

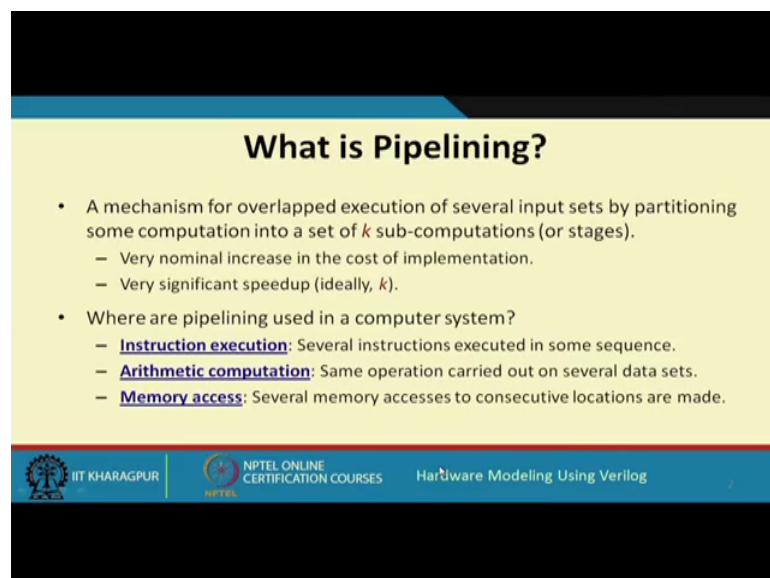
Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 32
Basic Pipelining Concepts

So, you recall whatever we have discussed during the course of the last few weeks. We have seen various ways of designing digital circuits and systems, combinational circuits, sequential circuits; very simple circuits, slightly more complex circuits. Now, one can you should remember whenever we are trying to design high performance systems, where the speed of operations and something called throughput; number of calculations that we are able to do per unit time that becomes important. We often go for a technique called pipelining, so we shall be looking at several examples of pipelining, how we can model pipelines in verilog and so on.



But in this lecture, let me give you a very brief overview about the basic concepts in pipelining, what it is and how does it give us an advantage in terms of speed up and increasing throughput. So, the topic is basic pipelining concepts.

(Refer Slide Time: 01:33)



What is Pipelining?

- A mechanism for overlapped execution of several input sets by partitioning some computation into a set of k sub-computations (or stages).
 - Very nominal increase in the cost of implementation.
 - Very significant speedup (ideally, k).
- Where are pipelining used in a computer system?
 - **Instruction execution**: Several instructions executed in some sequence.
 - **Arithmetic computation**: Same operation carried out on several data sets.
 - **Memory access**: Several memory accesses to consecutive locations are made.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, what is pipelining? Pipelining essentially is a method for overlapped execution of several input sets. Like what I mean to say is that, you see whenever you have some kind

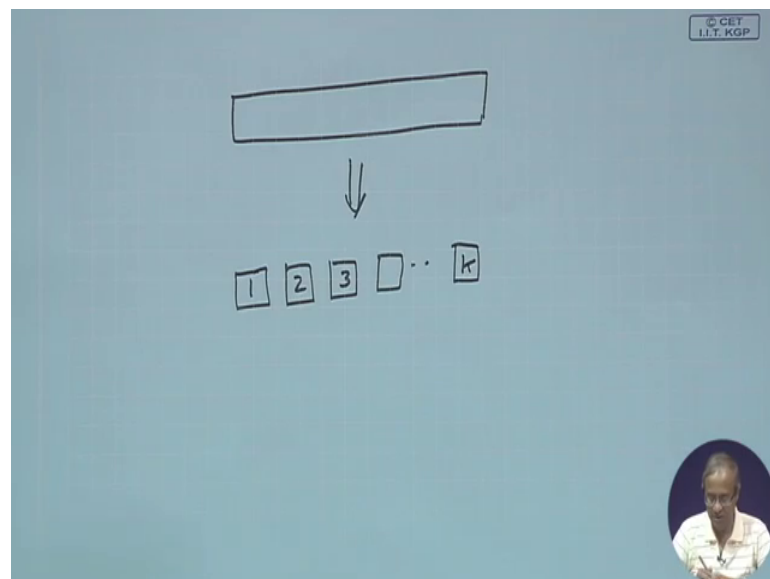
of competition; something you want to compute, there will be some input data you are applying and you are finally, getting some result.

