Advanced VLSI Design

Prof. D.K. Sharma
Department of Electrical Engineering
Indian Institute of Technology- Bombay

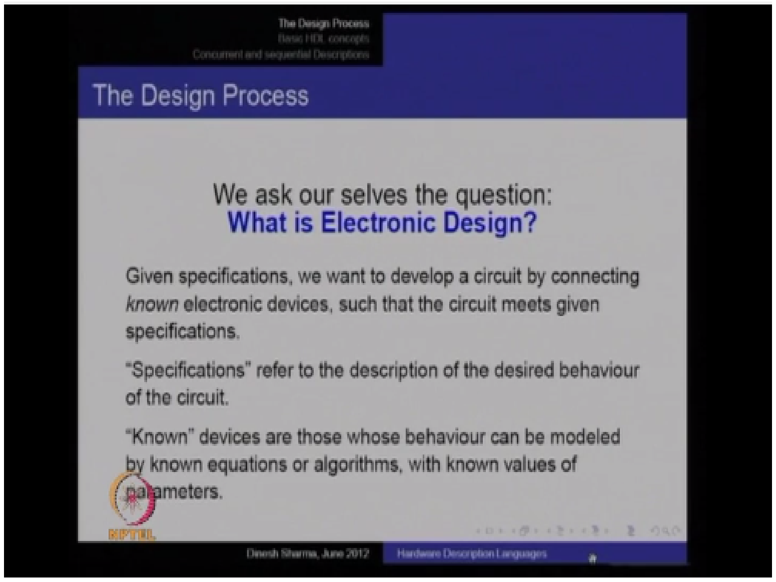
Lecture – 17
Introduction to Hardware Description Languages

Modern electronic design has become very complex and traditional design styles, therefore are inadequate to carry out these complex designs. Most of these designs these days, in particular in the digital domain, are all done using Hardware Description Languages. In this module, we shall look at Hardware Description Languages in general and then we will have a brief discussion on specific Hardware Description Languages.

In this lecture, we look at the underlying principles of all hardware description hardware description languages. Hardware Description Languages have evolved over time and the standards have changed; however, in this course, we shall look essentially what is the core of these designs or design styles. You are encouraged to practice designing using these and excellent textbooks and tutorial material is available on these.

I shall mention some of these as we go along and when we come to specific Hardware Description Languages. But in today's lecture, we want to look at the general underlying principles behind Hardware Description Languages and how they are connected to the Modern Design Process. Let us begin with asking ourselves the basic question what is electronic design.

(Refer Slide Time: 01:53)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

The Design Process

We ask our selves the question:
What is Electronic Design?

Given specifications, we want to develop a circuit by connecting *known* electronic devices, such that the circuit meets given specifications.

"Specifications" refer to the description of the desired behaviour of the circuit.

"Known" devices are those whose behaviour can be modeled by known equations or algorithms, with known values of parameters.

Dinesh Sharma, June 2012 Hardware Description Languages

There is no textbook answer to this; however, since the Hardware Description Languages are so closely linked to the requirements of electronic design, we should surely think about this question a little bit. So, what is electronic design. To my mind and this is not necessarily an encyclopedic definition, electronic design is that given specifications.

We want to develop a circuit by connecting now electronic devices such that the circuit meets the given specifications. The whole design process begins with somebody giving us specifications that I want an electronic circuit which will turn such and such lights on or which will turn this relay on or will display such and such things when such a thing happens. These are all specifications.

A designer is then supposed to sit down and think up an electronic circuit which is essentially just an interconnection of known electronic devices, such that this circuit has the behavior which meets the specifications. To me, this is electronic design; however, this is a somewhat lax definition. What are the specifications. We have introduced a new term specifications and which should actually understand what specifications are.

Specifications refer to the description of the desired behavior of the circuit. Obviously, there has to be a standardized way in which we specify the desired behavior.

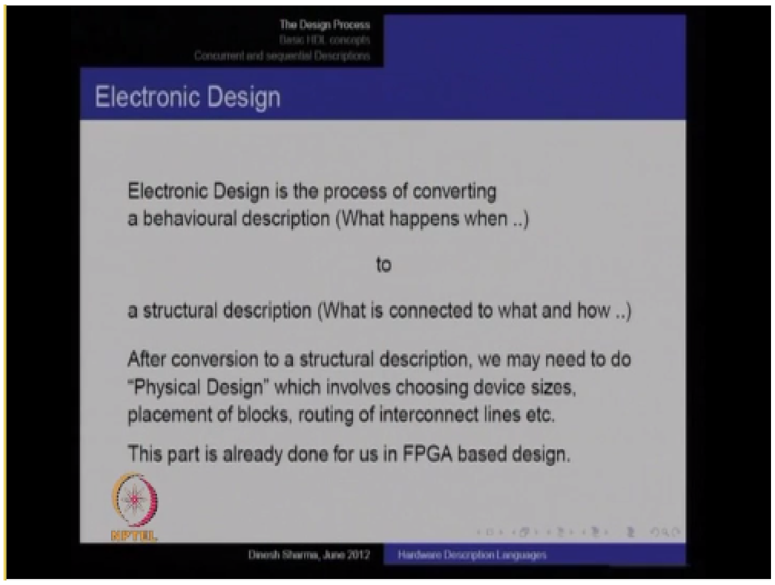
There is another term that we have used in this definition that is known devices. What are known devices. Known devices that those whose behavior can be modeled by known equation or Algorithms with known values or parameters. Essentially, what it means is that if the model and the model parameters of a device are known then this device is known.

The model and model parameters could be a device level at sub-circuit level or even at module level, that depends on the level at which we are simulating this. Once we have the model equations, then we can put many such known devices together and since we know the behavior of each one of these components and indeed the way that they are interconnected.

Given a good enough simulation program, we can now predict the behavior of this interconnected circuit and then we can see whether the behavior of this interconnected circuit is the same as the specifications. So, therefore if we have a standardized way of giving the specifications and then a program which takes devices whose models are known and can put these together and predict the behavior of the entire interconnected circuit.

Then we will be able to know before making the circuit whether our circuit is likely to meet the specifications or not. So, therefore in some way, electronic design is really the process of converting a behavioral description. What is a behavioral description. (1) It describes what happens when kind of things. (2) A structural description. What is a structural description what is connected to what and how?

(Refer Slide Time: 05:24)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Electronic Design

Electronic Design is the process of converting a behavioural description (What happens when ..)

to

a structural description (What is connected to what and how ..)

After conversion to a structural description, we may need to do "Physical Design" which involves choosing device sizes, placement of blocks, routing of interconnect lines etc.

