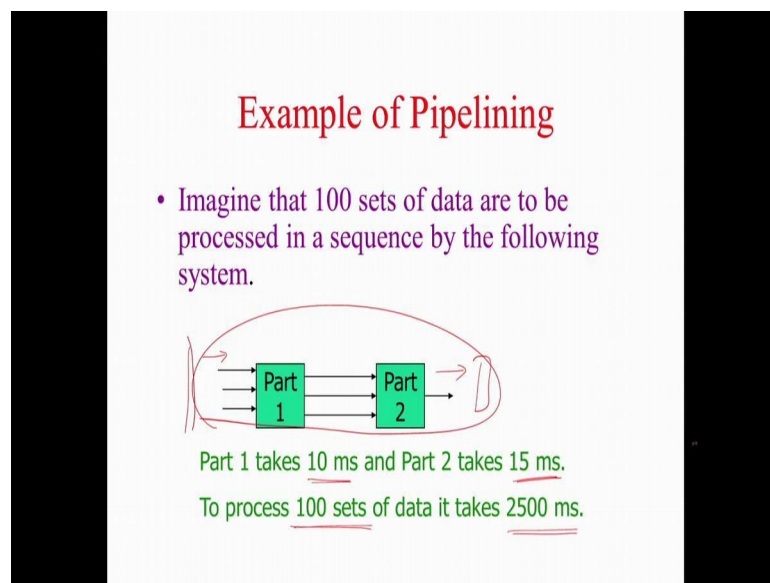


**Computer Organization and Architecture**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 17**  
**Part 2**  
**Pipelining**

(Refer Slide Time: 00:24)



Yeah. So, we will start with pipelining. So, let us look at this particular diagram there are parts to this circuit. So, the inputs come from this point it gets processed by part 1 and then it is fed to part 2 it gets processed and it comes out.

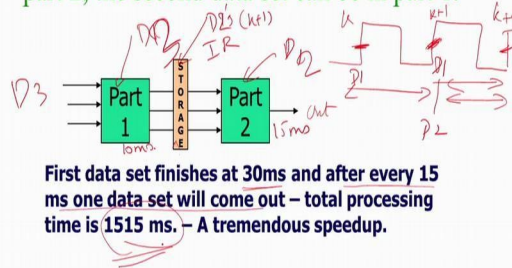
So, let us assume that part 1 takes 10 milliseconds of time and part 2 takes 15 milliseconds of time. So, when a data enters at this point for it to come out of part 2 completely processed it will take 25 milliseconds of time sorry, 15 millisecond 25 milliseconds of time and suppose I have 100 sets of data for me

to finish processing of this hundred sets of data would be 2500 milliseconds right. So, this is a circuit with part 1 10 millisecond part 2; 15 millisecond total time is 25 millisecond for processing one set of data and then suppose you are processing hundred sets of data it will take 2500 milliseconds, right.

(Refer Slide Time: 01:24)

## Example of Pipelining

- Consider the following changes, with a storage element. When first data set is in part 2, the second data set can be in part 1.



Now, suppose I put a storage element in between right. So, what happens here this storage element is nothing, but a register and it gets you know powered by a clock now what happens is I can get the first set of data and it will get processed. So, let us say there is a clock going like this. So, the first set of data comes in within 1 clock pulse that is this duration this data will get processed by part 1 and it will be available for storage in this part. So, when the clock pulse comes that is with this clock come the processed part of the first set of data essentially gets stored in this storage element please note that this is a flip flop.

So, what is a flip flop the content of it will change only at the positive edge of the clock or negative edge of the clock let us assume it is positive edge. So, what happens at this particular clock edge the first set of data enters it gets processed in part 1 at the end of this. When the next clock pulse come that processed result is stored here correct is it and what will happen from this clock to this clock as you see here in the second period this set of data gets processed by part 2 and in the next clock pulse it will be available for us as an output right.