Now, your computation maybe such that; you have to do the same kind of calculation on a large number of input data, so the data are coming one after the other. So, in that case what we are saying is that the computations that are going on; for the consecutive data items, they are somehow overlapping in execution. Because you see conventionally without pipelining what you will do?

First data comes we finish our computation; generate the result then only we take the second input. Again we do the computation; generate result then we take the third input, there is no overlap. Now, what I am saying is that; before the computation on the first data is complete; we have already started something on the second data. This is something which is called overlapped execution; we shall see how.

So, this overlapping normally we just express in terms of something called sub computations or stages. Well I shall be explaining through some examples; so, what you are saying is that we have some computation.

(Refer Slide Time: 03:27)



Let us say we represent it like this and as an alternative, we divide the computation into smaller pieces called stages. Let us say there are k number of stages; this is how what we are doing the partition. Now, in this kind of a partitioning the way pipelines are

implemented, the cost of implementation does not increase appreciably; very low increase. But the advantage is that we shall see how it comes, the speed up can be very significant; it will almost be equal to k how many times we have divided the computation into the number of partitions.

Now, in a computer system; pipelining can be used to speed up operations in various different places; when instructions are executed this is called instruction execution. When some computations are going on let us say arithmetic computation addition, subtractions, multiplications there also we can use pipelining to speed up and memory access. When a large number of memory accesses are going on consecutively using pipelining; again we can speed up the overall access time of the memory using some efficient technique.

(Refer Slide Time: 04:57)

A Real-life Example

- Suppose you have built a machine M that can wash (W), dry (D), and iron (R) clothes, one cloth at a time.
 - Total time required is T .
- As an alternative, we split the machine into three smaller machines M_w , M_D and M_R , which can perform the specific task only.
 - Time required by each of the smaller machines is $T/3$ (say).

Let us take a real life example to illustrate how pipelining helps; this is nothing to do with a computer system or a digital system. Let us take this rectangular box indicates some operation that you are doing. What is the operation? This corresponds to machine M ; let us say which can wash, dry and iron clothes one at a time. Suppose, you have manufactured such a machine which will take one cloth at a time, it will automatically wash, dry and iron and it will give you the nicely ironed cloth as the output.

Let us say the total time taken for this W , D and R steps all taken together is T ; capital T . So, if we have number of clothes, so how much time will it take total to complete for all

the N? For every cloth; we need T, so for N clothes we need N multiplied by T; this we represent as $T \cdot N$; means I have a single partition one is that 1.

Now, let us say that well you have thought that well instead of a single machine doing everything; because you see any way washing, drying and ironing are 3 different things. Washing involves soap and water, drying involve heater blowing air and so on and ironing involves pressing and so, on. So, there is nothing which is repeated in the steps; they are as such different. So, you identify that and what you come up with that well let us now do this.

Let us divide this machine into 3 smaller machines; one which can only wash, second which can only dry, third which can only iron. Well, now you can argue that well because we have divided the functionality and there is very little lower lap; the total cost does not increase appreciable; of course, there will be little increase in cost because we have to do a necessary packaging, we have to put it inside a nice cabinet and so on, but whatever they is there inside; the actual electromechanical systems they are not much different. So, we split the machine into 3 smaller machines M W, M D and M R; these 3 boxes, which can perform specific tasks.

Now here what happens let us see; now the total time was T. Let us say now; this individual stages; they will take one third time each $T/3$, $T/3$, $T/3$. So, the total time is still T; now see when a cloth comes, you wash it. After washing is complete; you give it for drying, now you see while drying is going on; this washing machine is free.