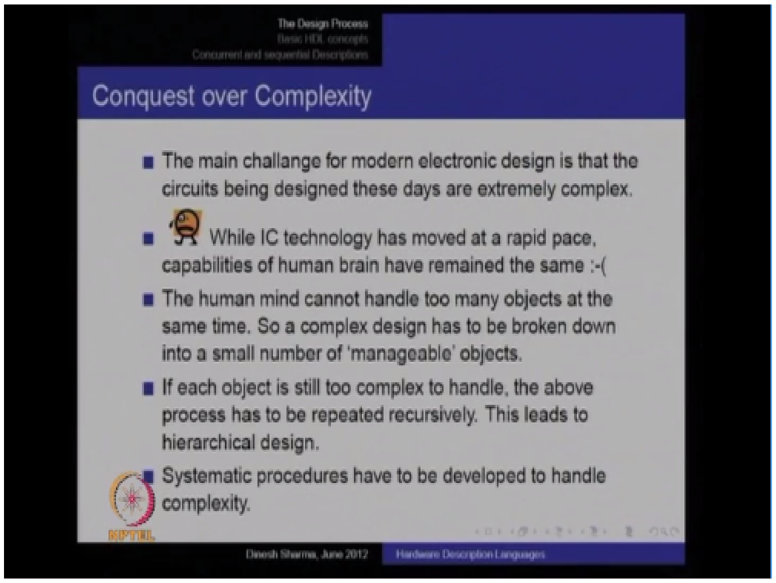
This part is already done for us in FPGA based design.

Dinesh Sharma, June 2012 Hardware Description Languages

So we begin with specifications which is nothing but a behavioral description. It says that if the door is opened, then the light comes on or whatever. So, that the behavioral description, what happens when kind of description, and the process of electronic design is the conversion of this kind of description to an equivalent description of electronic devices and their connection which is a structured.

After conversion to structural description, we may need to do one more step which is physical design which involves choosing device sizes, placement of blocks, routing of interconnect lines, etc. This part is already done for us if the design is on an FPG. On the other hand, if it is want to design an (()) (06.26) or indeed a custom circuit, then these parts will have to be explicitly done. However, most of the work of design is essentially converting this specification to an interconnection.

(Refer Slide Time: 06:46)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Conquest over Complexity

- The main challange for modern electronic design is that the circuits being designed these days are extremely complex.
- While IC technology has moved at a rapid pace, capabilities of human brain have remained the same :-(
- The human mind cannot handle too many objects at the same time. So a complex design has to be broken down into a small number of 'manageable' objects.
- If each object is still too complex to handle, the above process has to be repeated recursively. This leads to hierarchical design.
- Systematic procedures have to be developed to handle complexity.

Dinesh Sharma, June 2012 Hardware Description Languages

So, what is the problem with this thing. What is the main challenge. The main challenge for Modern Electronic Design is that the circuits being designed these days are extremely complex. It is not so easy to take these huge complex specifications and to translate it to a circuit and while IC technology has moved at a rapid pace, capabilities of human brain have remained the same. There is no scaling of the human brain and we cannot handle too many objects at the same time.

Therefore, we must break down a complex design into a small number of manageable objects. At this time, we do not know how to design these objects; however, we can set down the specification of what each one of these objects will do. Then, we see that if we had such objects, what would the interconnection of such objects will do. Will it meet the specifications or not? At the end of it, if each object is still too complex to handle.

The above process has to be repeated recursively, that means we take the object and then divide it down into submodules as we did for the whole circuit. We continued this process till we find that the whole description contains only the known devices. So, in other words, initially we design the circuit by breaking down into modules and these need not be known objects. These are some things which are yet to be design.

However, for example, you might take a microprocessor. The microprocessor might be thought of as a combination of registers, ALU, bus interface unit, instruction decoder, or what have you.

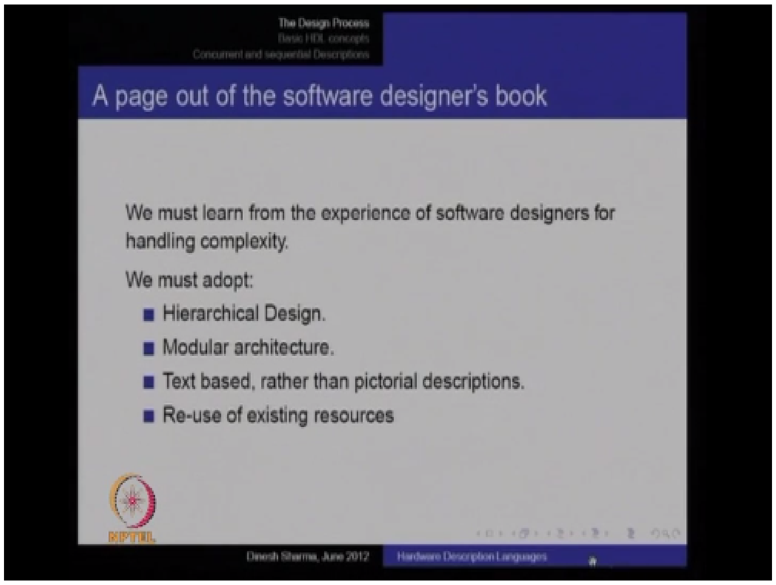
Right now, we do not know how to design the instruction decoder or the register file or the bus interface unit or what have you. However, we can write down that we intend to design an ALU which has essentially the capabilities which we desire.

The bus interface unit should have such and such behavior and assume that in due course we will have the design of these things done, so we can pretend that these modules are in fact known devices. Currently, they are not but it is reasonable to assume that they are known devices. We have a behavioral model of these and we can treat these as known devices. Then, we complete the circuit design as if these known devices were available.

Once the circuit works to our specifications, we are happy with it. Then, we look down on these objects and say whether these objects are really known or not. If they are not, then we further apply the same process to them till they have been simplified to a level where they consist of transistor, flip-flops, or whatever, all of which are known devices. Once we have done, then we have reduced the entire circuit to an interconnection of known devices and that is indeed the design procedure.

However, the complexity is high and therefore, systematic procedures have to be developed to handle this complexity. So, how do we learn how to handle this complexity and we take a page out of the software designer's book.

(Refer Slide Time: 10:10)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

A page out of the software designer's book

We must learn from the experience of software designers for handling complexity.

We must adopt:

- Hierarchical Design.
- Modular architecture.
- Text based, rather than pictorial descriptions.
- Re-use of existing resources

Dinesh Sharma, June 2012 | Hardware Description Languages

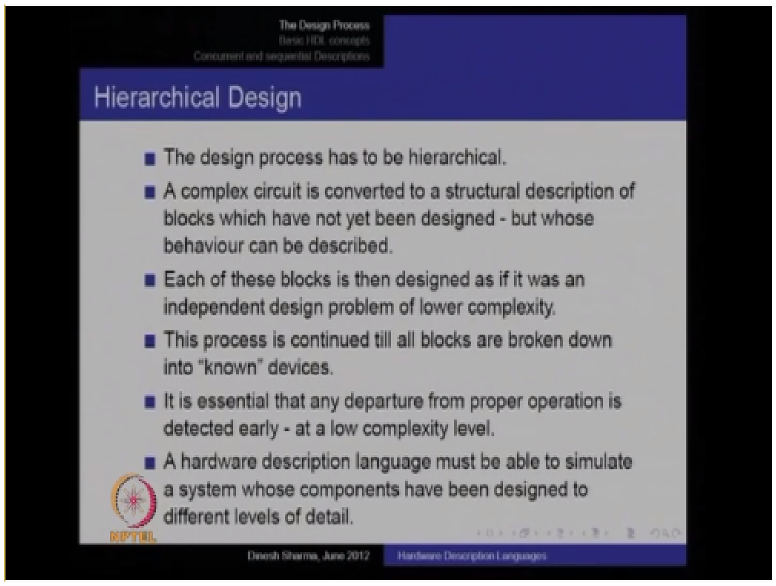
After all, very complex software has been written for quite some time. What have they done. Well we must adopt hierarchical design. We have already seen an example of this, that means you design the circuit in terms of modules, then design these modules in terms of sub-modules and so on. So, there is a hierarchy of complexity. The architecture must be modular that means the changes to one particular module should not affect the design of other modules.

