because here the load is higher we are using a larger gate here and because the load is less here we are using a smaller gate here. So, V A with capacitance 2, and V B with capacitance 3 let us see V A with load 2 it is 30 and V B with capacitance 3 is 33.

So, this will be 30 and this will 33. So, instead of 45 now delays is coming to 30 here and 33 here we have been able to reduce the delay right. But as I said the fanouts of a and b are increasing, now this a will be having one additional and fanout b would also having additional fanout.

(Refer Slide Time: 10:25)



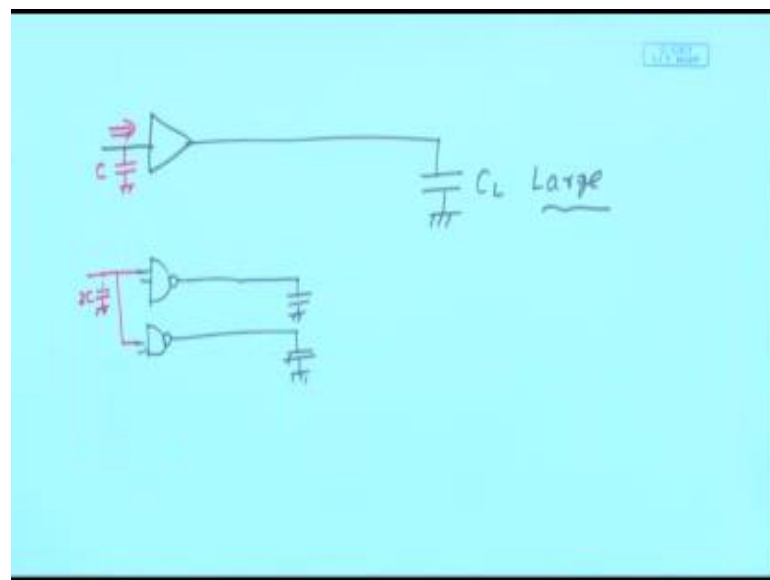
• When the downstream capacitance is large, buffering may be a better alternative than cloning because buffers do not increase the fanout capacitance of upstream gates.

• However, buffering cannot replace placement-driven cloning.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

When the downstream capacitance is large, buffering can be better than cloning. You see here the reason is there buffer do not increase the fanout capacitance of upstream gates, like here what I am saying is.

(Refer Slide Time: 10:52)



Let us say I have a scenario where I am driving some load, suppose this load is large I am driving a large capacitance; what I am saying is that here may be the buffering is a better option instead of cloning. So, cloning what we are saying that cloning by cloning

we are sharing the load, some part of the load we are sharing here some part of the load we are sharing here dividing output.

So, we are saying that the use of buffer can be better in cases where your driving some of the load capacitance is very large. Because the advantage of buffering is that say at the input side you are not increasing the fanout of the load capacitance, this capacitance becomes remains the minimum capacitance. But here if you are using like here suppose if this is C , then the effective capacitance here will becomes twice C because you are driving 2 gates now right? This is what I am just referring to here; because buffers do not increase the fanout capacitance of upstream gates.

But of course, placement driven cloning is very powerful concept as we said that, if there are some fanout targets which are located in somehow else of the chip, some other part of the chip then a clone of the driving gate can be placed closer to that; these will consists of so called placement driven cloning. We are doing cloning not just by looking at the loads or other things, and also we are looking where they are going. If you see there a cluster here you place a clone place a copy of clone near to that cluster, if there are some other points going there you place another copy of clone near to that cluster like that you do.

(Refer Slide Time: 13:03)

- The gate v drives five signals $d-h$, where signals $d-f$ are close, and g & h are located much farther away.
- To mitigate the large interconnect delay, the gate is cloned and a new copy v' is placed closer to g & h .

So, let us take an example here; let us say here there was a gate which was driving 5 signals $d f g h$; where let us say $d e f$ are closed together close to v , but g and n is little

further away. So, what we are saying is that you clone a gate and this new copy v dash you place it closer to g h, we do not keep it adjacent to v; keep it in little further maybe these wires will become longer, but this delay will become much less fine.