Now, the second cloth can be given for washing. Similarly when the first cloth goes for ironing, this second cloth can come for drying and the third cloth can go for washing. So, I will show you how means if we have a mechanism like this; this is called pipelining. As if there is a pipe; the clothes are flowing through a pipe and there are 3 stages; it first goes to W, then to D, then to R and then it goes out.

So, we will see how this calculation comes; for N clothes the total time taken here will be $2N + N$ multiplied by $T/3$.

(Refer Slide Time: 09:10)

$$T_1 = N \cdot T$$
$$T_3 = \frac{(2+N) \cdot T}{3}$$

Large N
 ≈ 1000

$$T_3 \approx \frac{1}{3} T_1$$

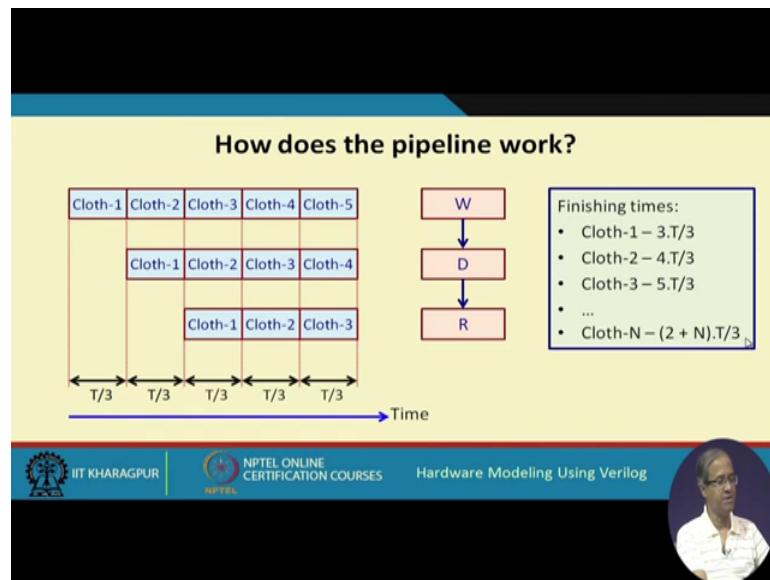
© CET
I.I.T. KGP

So, in the first case; the time was N multiplied by T and in the second case with 3 stages the time is coming as 2 plus N into T divided by 3.

Let us assume that the value of N is very large; let us say for example, I am washing 1000 clothes. So, with respect to 1000; you can ignore this 2; 2 is negligible. So, what you can say? You can say that your T_3 is approximately one third of T_1 ; T_1 was $N T$; T_3 is approximately $N T$ divided by 3. So, you have gained a 3 time speed up without investing on 3 machines; you see to get a speed up of 3; what you could have done initially that big machine you could have bought 3 copies.

Which means you would spend money 3 times; 3 times the cost, but now you have not done that; you have divided that big machines into 3 parts. It will involve a marginal increase in cost, but still the performance is improving 3 times; this is the essential idea behind pipelining; without increasing the cost appreciably we are getting very appreciable to speed up.

(Refer Slide Time: 10:54)