The description should be text-based rather than pictorial because then we can use standard parsing technique programs and so on to handle these descriptions, and we must reuse the existing resources. So, for example, if the design of an ALU is already available, we can just add on other things to design our microprocessor. In fact, if the design of a microprocessor is already available.

We can build a much more complex circuit by using one or indeed even more of these microprocessors. So, therefore we must reuse existing resources. These are the lessons that we have learned from software. In software, the designer's hierarchical subroutines are written to be modular, so that if you change a subroutine the rest of the program need not change and the description is text-based.

And we have libraries and we do not rewrite libraries every time we write a new program, we use the libraries. So, these are the techniques which have been used by software designers for a long time and even for complex hardware designs, we must use these techniques.

(Refer Slide Time: 12:02)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Hierarchical Design

- The design process has to be hierarchical.
- A complex circuit is converted to a structural description of blocks which have not yet been designed - but whose behaviour can be described.
- Each of these blocks is then designed as if it was an independent design problem of lower complexity.
- This process is continued till all blocks are broken down into "known" devices.
- It is essential that any departure from proper operation is detected early - at a low complexity level.
- A hardware description language must be able to simulate a system whose components have been designed to different levels of detail.

Dinesh Sharma, June 2012 Hardware Description Languages

So, how is the hierarchical design carried out. We have agreed that the design process has to be hierarchical. A complex circuit is then converted to a structural description of blocks which have not yet been design, we have just discussed this. However, these blocks can be described behaviorally. They have not been designed that means their structural design is not know, but their behavior that means their specification has been set down.

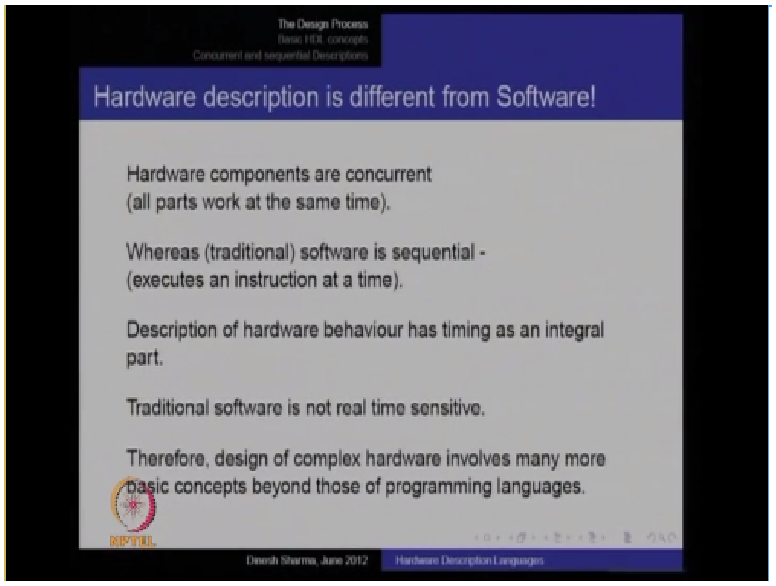
So, at this stage, the complex circuit is converted to a structural description of blocks which have not yet been design but whose behavior can be described. Each of these blocks is then designed as if it was an independent design problem of lower complexity. This process is continued till all blocks are broken down into known devices. It is essential that any departure from proper operation is detected early when the complexity level is low.

Therefore, once we go from one level of hierarchy to a next more detailed hierarchical level, at that time we must make sure that everything works properly at the less complex hierarchical level. And therefore if we have to design a language which will handle all these tasks, then the Hardware Description Language must be able to simulate a system whose components have been designed to different levels of detail.

For example, I have a circuit. I break it down into modules A, B, and C and I say that if the modules A, B, and C work as desired, then my circuit will work. Now, I design A to a higher level of detail and before I proceed any further, I would like to put this more detailed description of A with the less detailed descriptions of B and C, put them all together and see that the whole circuit works properly.

Therefore, the Hardware Description Language must be able to simulate the system whose components have been designed to different levels of detail. However, let us not get carried away in emulating software, after all hardware is different from software.

(Refer Slide Time: 14:24)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Hardware description is different from Software!

Hardware components are concurrent
(all parts work at the same time).

Whereas (traditional) software is sequential -
(executes an instruction at a time).

Description of hardware behaviour has timing as an integral part.

Traditional software is not real time sensitive.

Therefore, design of complex hardware involves many more basic concepts beyond those of programming languages.

Dinesh Sharma, June 2012 Hardware Description Languages

These are the major differences of hardware from software. First of all hardware components are concurrent. All parts work at the same time. Whereas traditional software is sequential, it executes one instruction at a time. Therefore, only one part of a very complex piece of software is active at a given time. In case of hardware, multiple parts are active and interactive. Description of hardware behavior has timing as an integral part.

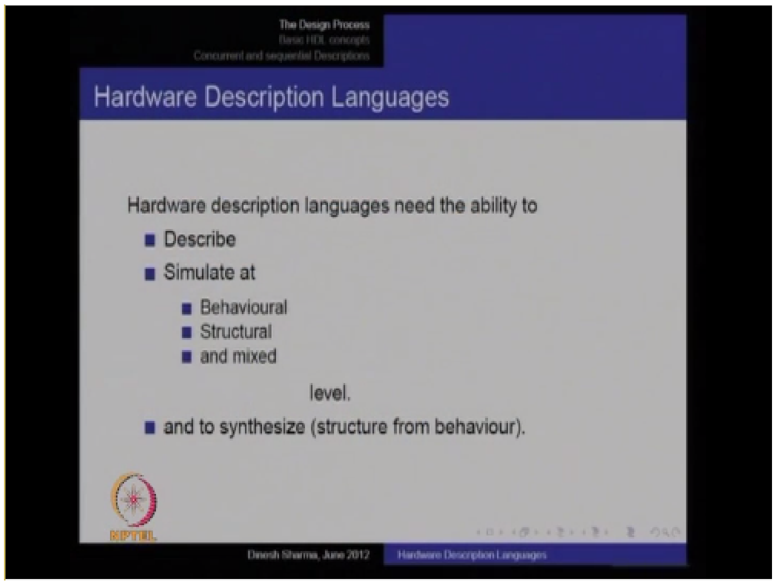
Changes in timing can change the entire behavior of the hardware. If the set input comes first or the recent input comes first, it can change the eventual behavior of a flip-flop. Therefore, the hardware is time sensitive. The timing of a signal is crucial to the operation of this hardware. On the other hand, traditional software is not real time sensitive. It may run slow, but it will eventually produce the same results.

Therefore, the design of complex hardware involves many more basic concepts beyond those of programming languages. All the concepts of programming languages will be used in Hardware Description Languages, but we must now introduce other components of Hardware Description Languages which take care of these requirements which are peculiar, which are specific to Hardware Description Languages.