So, in clock  $k$  the data enters in clock  $k + 1$  that intermediate results gets stored in flip flop in clock pulse  $k + 2$  the answer comes out fine now please note that between  $k$

and  $k + 1$  part 1 was working between  $k + 1$  and  $k + 2$  part 2 was working between  $k$  and  $k + 1$  part 2 was idle between  $k + 1$  and  $k + 2$  part 2 was part 1 was idle.

So, what I can do is I will pump the first data now in the next clock pulse I can pump the second data. So, what will happen in the first clock pulse the  $D_1$  enters  $D_1$  reaches here at this point I can say  $D_2$  also enters the system  $D_2$  enters here and during the second clock pulse that is between  $k + 1$  and  $k + 2$  what will happen part will be processing  $D_1$  while part 1 can be processing  $D_2$  right and all that you are processing will not affect the input to  $D_1$  because this is a storage element and any change here will get in to this storage only at the next rising edge. So, essentially this storage element acts as isolation between these 2 stages. So,  $D_1$  enters at clock  $a$ .

So,  $D_1$  enters at clock  $a$  at clock  $a + 1$   $D_1$  intermediate result is stored here this is at clock  $a + 1$  at that clock  $a + 1$   $D_2$  enters between  $k + 1$  and  $k + 2$   $D_1$  is processed here while  $D_2$  is processing in the clock  $k + 2$   $D_3$  enters  $D_2$ 's intermediate results are stored and so, between  $k + 2$  and  $k + 3$   $D_2$  is processed and  $D_3$  is processed. So, this is how the data can flow. So, what happens now we took 2500 milliseconds for processing hundred sets of data 25 milliseconds per data?

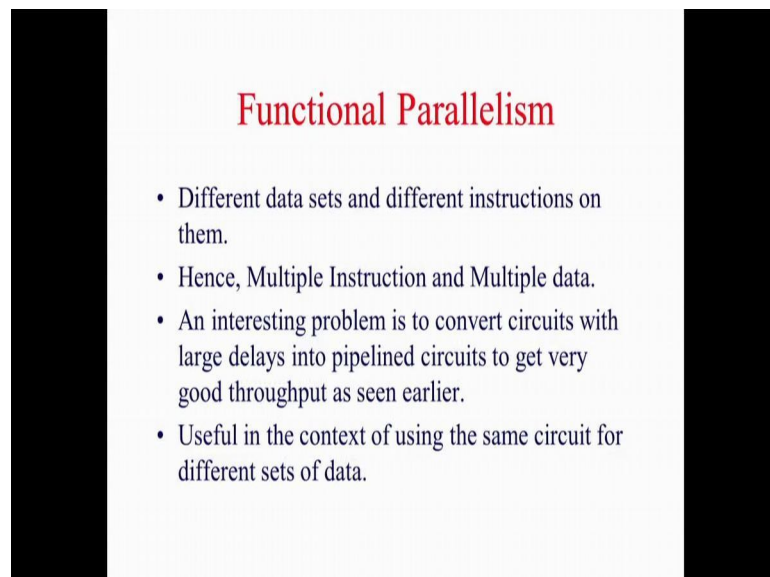
Now, what will happen this is 15 milliseconds this is 10 milliseconds right. So, the first data will come when please I have the same clock pulse. So, the; what should be the time period of the clock pulse should be at least 15 milliseconds right should be at least 15 milliseconds if it is less than 15 then this fellow cannot finish. So, it is, the first set of data will come only at 30 milliseconds and then after that every 15 millisecond you will get 1 data. So, totally I will finish the entire operation by 1515 milliseconds. So, 1000 data that 2500 now can finish in 1515 which is at most 40 percent increment right at least 40 percent we have improved is it right.

So, this is the notion of pipelining and what we do here in pipelining is that that we introduce storage between 2 parts of the circuit and that storage will be like isolation. So, that one part can work on different set of data without influencing the next part which is working on some other set of data and like this is can allow data to keep flowing are you



able to follow and one of the important thing that you should note here is that the frequency of the clock; the time period of the clock is dictated by the part of the circuit that has maximum (Refer Time: 07:26). So, how do you improve on this?

(Refer Slide Time: 07:31)



The slide is titled "Functional Parallelism" in red text. It contains a bulleted list of four points. The slide is flanked by two solid black vertical bars.