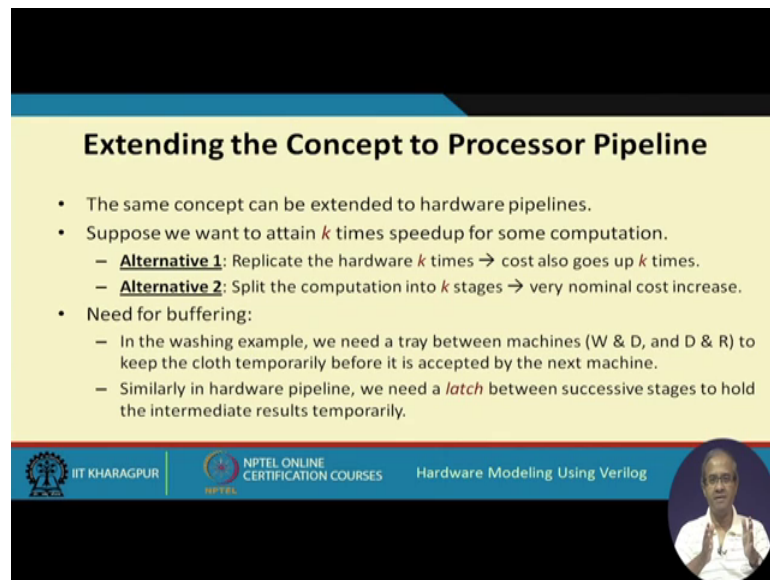


Now, let us see how this is coming. Let us look at the axis of time; this is washing, drying and ironing. So, cloth 1 comes here; cloth 1 comes for washing. So, it takes T by 3 time; after T by 3 cloth 1 comes for drying and the second cloth comes for washing; this requires again T by 3, then cloth 1 comes for this ironing, clothes 2 for drying, cloth 3 for washing.

Now, after T by 3 cloth 1 is done; so, cloth 1; you can take out then cloth 2 comes here cloth 3 comes here; cloth 4 comes here. After T by 3; cloth 2 can be taken out then cloth 3; can take. You see after this pipe is full; after every T by 3, T by 3, T by 3 time you can take out one cloth. So, one cloth per time T by 3 is being generated as output.

So, for one cloth the first cloth will take T by 3, T by 3, T by 3; 3 into T by 3. Second cloth will take one more time; 1, 2, 3, 4; 4 into T by 3, third cloth will take 5 into T by 3. So, proceeding in this way the N th cloth will take 2 plus N T by 3; that is what we showed in the expression.


(Refer Slide Time: 12:22)



Extending the Concept to Processor Pipeline

- The same concept can be extended to hardware pipelines.
- Suppose we want to attain k times speedup for some computation.
 - **Alternative 1:** Replicate the hardware k times \rightarrow cost also goes up k times.
 - **Alternative 2:** Split the computation into k stages \rightarrow very nominal cost increase.
- Need for buffering:
 - In the washing example, we need a tray between machines (W & D, and D & R) to keep the cloth temporarily before it is accepted by the next machine.
 - Similarly in hardware pipeline, we need a *latch* between successive stages to hold the intermediate results temporarily.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



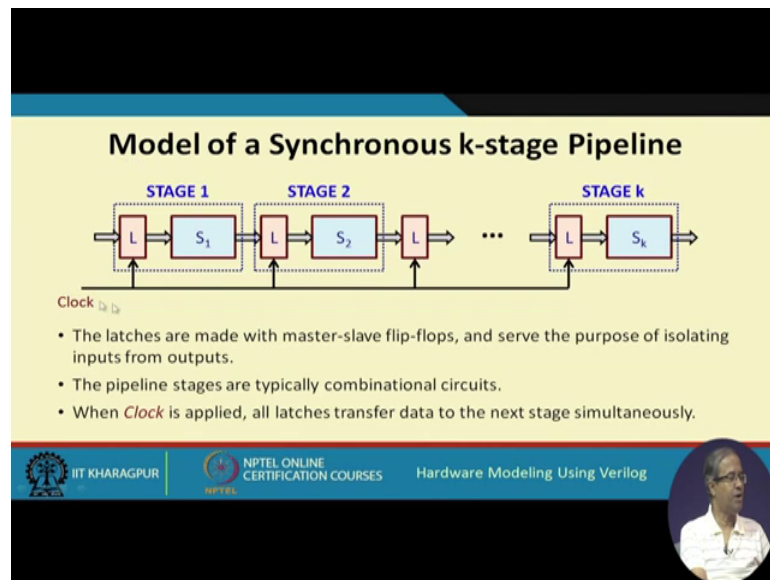
Now, extending this concept to a processor; suppose I am building a computer, I want to speed up instruction execution; that is called a processor pipeline. So, instead of executing one instruction at a time; I am dividing the total execution process in two stages and I am overlapping the execution in the same way.