(Refer Slide Time: 13:46)

Redesign of Fanin Tree

- Logic design often provides a netlist with the minimum number of logic levels.
- Minimizing the maximum number of gates on a path between sequential elements tends to produce a balanced circuit with similar path delays from inputs to outputs.
 - However, input signals may arrive at varied times, so the level minimized circuit may not be timing-optimal.

ST KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the other techniques redesigning of the fanin tree; redesigning of the fan in tree means whenever there is a fanin connection.

(Refer Slide Time: 14:04)

- The arrival time $AAT(f) = 5$ in the original circuit.
- The modified unbalanced network has a shorter input-output path for the latest arriving signal.

Original circuit: $a \langle 4 \rangle$, $b \langle 3 \rangle$, $c \langle 1 \rangle$, $d \langle 0 \rangle$ inputs to a 2-level fanin tree with output $f \langle 6 \rangle$.

Modified circuit: $a \langle 4 \rangle$, $b \langle 3 \rangle$, $c \langle 1 \rangle$, $d \langle 0 \rangle$ inputs to a 3-gate unbalanced network with output $f \langle 5 \rangle$.

ST KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

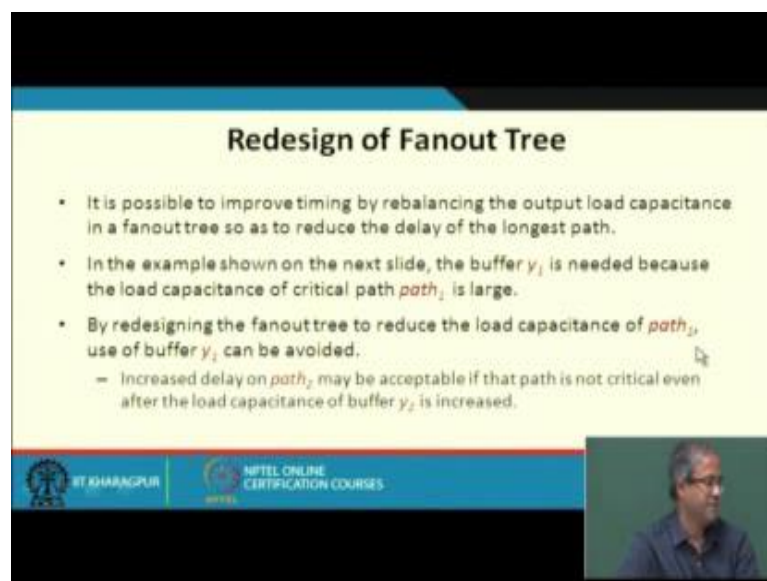
Just be take next take an example like this, suppose means we have a gate netlist like this here the type of this not important just there are 3 gates in 2 levels a b c d are the inputs,

and the actual arrival time of this of this inputs are shown 4 3 1 0, which means the first input is arriving late the last input is arriving earliest.

Now, if I implement this gate like this, 1 denotes the delay of the gates. If we implement this function like this, then the output will be available only after 6 minutes of time. Because one of the input a is coming only at 4; then 1 and 1 6. But suppose I do it like this now apparent different logic designer, logic design say that this a bad design, here I have 2 level that is means the delay was 2 there is 3 level. So, the delay should be three, but delay may be 3 in these static cases, but if you look at the arrival time this realization is better with this respect to timing.

Why? Because this slowest input a is encountering only one gate delay. So, now, the output will get ready by 5, because 1 and 1 this output will ready by 2 3 and 2, this output will ready by 4. So, this output will be ready by 5. So, here the output was coming at time 6, but after this restructuring your actual arrival time of output is becoming 5. So, this is the advantage. So, fanin restructuring is this whenever there are number of lines coming of the input of the gates, that using simply Boolean rules you can redesign the network such that this slowest signal lines you try to put in a place which will get the fastest route to the output like here you have done fine.

(Refer Slide Time: 16:31)



Redesign of Fanout Tree

- It is possible to improve timing by rebalancing the output load capacitance in a fanout tree so as to reduce the delay of the longest path.
- In the example shown on the next slide, the buffer y_1 is needed because the load capacitance of critical path $path_1$ is large.
- By redesigning the fanout tree to reduce the load capacitance of $path_1$, use of buffer y_1 can be avoided.
 - Increased delay on $path_2$ may be acceptable if that path is not critical even after the load capacitance of buffer y_2 is increased.