### Functional Parallelism

- Different data sets and different instructions on them.
- Hence, Multiple Instruction and Multiple data.
- An interesting problem is to convert circuits with large delays into pipelined circuits to get very good throughput as seen earlier.
- Useful in the context of using the same circuit for different sets of data.

Now, before going forward please note that this is an example of what we call as functional parallelism why this is functional why this is called functional parallelism because I have different data sets meaning the different instructions right and they are acted on by different part of the circuit right, so different parts of the; so, each; so, each circuit can represent instruction right that is working on the data. So, I have different instructions acting on different data right part 1 is working with part 1 is an instruction part 2 is another instruction when part 1 is working on D 1 part 2 will work on D 2 when part 1 is working on D 2 part 2 will work on D 3 or whatever vice versa.

Now, what happens because of this I have multiple instructions part 1 is an instruction by itself part 2 is an instruction by itself and I have multiple data? So, a pipeline is an example of a multiple instruction multiple data environment. So, any time you have a very large circuit in which you can do continuous operations

like multiplication I may not just have 1 2 numbers to be multiplied I will have several pairs of numbers to be multiplied in a program.



So, if we get such type of circuit where it could give continuous flow of data then the best thing is to pipeline and by pipelining already for a simple of the simple of the circuit we with a small pipelining introducing of a storage element we could get up to store up to 40 percent saving in your time the same thing is also possible. So, this is actually useful in the context of using the same circuit for different sets of data right. So, I had a one circuit which I pipelined and then I am using it for a example multiplier is an example of a circuit you will be using for multiplying thousands and millions of numbers right and if you have such type of things where I could have continuous flow of data then functional parallelism or whatever pipelining is extremely use full any doubts.

Can I proceed now let us I will I will just do this simple thing of how to convert a circuit a combinational circuit in to a pipeline circuit.

(Refer Slide Time: 09:57)

### Pipelining Concepts

- A combinational circuit can be easily modeled as a Directed Acyclic Graph (DAG).
- Every node of the DAG is a subcircuit of the given circuit – forms a stage of a pipeline.
- An edge of the DAG connects two nodes of the DAG.
- Perform a topological sorting of the DAG.

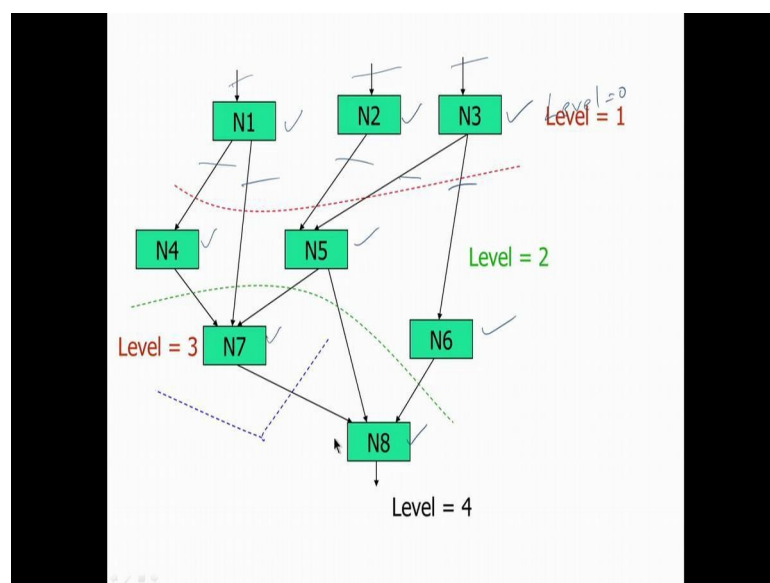
Any combinational circuit is basically a directed acyclic graph what do you mean by a directed acyclic graph it is a graph it is a directed graph directed graph in the sense that every edge has a direction right for example, for example, for example, this is a directed acyclic graph only on I will have something here. And then this is an example of a

directed acyclic where every edge has every edge has a direction it goes from  $i$  to  $j$  and there is no cycle I cannot start from any of this node and come back to that node for

example, I start with 1 I start with 1, I start with 1 and I can go to 2 no way I can come back to 1.