So here again I am just repeating; so, we had two alternatives, one is we can repeat the replicate the hardware k times, which also will increase a k times increasing the cost. But, secondly a second alternative we are not replicating rather you are splitting the hardware into smaller pieces; these are called stages very nominal cost increase. But one thing is required which we have not considered, you see you consider the washing example.

Suppose, there is a washing machine; there is a dryer, suppose washing machine has finished with a cloth, but dryer is not yet ready; it is still drying the last cloth. So, that cloth has to be temporary put in some kind of a tray or a buffer. So, there has to be some trays between the machines which can temporarily stored the clothes before it can be fed to the next machine.

So, this is the need for buffering. So, we need a tray between washing and drying machines and between drying and ironing machines. So, in the same way when we talk about a hardware pipeline; we need a latch or a register between successive stages which can store the latch, store the values; before it can be processed by the next stage.

(Refer Slide Time: 14:28)



This is how a hardware pipeline will look like. So, a total computation I am dividing into k number of stages S_1, S_2 ; up to S_k ; there are latches between the stages and there is a clock; clock is activating the latches. So, what is the purpose of the latch? You see when the first data comes, the data get stored in this latch and S_1 is working on that.

After S_1 has finished doing the computation; the result it will be storing in this latch. At the same time, the next data will be stored on this latch. So, when S_1 is working on the second data; S_2 is working on the result of the first data. Now, if this latch was not there; then what might have happened is that while S_1 is working on the next data, the output could be changing.

So, the input of S_2 could also be changing resulting in wrong computation. So, we are using the latch to hold temporarily the previous data so that while S_2 is processing the data. The data does not change; similarly for the other stages and these latches are typically master slave flip flops, which serve the purpose of isolating inputs from outputs. And the stages S_1, S_2 to S_k ; these are typically combinational circuits; we shall see examples later.

So, when clock is applied everything gets shifted to the right by one place. So, S_1 goes to S_2 , S_2 goes to S_3 , S_3 goes to S_4 and so, on. So, when clock comes; data in a pipeline will move forward in a lockstep fashion; this is what really happens.

(Refer Slide Time: 16:32)

Structure of the Pipeline

- **Linear Pipeline:** The stages that constitute the pipeline are executed one by one in sequence (say, from left to right).
- **Non-linear Pipeline:** The stages may not execute in a linear sequence (say, a stage may execute more than once for a given data set).

A possible sequence: A, B, C, B, C, A, C, A

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

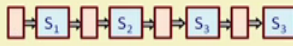
Now, talking about the structure of a pipeline; so, a pipeline can look like this as we have seen; it is called a linear pipeline. The stages are connected in a linear fashion; the output of one goes to the input of the other in exactly in a straight line fashion.

But for more complex systems of course, here we shall not be considering such examples. Pipelines can be non-linear also meaning that there will be stages; let us say A, B, C but data can flow not necessary from A to B, B to C; data can go from A to C also sometimes, C to B also and C to A also. So, there can be multiple paths the data can take for example, a possible sequence can be A, B say A, B; C B; C A; C A then it comes out; from this path. So, this can be a possible sequence of data computation, but these are for more complex cases; forget this for the time being.

(Refer Slide Time: 17:48)

Reservation Table

- The *Reservation Table* is a data structure that represents the utilization pattern of successive stages in a synchronous pipeline.
 - Basically a space-time diagram of the pipeline that shows precedence relationships among pipeline stages.
 - X-axis shows the time steps
 - Y-axis shows the stages
 - Number of columns give evaluation time.
 - The reservation table for a 4-stage linear pipeline is shown.



	1	2	3	4
S ₁	X			
S ₂		X		
S ₃			X	
S ₄				X

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

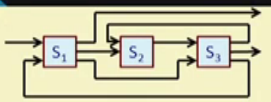
Now, you see these are some example pipelines; now how processing is carried out, this is depicted by a data structure which is called a reservation table; what does reservation table show? Reservation table shows the utilization pattern of the stages; like in the various time steps; how are the pipeline stages utilized, in time step 1; which stage you are using? In time step 2; which stage? In time step 3; which stage? And so, on; this is depicted nicely in the form of a tabular fashion in the reservation table.