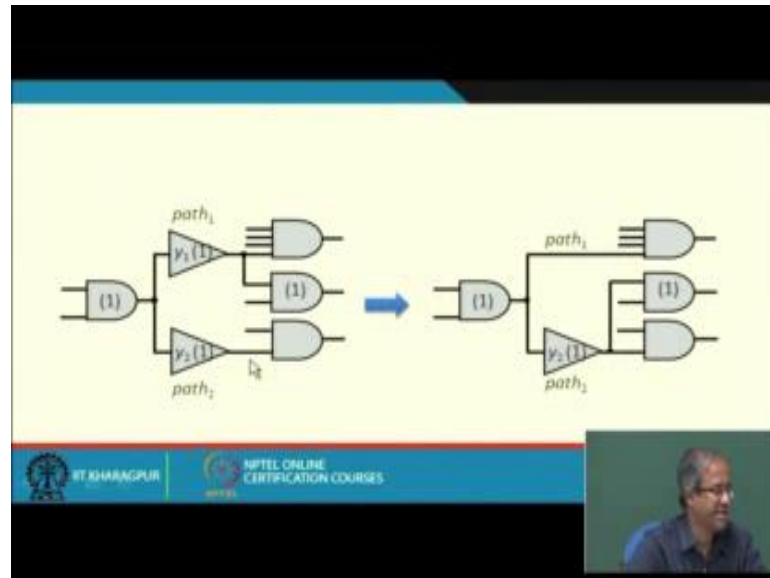
ST KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Similarly, we can do for the output side; like not only the input lines, but also the output load capacitance you can balance across the different sigma let us take an examples, first we shall comeback to the slide again.

(Refer Slide Time: 16:59)



Let us see the example first; let us suppose you have a circuit like this, where there was gate here the delay of 1, there were 2 buffers y_1 and y_2 . Let us say this path I call it as a path 1, this path I call it as part 2 this buffer is driving these 2 lines, and this buffer was driving this line let us say. But here what I do is that let us say we see these lines are ultimately coming from the same source same fanout, I broken up using buffer into this 1 and 2 and 3.

Now, suppose this one of this lines I can move it to here from here like this. So, if I do it then possibly I do not need this buffer any more here, I am not showing this may be going to other places also. So, one of the path, one of the buffers might get eliminated. Now maybe it was the case, that path 1 was the shorter, path we are trying to make it faster. So, this may be one method to do it. So, let us go back to the slide. So, in the example we just saw this y_1 was required because the load capacitance of the path 1 was large; load capacitance of this path 1 was large, you see it was driving 2 fellows to 2 inputs that is why we needed a buffer here. So, by redesigning the fanout tree what we did? We reduce the load capacitance of path 1 and so the buffer y_1 was avoided. So, this we shifted to here this buffer was already there.

So, it was already driving, so there is no problem, but this buffer you can avoid right? But this increased delay of path 2 we can accept suppose it was not only critical path. So, if you this kind of transformation actually the load on the second buffer will increase, so the delay on the path 2 will increase, but may be this path 2 as having sufficient positive flags. So, this was critical at all. So, even because of this additional increase in the delay, this is still remains with the positive flag. So, this can be acceptable fine.

(Refer Slide Time: 19:34)