So, I can only go to 4 I cannot back to 5, I can go to 6 and finish are you able to follow this. So, no way can I come back to the same thing in a directed acyclic that is why we call it as an acyclic graph there is no cycle here that is why we call it as an acyclic. So, every node of this directed acyclic graph is a subset of the given circuit. So, I can have and gate or gate multiplier etcetera and I could have larger thing I could have an adder I could have all the standard cells here and that could form a stage of a pipeline. So, an edge of the dag connects 2 nodes of the dag. So, what we do we first do a topological sorting of the DAG.

(Refer Slide Time: 11:56)



So, let me just tell you what you mean by topological sorting this is the basic circuit which has eight combinational elements each of these elements can be some adder subtractor or it can be a big combinational circuit now this is the thing. And now what happens I do something called topological sorting what is topological sorting first I will remove all the primary inputs when I remove all the primary inputs I will now go and find

out what are all the nodes that are does not have any input node at all input edge at all.

So, what are the nodes  $N_1$ ,  $N_2$ ,  $N_3$  these I call it as level 0 or level 1 you see now I

remove all these nodes and all the edges on this nodes.

So, essentially we remove off all these nodes now we find out who are all the fellows who does not have any input nodes then it becomes N 4 N 5 N 6 I remove them then it becomes N 7 and finally, this unit. So, this is basically topological sorting. So, I will just do that shorting now as we see here. So, first I remove all the primary inputs these 3 nodes N 1 N 2 N 3 became level 0. Next I remove N 1 N 2 N 3 sorry we call it level 1 here and then we remove N 1 N 2 N 3. So, N 4 N 5 N 6 comes up and that is level 2 I remove N 4 N 5 N 6 please note only N 7 is removed because N 7 as an edge to N 8. So, this becomes level 3 and then N 8 comes in. So, this becomes level 4 this is basically what we call as the topological sorting of any directed acyclic graph what we do we remove the primary inputs we get all those saved nodes N 1 N 2 N 3 I remove. So, that I make it as level 1 I remove N 1 N 2 N 3 I get now the saved nodes like N 4 N 5 N 6.

Now, I mark it as level 2 then I remove N 4 N 4 N 5 N 6 now the only node left is N 7 N 8 cannot be inside. So, N 7 is level 3 and then N 7 is removed N 8 becomes level 4 are you getting this. So, this is how I do a topological sorting of a given circuit.

(Refer Slide Time: 14:30)

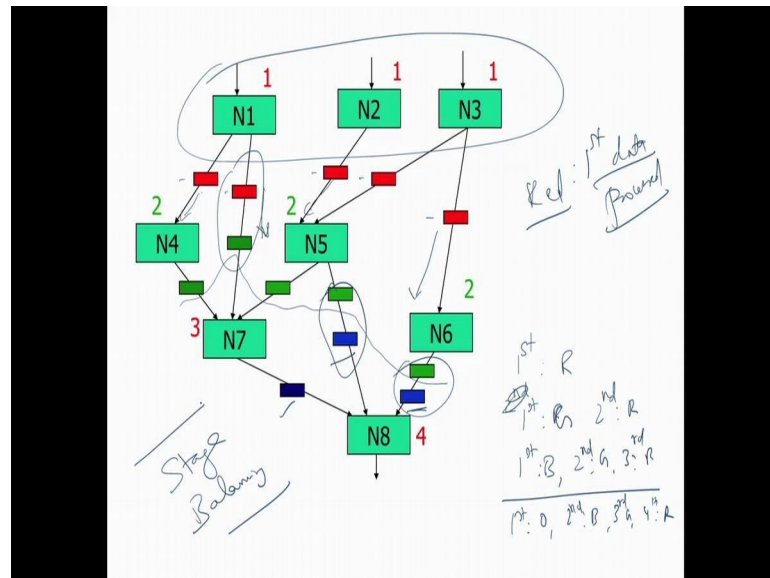
### A Pipelined Circuit

- If an edge connects two nodes of levels  $j$  and  $k$ ,  $j < k$ , then introduce  $k-j$  storage levels in between, in the edge.
- Each edge can carry one or more bits.