Now, in the reservation table; the X axis shows the time steps and Y axis shows the stages. The number of columns will give you the total number of stages; that means, the evaluation time. Let us take an example of a fourth stage linear pipeline; suppose I have a four stage pipeline at this pink boxes are the latches; the reservation table can be shown like this; these are the stages.

So, in time step 1; you are using X; S 1; time step 2 you are using S 2; time step 3; S 3, time step 4; S 4. So, it will be a diagonal matrix kind of a thing; X will go like this. For a linear pipeline, the reservation table will always look like this.

(Refer Slide Time: 19:22)

- Reservation table for a 3-stage dynamic multi-function pipeline is shown.
 - Contains feedforward and feedback connections.
 - Two functions X and Y.
- Some characteristics:
 - Multiple X's in a row* :: repeated use of the same stage in different cycles.
 - Contiguous X's in a row* :: extended use of a stage over more than one cycles.
 - Multiple X's in a column* :: multiple stages are used in parallel during a clock cycle.



	1	2	3	4	5	6	7	8
S ₁	X				X			X
S ₂		X		X				
S ₃			X		X		X	

	1	2	3	4	5	6
S ₁	Y				Y	
S ₂			Y			
S ₃		Y		Y		Y

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

But for a non-linear pipeline reservation table can be more complex because as we have seen for a non-linear pipeline, there can be feed forward like S 1 to S 3 or feedback S 3 to S 2, S 3 to S 1 such connections.

Let us say suppose; I am computing two functions X and Y. Now in the function X; my pipeline utilization can be like this; first the data comes it goes to S 1, then S 2, then S 3. From S 3; it again goes to S 2; you see from S 3 to S 2; there is a path, from S 2 again to S 3, S 3 to S 1, S 3 to S 1 there is a path and S 1 to S 3, S 1 to S 3 also there is a path. S 3 to S 1 and finally, it finishes from S 1 there is a path for the data to go out.

And similarly for the other computation Y; say from S 1 to S 3; S 2; S 3, S 1; S 3 then goes out. You see from S 3 also there is a path to go out; so S computation the results are generated here Y generated here. But here; means it is not very important for you to understand all this things because the examples we shall be taking the reservation tables for the linear pipeline. So, we shall only be considering linear pipelines.

So, in a reservation table multiple cross marks in a row means repeated use of the same stage in different cycles. Like here; S 1 will be used in time step 1 also in time step 5 contiguous X's in a row; suppose there are 2 X's side by side; which means a stage will be used for more than one cycle; multiple X's in a column it is not shown in the example; which means more than two stages may be active at the same time step.

(Refer Slide Time: 21:28)

Speedup and Efficiency

Some notations:

- τ :: clock period of the pipeline
- t_i :: time delay of the circuitry in stage S_i
- d_L :: delay of a latch

Maximum stage delay $\tau_m = \max(t_i)$

Thus, $\tau = \tau_m + d_L$

Pipeline frequency $f = 1 / \tau$

- If one result is expected to come out of the pipeline every clock cycle, f will represent the maximum throughput of the pipeline.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 10

Let us look into some simple calculation; so, for a pipeline what we have said very loosely is that; if there are k number of stages, we expect our speed up to be approximately k . Let us see how it comes; so, some notations let τ denote the clock period of the pipeline; t_i is the time delay for stage S_i .

(Refer Slide Time: 22:02)

Diagram of a pipeline with stages $S_1, S_2, S_3, S_4, \dots$ and delays $t_1, t_2, t_3, t_4, \dots$. Latches are shown between stages with delay d_L . The total delay for each stage is $t_i + d_L$.