Swapping Commutative Pins

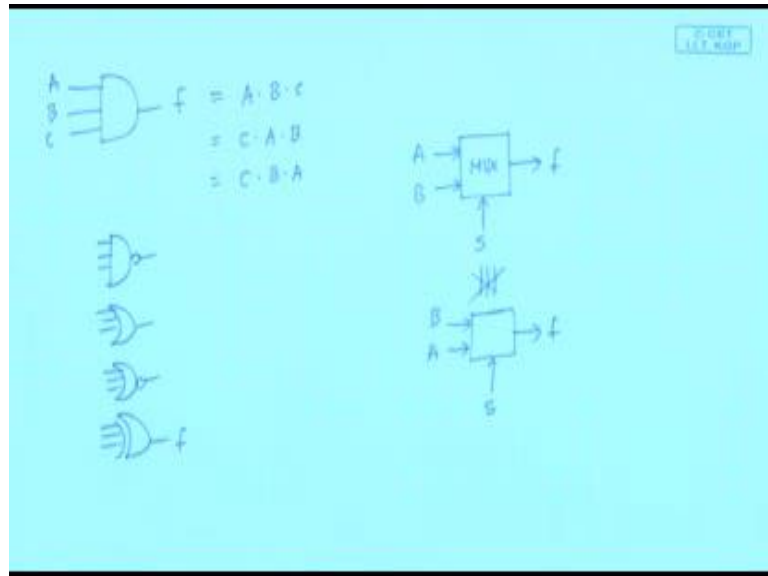
- Although the input pins of some gates (e.g. a NAND gate) are logically equivalent, in the actual transistor-level netlist they can have different delays to the output pin.
 - Path delays can change when the input pin assignment is changed.
- Rule-of-thumb:
 - Assign a later (sooner) arriving signal to an equivalent input pin with shorter (longer) input-output delay.

BT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Swapping commutative pin means there are many gates, where if you swap some of the pins they remain functionally equivalent, let us take some examples.

(Refer Slide Time: 19:49)

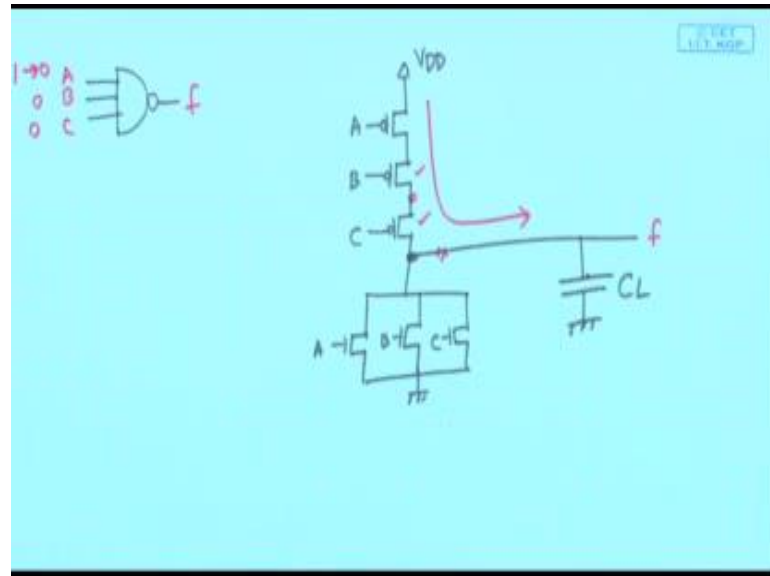


Let us take a 3 input and gate A B C which output is A B and C, now if you swap the gate with C A B it is same or C B A they are all same because this and function is commutative, A B is equal to B A. So, you can use there is a the rule of you can use rule commutative to prove this easily. Similarly the basic gates are all commutative in nature NAND or nor even exclusive or they are all commutative in nature you can swap inputs in any way you want the output function not will change, they remain functionally the same. But suppose I have some other function blocks let us say multiplexer, I have a multiplexer I have A B and the select line S here and this is the output.

Now, suppose I say that I swap my inputs, I make this B I make this A, this function will not be equivalent to this, this is not equivalent. Because if S equal to 0 in this case A was selected, but now here B will get be selected. So, there are certain functional blocks for which you cannot do this swapping of the pins, and the pins which can be swapped they are called commutative pins. For all the basic gates the pins are all commutative and you can easily swap all of them right. So, this method says that you can swap commutative pins.

Now, another thing you observed at the level of the gates this looks very fine that even if you swap this no change.