And two of these are major, one is the timing is integral part of hardware and second that various components of hardware are active at the same time. So, therefore this parallel operation of different components is something which is peculiar to hardware description. Other than that, Hardware Description Languages are going to look very much like software, indeed so much so that sometimes we do not take the pains to understand that a hardware description is inherently different from software.

And tend to think of it as a software program which can often lead to very serious problems. Therefore, in this lecture, we are underlining those concepts which are specific to hardware. So, let us look at Hardware Description Languages and see what are the abilities that these must have.

(Refer Slide Time: 17:10)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Hardware Description Languages

Hardware description languages need the ability to

- Describe
- Simulate at
  - Behavioural
  - Structural
  - and mixed

  level.
- and to synthesize (structure from behaviour).

Dinesh Sharma, June 2012 | Hardware Description Languages

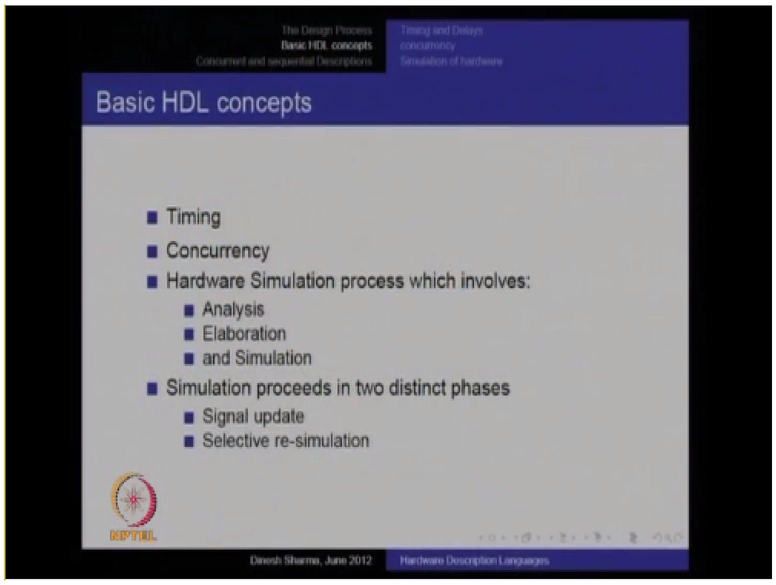
Hardware Description Languages need the ability to describe a circuit of course but also to simulate the behavior of the circuit at behavioral, structural and mixed levels, and finally to synthesize as far as possible the structural description from behavioral description. So, if we have a Hardware Description Language here, we would like eventually to have these capabilities. That means, we should be able to describe a very complex circuit, (()) (17:52) by itself.

But having described the very complex circuit, we must be able to simulate and the simulation should be possible if all parts of this are described structurally, behaviorally, or in a mixed way. Also once the description is in a standardized form which design tools can understand. Then all the mechanical work of converting a behavioral description to structural description should be done by a program to the extent possible and this activity is called synthesis.

When hardware is described specifically in a particular way which permits a synthesizer to convert this behavioral description to structural description, then that kind of design is called synthesizable hardware description. Not all description needs to be synthesizable; however, much of useful hardware design must be synthesizer. However, there is a large part of Hardware Description Languages which is not synthesizable, which is meant for testing, debugging circuits and so on.

So, therefore we must learn both kinds of Hardware Description Language and we must be aware that this part is synthesizable this is not, and for what is the non-synthesizable part to be used. The basic concepts involved in Hardware Description Language are timing, concurrency because we have seen that the part which is peculiar to hardware is that different components of this are working concurrently and the hardware simulation process which itself involves several stages.

(Refer Slide Time: 19:34)



Basic HDL concepts

- Timing
- Concurrency
- Hardware Simulation process which involves:
  - Analysis
  - Elaboration
  - and Simulation
- Simulation proceeds in two distinct phases
  - Signal update
  - Selective re-simulation

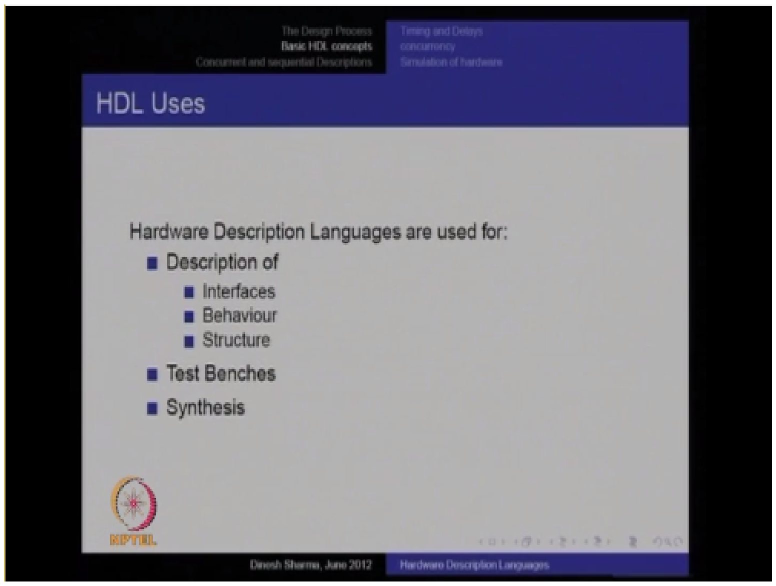
These stages are analysis that means you analyze the given description, elaboration that means you build up a circuit from the components that have been described, and finally the simulation. The simulation step itself leads to distinct phases. First is single update that means you bring the signals to their current values and because signals have changed, you must find out which components of this very complex circuit need re-simulation and re-simulate only those.

This re-simulation will cause changes in signal values and therefore you will go around this loop, updating signals at every time and then re-simulating only those components of the circuit which need re-simulation. So, this is how the simulation will proceed. It will proceed in a loop and in phased manner. That means all signals will be updated first, then we will analyze these updated signals to see which signals have changed; and if inputs have not changed, the output will not change.

Therefore, those parts of the complex circuit description need not be re-awoken. However, when the inputs have changed, we must go to those blocks which must be re-simulated. Re-simulate only those blocks and find out as a result of this re-simulation which signals will now change. So, this whole cycle continues till the circuits settles down to its final values. We shall actually see a tutorial of this.

Because it is very important to see this for a very simple circuit to see exactly what is going on behind the scenes when you use Hardware Description Language.