Example values: $10, 12, 8, 14$ (stage delays), $d_L = 1$

Calculation: $\tau = 14 + 1 = 15$

Frequency: $\therefore f = \frac{1}{15} \text{ GHz}$

© IIT KGP

Let us say we consider a pipeline like this; we have stage S_1, S_2, S_3 and so on and there will be latches in between the stages. So, I am assuming that the time to compute S_1 is T_1 ; this is T_2 , this is T_3 , this is T_4 and so on. So, T_i is the time delay of stages i

and let us say d_L is the delay of a latch. So, what will be the total delay of a stage? T_1 plus the delay of a latch, so the delay of this will be T_1 plus d_L delay of this will be T_2 plus d_L and so on.

Now, you see we are having a common clock; so, we will have to look at which stage is the slowest. We find out the maximum of T_i ; call it τ_m and τ_m plus the latch delay; that will be the clock period of the pipeline clock period cannot be less than this and the pipeline frequency has to be reciprocal of that like. Let us take an example; suppose I have a pipeline with four stages, where the stage delays or let us say 10 nanosecond, 12 nanoseconds, 8 nanoseconds and 14 nanoseconds and the latch delays let us say 1 nanosecond.

So, what will be the maximum clock frequency? Here my τ will be max of the delays which is 14; plus the latch delay 15. So, my clock frequency can be $1/15$; so, many gigahertz because these are in nanoseconds; so, let us move on.

(Refer Slide Time: 24:26)

• The total time to process N data sets is given by

$$T_k = [(k-1) + N] \cdot \tau$$

$(k-1) \tau$ time required to fill the pipeline
1 result every τ time after that \rightarrow total $N \cdot \tau$

• For an equivalent non-pipelined processor (i.e. one stage), the total time is

$$T_1 = N \cdot k \cdot \tau$$

(ignoring the latch overheads)

• Speedup of the k -stage pipeline over the equivalent non-pipelined processor:

$$S_k = \frac{T_1}{T_k} = \frac{N \cdot k \cdot \tau}{k \cdot \tau + (N-1) \cdot \tau} = \frac{N \cdot k}{k + (N-1)}$$

As $N \rightarrow \infty$, $S_k \rightarrow k$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, what will be the total time that will be required to process N data sets? You see there are k number of stages in the pipe; so, k minus 1 time steps will be required for the pipe to fill up and after the pipe has filled up; there will be a one output generated every clock. So, k minus 1 for the pipe to fill up; after that one output every clock, there are N data sets; so, for all the results I need N clock cycles after that.

So, you see $k - 1 + N$; whole into clock period; this will be the total time taken to process; process N data. But, if we assume that we have an equivalent non pipelined processor, where we did not use pipeline then the time taken would be T_1 ; N number of say means N in into τ , N is the number of data and k in to τ we are assuming is the time to process every data.

So, we are ignoring the latch overrides. So, approximately it will be $N/k\tau$; the speed up will be the time taken on this processor divided by the time taken on the pipeline. So, it will be like this τ cancels out like this and as I had said as N becomes large. So, you can ignore this k as compared to N and this 1 . So, it becomes N/k divide by N approximately k ; so, speed up will be approximately equal to k .

(Refer Slide Time: 26:21)

• Pipeline efficiency:
 – How close is the performance to its ideal value?

$$E_k = \frac{S_k}{k} = \frac{N}{k + (N - 1)}$$

• Pipeline throughput:
 – Number of operations completed per unit time.