(Refer Slide Time: 22:03)



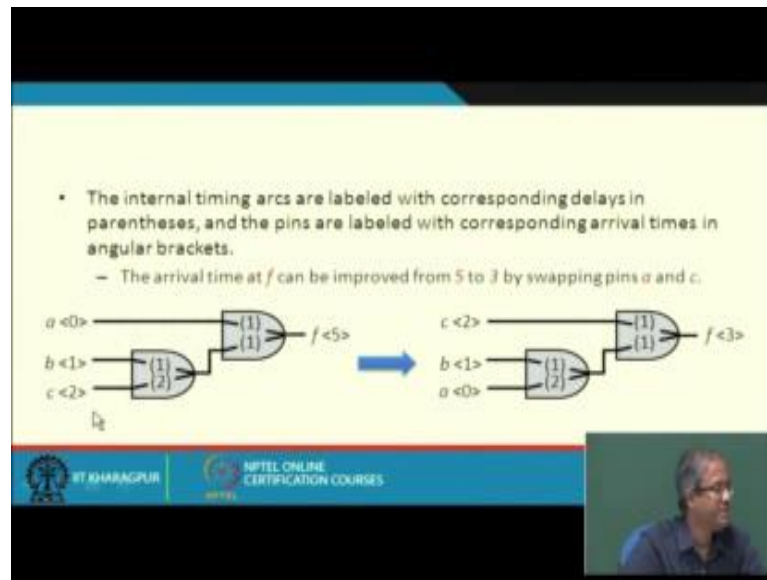
But if you look at a transistor level circuit, let us say I am just showing you one simple example a 3 input NAND. So, a transistor level circuit will consist of the pull up there will be 3 beta transistors, let us say A B C. In the pull down there may be 3 parallel transistors n type, now this output maybe driving some load capacitance CL, this is V DD. Now what I am saying is that although we are saying that this gate is commutative with respect to the inputs A B and C let us say the output is f is f, but you see when this A is switching, right.

Suppose B and C are already 0 and 0, A was 0, A is becoming A was 1 it is becoming 0. So, B 0 and 0 means this was conducting, this was conducting, A was 1 means this is of now this is conducting, now when A is conducting. So, this will be the path.

Now, the other case is that when I talk about C, this A B were already conducting; now I have to consider delay between this point and this point only. So, the net delay or this signal propagation delay did not be all from VDD to here, for some cases I may need to take the whole path some cases shorter paths. So, the delay across the pins can also be little different for a gate like this what we are trying to say. So, here what we are saying is that in actual transistor level netlist, the commutative pins which are so functionally can have different delays.

So, rule of the thumb is that if a signal coming later, you assign it to an input pin which is faster, at first signal is coming earlier we can assign it to a pin which is slower just rule of thumb.

(Refer Slide Time: 24:28)



So, this is simple example is shown here, these indicate the delays of the inputs; like these are 2 input gates both delays are 1 and 1, here it is 1 and 2 and these are the arrival times. So, for this realization the output arrival time will be 2 plus 2 plus 1 5, but if I make a commutation in the input it bring c here, b and a here then 0 1 0 is earlier. So, by 2 this will ready 2 by 3 this will be ready. So, from 5 we have reduced the delay to 3.

(Refer Slide Time: 25:22)

Gate Decomposition

- In CMOS, a gate with multiple inputs usually has larger size and capacitance, as well as a more complex transistor-level network.
- Decomposition of multiple-input gates into smaller, more efficient gates can decrease delay and capacitance.
- An example is shown on the next slide, where two equivalent NAND-level networks are shown.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is also one kind of a technique in which you can adjust the inputs by committing them so as to improve the delay. And last one is gate decomposition means there are some rules of boolean algebra. So, using which you can decompose gates into smaller gates, well the reason is (Refer Time: 25:44) one is of course, if you divide into smaller gates may be the overall delay become less, and also for a larger gates your MOS level transistor level network can be very complex, there will be more capacity effects involved delay will be larger, but if you can break it up into smaller gates that can be advantageous in some cases.

(Refer Slide Time: 26:14)

The diagram illustrates the decomposition of a 3-input NAND gate into two 2-input NAND gates, and then further decomposition into three 2-input NAND gates. Dashed arrows indicate the flow of the decomposition process.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this slide was showing a simple example, suppose there was 2 level and or realization. Now we know that any 2 level and or equivalent to 2 level NAND, NAND. So, this is equivalent to this. So, I can have a realization like this, here we need three 2 input NAND gates and one 3 input NAND gates, but this 3 input NAND gates you can possibly break it up into a 2 input and followed by 2 input NAND. So, here I have we have Alton realization where all the gates of 2 inputs only. So, this kind of a transmission may also have an impact on the delay, because in this case the delay of a 2 input gate and a delay of 3 input gate will not be the same they will be different. But here we are trying to meet the uniform.