(Refer Slide Time: 21:54)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

HDL Uses

Hardware Description Languages are used for:

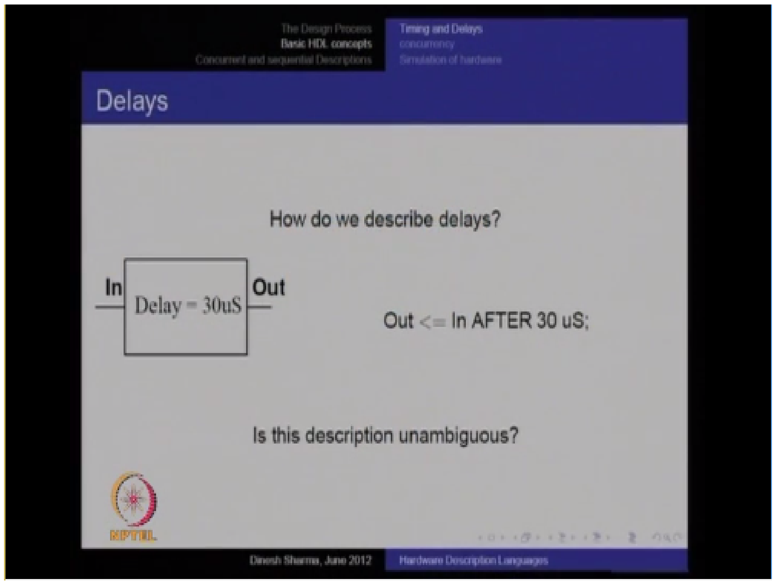
- Description of
  - Interfaces
  - Behaviour
  - Structure
- Test Benches
- Synthesis

Dinesh Sharma, June 2012
Hardware Description Languages

What are Hardware Restriction Languages used for? Well, these are used for description, description of interfaces, description of behavior and description of structure. These are also used for describing test benches which essentially test the circuit to see whether it meets specification or not and these are also used for synthesis, where if we have been careful in our description of the circuit.

Then rather than we reducing the behavioral description to a structural description, a program can be let loose on this description and a program itself will convert this behavioural description to a structural description. Let us look at the basic components which are different from software in a little more detail and timing is one of those. So, let us at delays and see how delays are actually handled in Hardware Description Languages.

(Refer Slide Time: 22:58)



Delays

How do we describe delays?

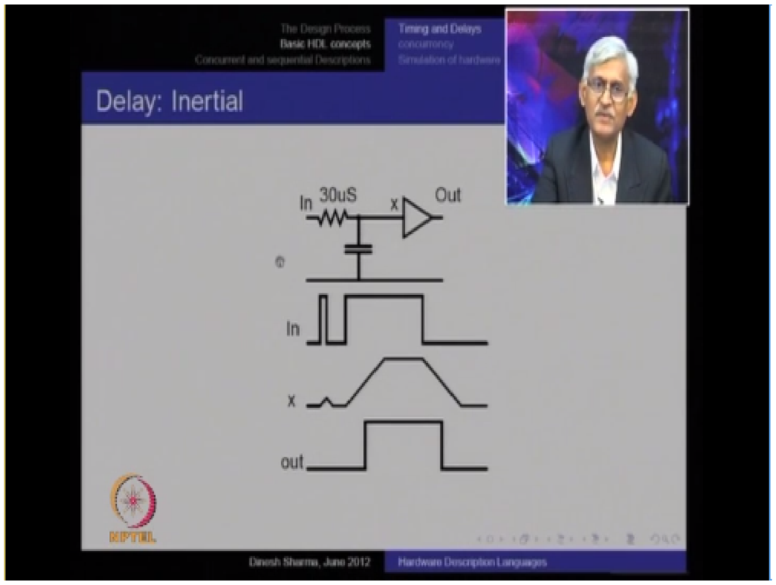
Out <= In AFTER 30 uS;

Is this description unambiguous?

How do you describe delays? Well, the actual syntax is not material at this stage when we look at Specific Deception Languages, then we will look at the syntax. Right now the syntax need not bother us; however, we choose some specific syntax here. This happens to be VHDL syntax and see how do we describe delays.

So, I have an input signal in hardware and I want to describe this delay of 30 microseconds, let us say, and the output is essentially the signal when it has been delayed by 30 microseconds. We need to examine this very important question that is this description unambiguous. Let us see why I raised this question indeed that this very simple description could at all be ambiguous. So, let us implement this delay in a circuit.

(Refer Slide Time: 23:59)



Here is a circuit. Notice that the circuit is not all digital. This delay part is in fact analog and these waveforms are somewhat stylized. These are not actual waveforms. However, the input is digital and so is the output. This is a discriminator. It looks like the voltage at level X and if it is below some threshold, it outputs are digital zero.

If it is above some threshold, it outputs are digital one, and the values of this delay element R and C are so adjusted that we get an overall delay of 30 microseconds. Now assume that we apply an input to this black box. Inside the black box is the circuit. Our black box of delay equal to 30 microseconds has this circuit and we apply to that circuit this waveform.

This waveform has a pulse which is much narrower than 30 microseconds and another pulse which is much wider than 30 microseconds, and let us figure out in our minds what is expected to happen. When the input is held at zero, then of course this capacitor is already discharged initially and it remains at zero.

So, the input remains at zero. As the input rises to 1, it starts charging this capacitor through this register; however, the time constant of this is large and the pulse is relatively narrow. As a result, the charging process has not proceeded too far when the pulse actually comes down. Because the pulse comes down, this point is now connected to ground and every capacitor starts discharging, so the capacitor discharges.

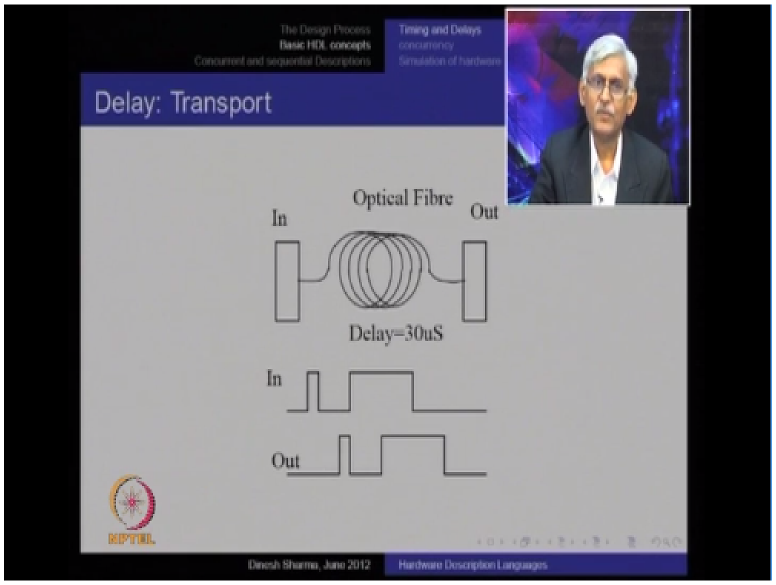
As I said before, this diagram is somewhat stylized the actual waveforms will be exponential charge and exponential discharge as you very well know. At this point, the wider pulse arrives and the capacitor starts charging again and because the pulse is wide enough, this charging will now be complete when the pulse goes down, the capacitor will start discharging and if this level lasts long enough, the discharging will be complete.

Along this charging and discharging waveform, we will pass that discrimination level of this discriminator. As long as the voltage is below that discrimination level, the output will remain at zero. However, as soon as the voltage exceeds that discrimination level, the output will become one and to remain one as long as the voltage at the input that node X remains higher than the discrimination level.

Once it falls below the discrimination level, then the output will come down to zero. As a result, we can expect to get a pulse like this at the output. Notice that the input is fully digital and so is the output. This point X is completely internal to that black box; and what has happened, well the narrow pulse has vanished but the wide pulse appears in a delayed form as we wanted. Okay, this vanishing of the narrow pulse is somewhat unexpected.