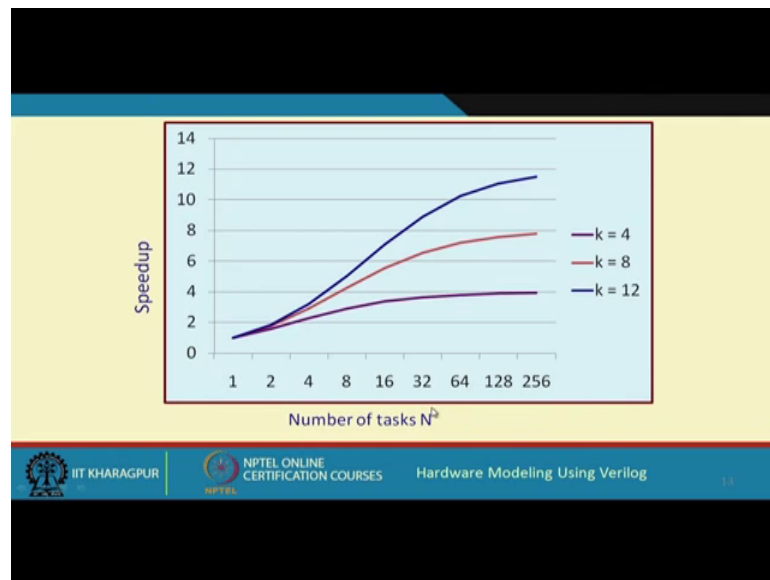
$$H_k = \frac{N}{T_k} = \frac{N}{[k + (N - 1)] \cdot \tau}$$

The slide also features logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and the course title 'Hardware Modeling Using Verilog'. A small circular inset image of a man is visible in the bottom right corner of the slide.

And pipe line efficiency is defined as S_k divided by k ; you see k is the maximum possible speed up and S_k is the actual speed up. So, you see your actual thing is $k - 1 + N$ into τ and this is N . So, if you just divide this up; this is defined as the pipeline efficiency.

Pipeline efficiency equal to 1 means; you are having an ideal value of k and pipeline throughput says that how many operations are completed per unit time; you are processing N number of data items, you are taking a time T_k . So, N divided by T_k will be number of operations per unit time; N divided by t_k . So, these are some simple calculations and this actually shows you the speedup curve; let us say tasks N .

(Refer Slide Time: 27:33)



So, as you plot speedup versus N for different values of k ; you will see that your curve looks like this. For k equal to 4; your speedup increases, but it levels to 4. So, as N increases; it goes approximately to 4; for k equal to 8, it approaches 8; for k equal to 12; it approaches 12. So, this case the maximum speed up that can be attained in a pipeline.

(Refer Slide Time: 28:09)

Clock Skew / Jitter / Setup time

- The minimum clock period of the pipeline must satisfy the inequality:
$$\tau \geq t_{\text{skew+jitter}} + t_{\text{logic+setup}}$$
- Definitions:
 - *Skew*: Maximum delay difference between the arrival of clock signals at the stage latches.
 - *Jitter*: Maximum delay difference between the arrival of clock signal at the same latch.
 - *Logic delay*: Maximum delay of the slowest stage in the pipeline.
 - *Setup time*: Minimum time a signal needs to be stable at the input of a latch before it can be captured.

So, there are some other issues like clock, skew and jitter. You see clock frequency does not only depend on the logic delay and the time you need to set and reset the latch. There are other times called skew, jitter and the setup time.

Skew means the clock signal might get delayed on its way to the different flip flops and latches. Suppose you generate the clock here, you will have to deliver the clock to different points; the length of the parts can be different. So, there will be different delays that the clock signal will experience before it reaches the different flip flops or latches. This is what is called skew; maximum delay difference between the arrival of the clocks.

Jitter is well on the same latch; sometimes there will be a variation in delay. Jitter arises due to environmental disturbances like noise in the power supply and so on. Logic delay; we have already seen the delay in the slowest stage in the pipeline the tau. And setup time, we have mentioned this earlier; setup time is the minimum time that you should make a signal stable at the input of the latch so that the latch can accept it. So, not only the latch delay; you will have to add the latch setup time also to it, these are some small delays that get added.

So, with this we come to the end of this lecture; where we just gave you an overview of what is pipeline? How it works? What are the basic principles behind it? So, now, that we have seen what a pipeline is and how it works; in the next couple of lectures we shall give you some very simple examples of pipelines and illustrate how we can actually quote these designs in verilog? How we can implement these designs in verilog and through simulation we shall see that actually it is working in a pipeline.

Thank you.