Now, what we do to pipeline this circuit we actually connect it with the edge. So, that

between vertices such that for every edge that is connecting between  $j$  and  $k$  that  $j$  is some level  $k$  is same level.

(Refer Slide Time: 14:51)



I will put  $k$  minus  $j$  storage levels in between for example,  $N_1$  is in level  $N_1$   $N_2$   $N_3$  are in level 1;  $N_4$ ,  $N_5$ ,  $N_6$  are in level 2,  $N_7$  and  $N_8$  are in level 3  $N_7$  is in level 3 sorry  $N_7$  is in level 3 and  $N_8$  is in level 4 now between  $N_1$ ,  $N_4$  there was an edge there is an edge now how many storage; storage I will put in between storage levels between  $N_1$  and  $N_2$  that difference is 2 minus 1 that is 1. So, I put one storage between  $N_3$  and  $N_7$  and  $N_1$  the difference is two. So, I put 2 storage namely the red and the green are you able to follow between  $N_1$  and  $N_7$  the it is 3 minus 1 2. So, I put 2 storage here.

Between  $N_2$  and  $N_5$  the difference is one because  $N_5$  is level 2  $N_2$  is level 1 then I put only 1 storage and similarly  $N_3$  and  $N_5$  there is only 1 storage I put here now coming to  $N_4$   $N_4$  and  $N_7$  is at 1 level. So, I put one  $N_7$  and  $N_5$  there is only one  $N_1$  and  $N_7$  it is greater than 1. So, I put one here between  $N_5$   $N_8$  this is 4 sorry this is 4 and this is two. So, 2 storages between  $N_6$  and the uh  $N_8$  again 2 storages between  $N_7$  and  $N_8$  there is one storage. So, what I have done I have done a topological sorting for every vertex I have

assigned the level and if there is an edge connecting level  $k$  and level  $j$  level  $k$  node and level  $j$  node we essentially put  $k$  minus  $j$  storage elements in that edge



right are you able to follow once I do this what will happen. Now the first set of data comes it gets it gets processed in the it gets processed at the first level and gets stored on the red; red as first data processed it gets stored on the red in the next cycle the second set of data comes to level one the first set of data is said on to level 2 nodes. So, this feeding this is feeding and this is feeding.

At the end of the second thing what will happen is your data will now go and stay on all the greens here right. So, the first set of data entered in the first clock pulse it came to the red block in the second clock pulse it moved to the 3 third block the green block sometime through node or sometime just direct from a buffer as you see here now. So, the first set of data it is in red. Now in the next clock pulse first 2 will be in red first will be moved to green after passing the level 2 processing or directly and first second will be in red at the end of third at the end of the third clock pulse the first will be in blue these entities first will be in blue second will be in green third will be in red correct and at the fourth clock pulse the first will be in output second will be in blue third will be in green fourth will be in red and. So, on are you able to follow right.

So, I have introduced I have introduced buffers in which or storage elements between edges right and I have I have introduced that many storage elements that is equal to the difference in the topological sorting once I put so many storage elements please note that first state of set of data comes it is executing and the result will be stored in the red. Now in the next clock pulse the second level nodes will start working on the red level nodes as their input and they will start producing green level nodes the green nodes. So, the first set of data came to red and then to green then the first set of data is in the green the second set of data can be in the red when the first of data is in the blue the second set of data can be in green third set of data can be in red and then are you able to follow right.

So, please also note that there are cases where I will just be transferring information I am not doing any processing between these 2 or these 2 here or these 2 I am just forwarding the data. So, that I keep phase of the speed of execution correct. So, this is this is basically what I intend to teach you today.

(Refer Slide Time: 20:37)

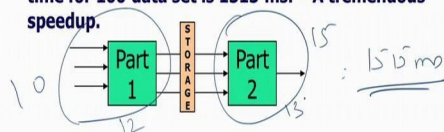
## Optimization

- Delay at every stage should be almost equal.
- The stage with maximum delay dictates the throughput.
- Number of bits transferred across nodes to be optimized, that shall reduce the storage requirements.