It is not clear from the description whether this is desired or not. We could have just as well implemented that delay in a different way.

(Refer Slide Time: 28:24)



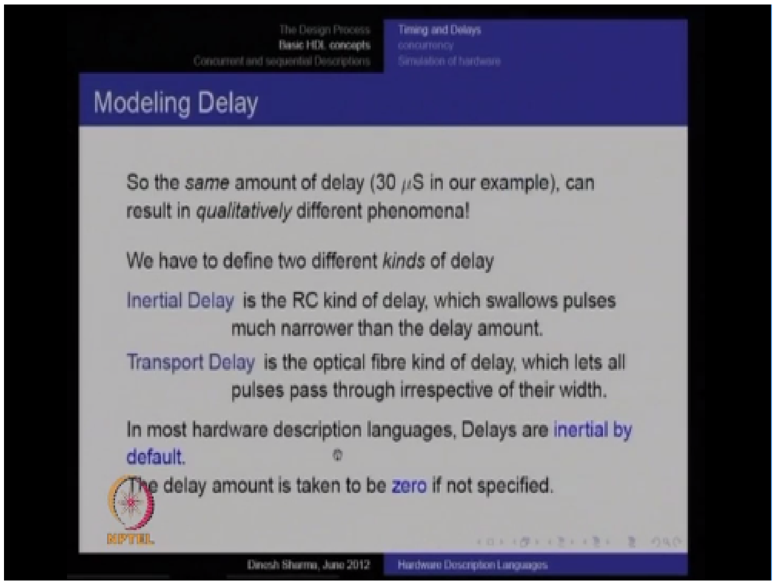
So, here is our black box again and somehow we are able to see inside this black box and what do we find inside that black box. In order to implement a delay of 30 microseconds, somebody has an interface here which converts the digital signal to light. This light is coupled to a long optical fibre which introduces a delay of 30 microseconds and there is an output circuit which converts this light back to a digital circuit.

What do you think will happen now? This input, this narrow pulse will result in a narrow flash of light. This wide pulse will result in a wider flash and both these flashes will arrive here 30 microseconds later. These light flashes will then be converted to an electrical signal and we will get an exact replica of the input at the output delayed by the amount of delay which corresponds to the length of this optical fibre presumably adjusted to give us a 30 microseconds delay.

So, notice that our black box does not behave the same in the two cases. Both cases were designed to introduce a delay of 30 microseconds. In the first case, this narrow pulse vanished. In the second case, it did not. Therefore, just saying that you delay by 30 microseconds is not adequate for accurate description of physical systems.

We have learnt that there are two kinds of delay. One kind in which pulses which are much narrower than the delay vanish and the other in which they do not. These kinds of delays are called inertial and transport delays.

(Refer Slide Time: 30:31)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Timing and Delays
concurrency
Simulation of hardware

Modeling Delay

So the *same* amount of delay (30 μS in our example), can result in *qualitatively* different phenomena!

We have to define two different *kinds* of delay

Inertial Delay is the RC kind of delay, which swallows pulses much narrower than the delay amount.

Transport Delay is the optical fibre kind of delay, which lets all pulses pass through irrespective of their width.

In most hardware description languages, Delays are inertial by default.

The delay amount is taken to be zero if not specified.

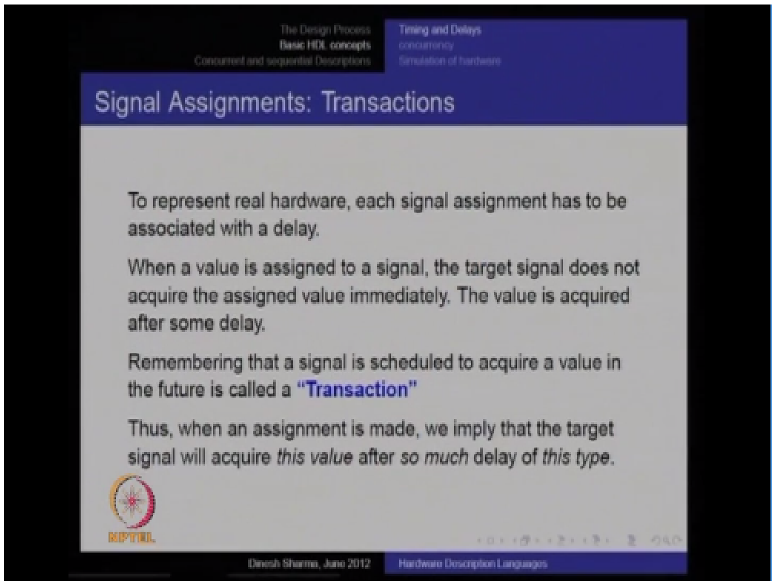
Dinesh Sharma, June 2012 Hardware Description Languages

So, the same amount of delay which was in our example 30 microseconds can result in qualitatively different phenomenon. In one case, the narrow pulse vanishes; in the other, it does not. So, therefore, we must define two different kinds of delay. Inertial delay is the RC kind of delay, the first circuit that we saw, which swallows pulses much narrower than the delay amount.

Transport delay is optical fibre kind of delay which lets all pulses pass through irrespective of their width. In most Hardware Description Languages, delays are inertial by default because these are no normally caused by RC kind of circuit. However, there are cases for example, specifying an input, which are then transport by default. So, if you do not specify the delay at all, then the amount is taken to be zero and the kind is taken to be inertial.

However, you do have the option of not only specifying the amount of delay like 30 microseconds here, but also the kind of delay and if you so choose you can specify either inertial or transport kind of delay. If you do not specify the kind, it will be taken in most circumstances as inertial; and if you do not specify the amount, it will be taken as zero. We should now look at the other thing which is specific to hardware description and that is how to handle concurrency.

(Refer Slide Time: 32:35)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware
Dinesh Sharma, June 2012
Hardware Description Languages

Signal Assignments: Transactions

To represent real hardware, each signal assignment has to be associated with a delay.

When a value is assigned to a signal, the target signal does not acquire the assigned value immediately. The value is acquired after some delay.

Remembering that a signal is scheduled to acquire a value in the future is called a **"Transaction"**

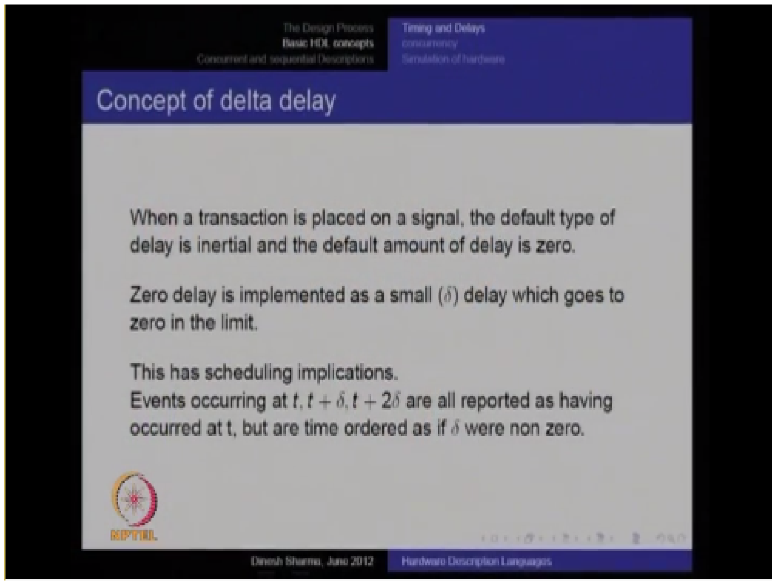
Thus, when an assignment is made, we imply that the target signal will acquire *this value* after *so much* delay of *this type*.