(Refer Slide Time: 27:19)

Boolean Restructuring

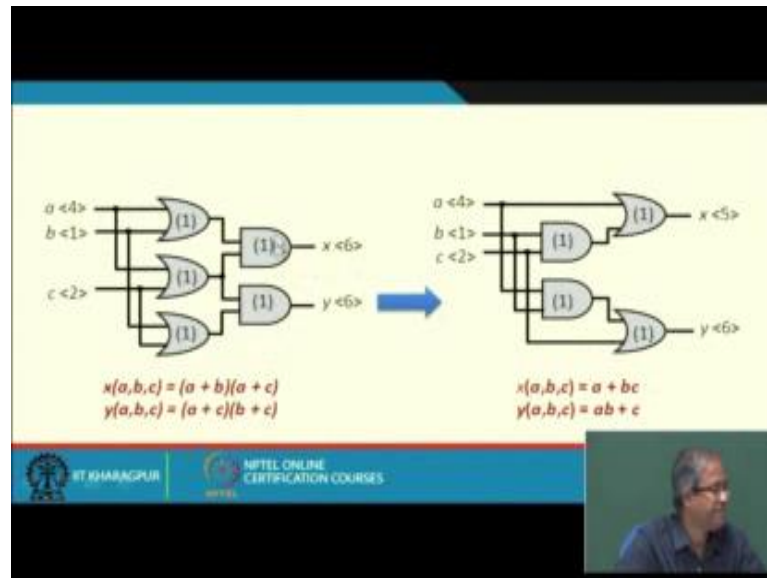
- Rules of Boolean algebra can be used to implement a function in various ways.
- An example is shown on the next slide.
 - The distributive law $f(a,b,c) = (a + b)(a + c) = a + bc$ is exploited to improve timing.
 - The two functions $x = a + bc$ and $y = ab + c$ are realized with signal arrival times $AAT(a) = 4$, $AAT(b) = 1$, and $AAT(c) = 2$.
 - When implemented with a common node $a + c$, we get $AAT(x) = AAT(y) = 6$.
 - Implementing x and y separately achieves $AAT(x) = 5$ and $AAT(y) = 6$.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And a Boolean restructuring is a technique where you can use the rules of switching algebra have Boolean algebra, to find out some common sub expressions essentially; and there is example which is shown in the next slide, which uses the distributive law of Boolean algebra, which says that $a + bc$ is the same as $a + b$ and $a + c$ right.

So, this kind of a distributive law you can use to improve timing; and in the example we are taking 2 functions one is x is equal to $a + bc$, other is y is equal to $ab + c$ right we shall come back to this let us see this functions first.

(Refer Slide Time: 28:14)



So, this x is the function used to recall a or b c. Now using distributive law you can write it as a or b a or c, similarly y was a b or c. So, in distributive law you can write a or b, a or c. So, just exactly like this we have to implement like this a or b, this implements a or c, this implements b or c. So, by just ending the required to all outputs you get x and y.

Now, if we assume that the input arrival time of the inputs are 4 1 and 2; a is coming at 4, b at 1, c at 2, 4 1 2. So, the delay will be 4 plus 1 plus 1 the maximum for both the cases 6 and 6. But now suppose we are not using this way, we are using straight implementation like this a or b c, a b or c that is you are using and in the first level or in the second level. So, you see since a is coming out straight. So, here the delay will be 5, but why still 6? Because a is coming at 4 then you need to get delays. So, at least one of them were able to speed up, right.

So, with this we have seen a number of techniques using restructure netlist, so that some you can say characteristics which respective timing can be improved, because we need a lot of this kind of corrective actions to be taken after we carry out static timing analysis we find out that there are lot of timing requirements, which are not been met we may have to make a number of changes and transmissions the circuit netlist placement. So, we have talked about some of these techniques which have quite commonly used.

So, in our next lecture we shall be continuing with this we shall looking at the overall performance driven flow at what we have learned so far, that what it looks like in an integrated sense.

Thank you.