(Refer Slide Time: 20:44)

## Stage Time Balancing

If Part 1 takes 10 ms and Part 2 takes 15ms then, First data set finishes at 30ms and after every 15 ms one data set will come out – total processing time for 100 data set is 1515 ms. – A tremendous speedup.



If Part 1 takes 12 ms and Part 2 takes 13 ms then, First data set finishes at 26 ms and after every 13 ms one data set will come out – total processing time for 100 data set is 1313 ms. – A significant improvement.

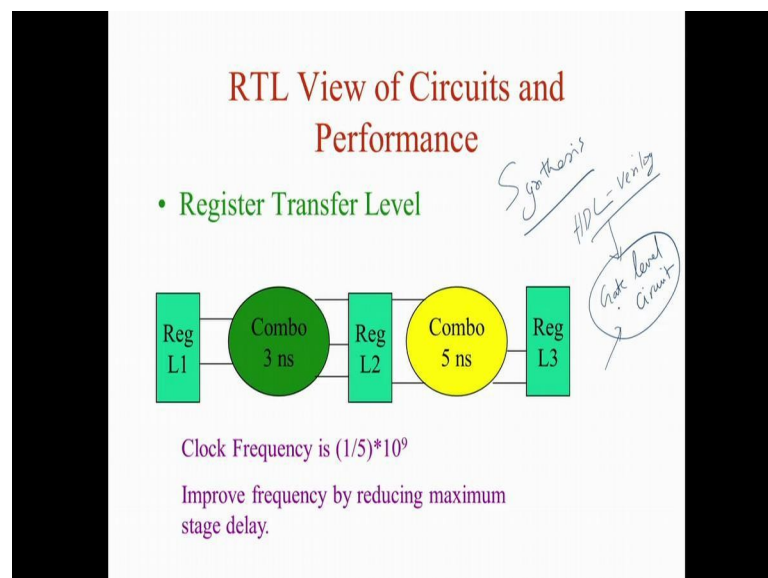
Now, one of the important point you should have noticed is that let us go to this problem if I

have made part as initially part one was 10 initially part one was 10 and part 2 was 15 and we got 1515 milliseconds right now suppose I go I do some circuit level techniques and move some 2 millisecond worth of data outside this then what will happen part one will become now 12 milliseconds while part 2 will take 13 milliseconds if.

So, what is the maximum time 13? So, when will the first data of set finish at 26 milliseconds 13 for this 13 for this plus I cannot change the clock and all and then after every 13 millisecond one data set will come. So, essentially the total processing time for this 100 data set is 1313 millisecond. So, we start started off looking at 2500 milliseconds from there we come to 2515 now it can be 1313 milliseconds. So, this is not because I; this is not because I adjusted the storage. So, that they become almost equal right once it was 10 another it was 15 I got 1515, now I made 112 another 13 they are almost equal when that stages are almost equal I get the maximum benefit is it.

Are you able to follow? So, when we do this pipelining when we are doing this pipelining please note that N 2 to N 8 right or modules and they should have the same complexity in terms of time one should now consume large delay one should not consume. So, essentially; that means, every stage is balanced we call this word stage balancing I need to balance the stage. So, that I get good speed and by not balancing the state stage we landed up with 2000; 1515 milliseconds now with balancing the state we can lead up to 1313 is it ok.

(Refer Slide Time: 23:08)



Now, let us talk about what is register transfer. So, already we have seen in a pipeline circuit there is some register feeding a combination circuit which will again feed their

register which will again feed combination circuit. So, the way we described the design is how some set of data move from one register to another that is why we call this as RTL register transfer level the entire modelling of this is done through r t l. So, so what would be the clock frequency here it is  $1 \text{ by } 5 \text{ into } 10^9$  correct because 5 nanosecond is  $1 \text{ by } 5 \text{ into } 10^9$  power minus nine will be  $1 \text{ by } 5 \text{ into } 10^8$  that is the clock frequency right. Suppose I go and make it as 4 nanoseconds here and 4 nanoseconds here and 4 nanoseconds here instead of 5 and 3 then certainly my frequency will become  $1 \text{ by } 4 \text{ into } 10^9$  it is  $0.25 \text{ into } 10^9$  right. So, if I want to improve my frequency I need to balance my stages.