Now to represent real hardware, single assignment has to be associated with a delay. When a value is assigned to a single, the target signal does not acquire the assigned value immediately. The value is acquired after some delay. Therefore, there are two values associated with any signal, one is its current value and the other which is potentially different, it could be the same it could be different, but which is potentially different is the value that it will acquire in near future.

That means when you assign a new value to a single, it does not acquire this new value immediately. Indeed, you put a marker on that signal saying after this much delay this signal should acquire this value. This marker is called a transaction. Thus, when an assignment is made we imply that the target signal will acquire in future, will acquire this value after so much delay of this type.

So, the acquisition of a new value is not automatic; however, if you do not specify a delay, then as we said that the amount of delay will be taken to be zero. However, we must understand the case of zero delay a little carefully.

(Refer Slide Time: 34:27)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Concept of delta delay

When a transaction is placed on a signal, the default type of delay is inertial and the default amount of delay is zero.

Zero delay is implemented as a small (δ) delay which goes to zero in the limit.

This has scheduling implications.
Events occurring at $t, t+\delta, t+2\delta$ are all reported as having occurred at t, but are time ordered as if δ were non zero.

Dinesh Sharma, June 2012
Hardware Description Languages

When a transaction is placed on the signal, the default type of delay is inertial and the default amount of delay is zero. In fact, inside the language, this delay is not made zero. In fact, it is implemented as a small delay which is called a delta, and then when we report the result, we take the limit taking dealt equal to zero. So, why do we take this roundabout way. If the user wants a zero delay then just let us given zero delay.

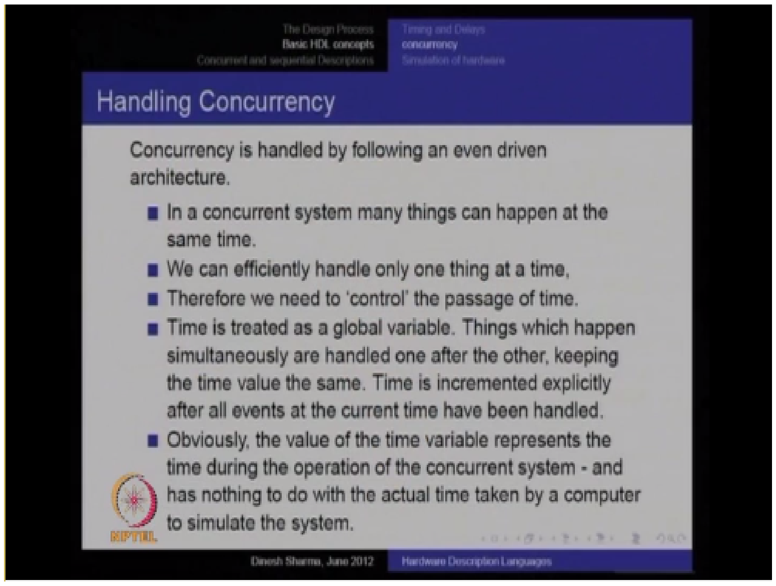
Why do we take this indirect way of actually implementing a delay and eventually making that delay equal to zero and the reason for this is that scheduling is very important in simulating hardware. It is very important to know what occurred before what and what occurred after what. Therefore, for scheduling purposes. We keep track of these delta delays. Something might occur at time T, something with a zero delay will cause a signal change and that signal change will take effect at T+ delta.

This change may cause yet another change and that change will occur at T plus 2 delta. The cause and effect will always have a delta time difference. Therefore, if we ever have to resolve which signal is earlier and which is later, which is important that times, then internally we always know which was the cause and which is effect, which signal arrived at delta time earlier and which signal arrived at delta time later.

Finally, when we report back to the user what is happening, the user always wanted zero delay and it is only at this time that we put delta equal to zero and report the final results as if delta was zero. But internally we keep track of all these deltas and treat as if delta is non-zero. Remember the actual value of delta is unimportant. It is some infinitesimally small time. The only point which matter is at how many deltas did this signal acquired its value.

So that we can do a time ordering. The time ordering of signals is indeed very important in Hardware Description Languages and as long as events are ordered in time, things are fine. If the user does not want to specify a delay, so be it, we shall order it in time using delta delays. So, this concept is very specific to Hardware Description Languages. You will not see such a concept in any programming language.

And indeed, lack of understanding of Hardware Description Languages very often springs from the lack of understanding of this delta delay concept. Now let us see how we handle concurrency.
(Refer Slide Time: 37:57)



The Design Process | Timing and Delays
Basic HDL concepts | concurrency
Concurrent and sequential Descriptions | Simulation of hardware

Handling Concurrency

Concurrency is handled by following an even driven architecture.

- In a concurrent system many things can happen at the same time.
- We can efficiently handle only one thing at a time,
- Therefore we need to 'control' the passage of time.
- Time is treated as a global variable. Things which happen simultaneously are handled one after the other, keeping the time value the same. Time is incremented explicitly after all events at the current time have been handled.
- Obviously, the value of the time variable represents the time during the operation of the concurrent system - and has nothing to do with the actual time taken by a computer to simulate the system.

Dinesh Sharma, June 2012 | Hardware Description Languages

Concurrency is handled by following an event driven architecture. So, in a concurrent system, many things can happen at the same time. It is not as if one latch is being set. many signals might change at the same time. However, since we will be writing Hardware Description Language which will run on a computer, we can efficiently handle only one thing in real time at a given time.

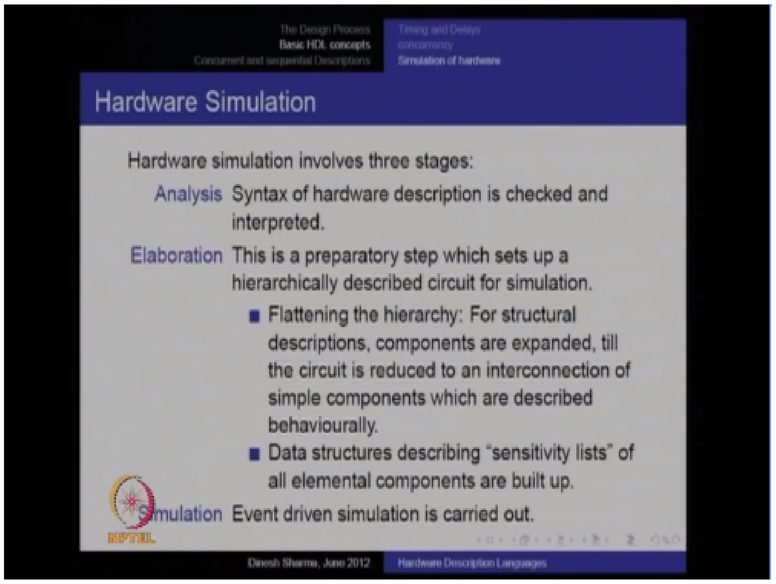
Therefore, we need to control the passage of time. In simulation, we are doing various things one after the other, but we wanted to appear as if all the things appeared at the same time. Therefore, we need to control the passage of time and we manage it by treating the time as a global variable. Things which happen simultaneously are in reality handled one after the other, but we keep the value of this global variable time the same.