(Refer Slide Time: 24:21)

## High Speed Circuits

- Carry Ripple to Carry Look ahead
- Wallace Tree Multipliers
- Increased Area and Power Consumption
- Lower Time delays
- Why Laptops have lesser frequency ratings than Desktops?

• Reference: Cormen, Lieserson and Rivest, (First Edition) Introduction to Algorithms (or) Computer Architecture by Hamacher et al.

So, that is what many of the carry ripple carry look ahead Wallace tree multipliers all of them when you take you give a circuit, but what you do with a circuit is you and pipeline the circuit. So, that we see that you know the frequency increases or your time period decreases.

So, when I make this 4 and 4 I use some circuit technique to make it 4 and 4 or I improve the technology. So, that 3 becomes 2 and this 5 becomes 4 automatically my frequency is

going to increase right if I derived increases my frequency is going to increase right and that is also one of the reason why laptops have lesser frequency ratings than desktop

because I because these delays would be quite large in laptops I will not give lot power. So, this cannot work very fast laptops are power hungry. So, if I give less power they will work slowly and then that is why they are still slower than the clock frequency is slower right. So, this; what I wanted to cover today. So, what we have done.

So, far is we have learnt what is pipelining right and we there is a need for storage between 2 parts of the circuit to create isolation and the data can be flowing from one end to another and we model the entire operations as a register transfer level where in that data moves from one register to the next register and so on so forth right in different clock pulse. So, the; was we described the circuit that itself is called register transfer level. So, RTL description RTL view what it means data flows from one registers to another stage during the clock pulse and then for me the most important thing is I should have a balance in the pipeline one stage should not be extremely slow then the entire pipe will get off I completely screwed up. So, we need to have that balance.

We have given some examples of this Wallace tree. So, the example I gave here is very close to a Wallace tree multiplier. So, now, given any combinational circuit we can do a topological sorting get the level numbers then for every edge connects the  $i$  and  $j$  the level number of  $j$  minus level number of  $i$  that many storage elements we can put and it automatically gives you a combinatorial arithmetic circuit which could be pipelined or which is already piped.

So, this is now we go and improve the speed, but pipelining is very very important is very very useful when I have consistent flow of data are you getting this if I do not have consistent flow of data once in a blue moon I am going to do then there is nothing that I will get here right. But if I have consistent flow of data one after another it is coming for this then pipelining becomes extremely effective now when we look at the benchmarks right traditional benchmarks that we have seen there are lot of benchmark almost all the benchmarks will have significantly large parts of the system which will which will allow you to do continuous you know instructions in to the c p u or continuous set of data in to the system and then there these type of these type of parallelization or these type of pipelining will be very very useful.



So, the most thing is that we need to get to equal. So, what happens is when we design the circuits when we design the circuits we design it in some hardware description language and we go and do a process called synthesis of the circuit we do something called synthesis of the circuit what is synthesis you write something in an hardware description language like very log or whatever you did last time in your course and that is converted in to a gate level circuit right now when I convert it to a gate level circuit there are tools there are fabrication libraries basically come and tell us.

So, these are all the gates involved these are the ways it is connected it will give you what would be the delay. So, I can estimate the delay while I am designing the system at a very early stage the moment I have a description I can give it to a tool and that tool will convert it to gates. And then you will know that you are going to fabricate in some fabrication unit at fabrication unit will tell you what are all the delays of the different components and then you can estimate this delays the movement I can estimate these delays I can get a feel of the frequency right.

So, very early in your design stage itself you will get what would be the ultimate frequency we will get a cap on what is what is a maximum frequency to which it can work and if that does not suit your actuality then you probably start working at it at a very early stage to improve your frequency is it fine. So, this is something which is very interesting here.

Thank you.