Then, the time value is incremented explicitly after all the events which occur at the current time have been handled. So, in other words, the actual time as it passes when the program is running has no value whatsoever as far as the simulation time is concerned. The simulation time is changed discreetly it is brought to a level. Then in whatever time that it takes in real life we handle all the events which are said to have occurred at this time.

And then increment the time value inside the variable time is incremented to the next time value. Therefore, as far as the results of simulation are concerned, it appears that all the previous events before incrementing the value of the time occurred at the same value of time. Obviously, the value of the time variable represents the time during the operation of the concurrent system and has nothing to do with the actual time taken by a computer to simulate the system.

You may have a slow computer, you may have a fast computer, and the actual amount of time taken to simulate may be different. We are not talking of that time. We are talking of the time which is being simulated for the behavior of the circuit, okay and that is handled in this special way. So, let us see how hardware simulation then proceeds.

(Refer Slide Time: 40:44)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Hardware Simulation

Hardware simulation involves three stages:

Analysis Syntax of hardware description is checked and interpreted.

Elaboration This is a preparatory step which sets up a hierarchically described circuit for simulation.

- Flattening the hierarchy: For structural descriptions, components are expanded, till the circuit is reduced to an interconnection of simple components which are described behaviourally.
- Data structures describing "sensitivity lists" of all elemental components are built up.

Simulation Event driven simulation is carried out.

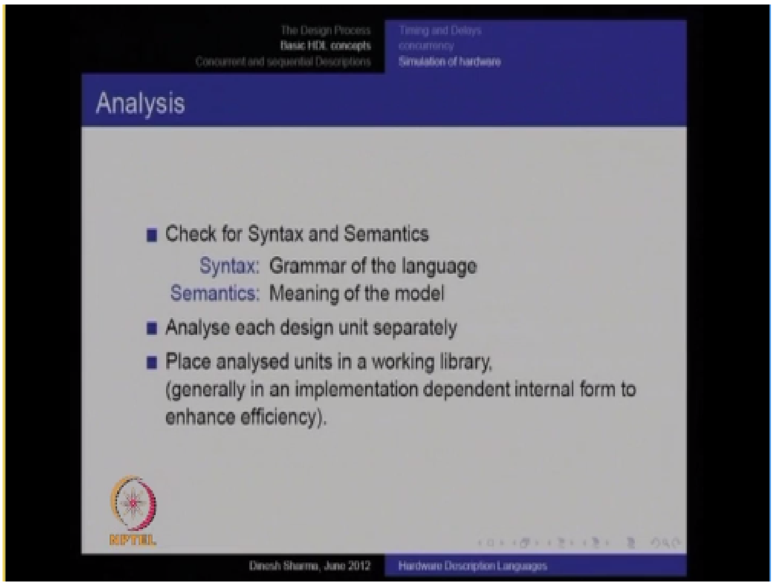
We have given a description in a text form conforming to a particular syntax described by some hardware language. So, the first step is when we want to run that simulation, the syntax of this hardware description is checked and interpreted. Only if this syntax is correct and make sense to the language that we will proceed to the next step.

The next step is called elaboration. This is a preparatory step which sets up hierarchically described system for simulation. So, there are many things which are done at this level. Even though the description follows a hierarchy, we flatten the hierarchy because eventually the circuit is the same. The circuit does not know that it was designed following this hierarchy, the circuit in the final circuit. So, we flatten the hierarchy.

For structural descriptions, components are expanded till the circuit is reduced to an interconnection of simple components which are described behavior, and then we set up data structures which describe sensitivity list of all elemental components. So, we know which component receives which signals as the input and this sensitivity list says that if this signal changes, then re-simulate this component.

Remember the circuit is very complex and we cannot afford to simulate all components at all the time. We re-simulate only those components whose inputs have changed or in other words, if there is a change in a signal to which this particular component is sensitive, and only after analysis and elaboration, this simulation is actually carried out. The simulation itself is a two-step process as I have described earlier.

(Refer Slide Time: 42:57)



The Design Process | Timing and Delays
Basic HDL concepts | concurrency
Concurrent and sequential Descriptions | Simulation of hardware

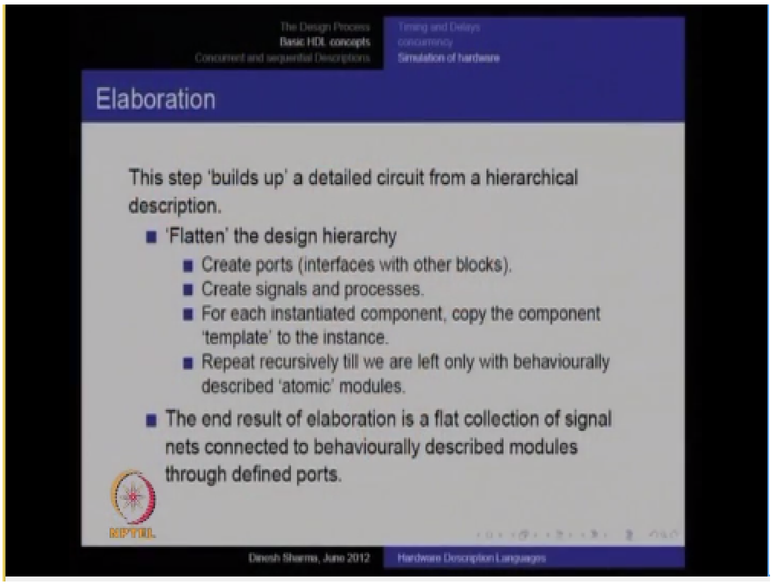
Analysis

- Check for Syntax and Semantics
  Syntax: Grammar of the language
  Semantics: Meaning of the model
- Analyse each design unit separately
- Place analysed units in a working library, (generally in an implementation dependent internal form to enhance efficiency).

Dinesh Sharma, June 2012 | Hardware Description Languages

So, what does analysis do. It checks for syntax and semantics. Syntax in the grammar of the language, semantics of the meaning of the model. You analyze each design unit separately. You place analyze units in working library and generally in an implementation dependent internal form to enhance the efficiency. Remember your description was a text description and it need not be kept in this form. Then you go to elaboration.

(Refer Slide Time: 43:30)



The Design Process | Timing and Delays
Basic HDL concepts | concurrency
Concurrent and sequential Descriptions | Simulation of hardware

Elaboration

This step 'builds up' a detailed circuit from a hierarchical description.

- 'Flatten' the design hierarchy
  - Create ports (interfaces with other blocks).
  - Create signals and processes.
  - For each instantiated component, copy the component 'template' to the instance.
  - Repeat recursively till we are left only with behaviourally described 'atomic' modules.
- The end result of elaboration is a flat collection of signal nets connected to behaviourally described modules through defined ports.

Dinesh Sharma, June 2012 | Hardware Description Languages

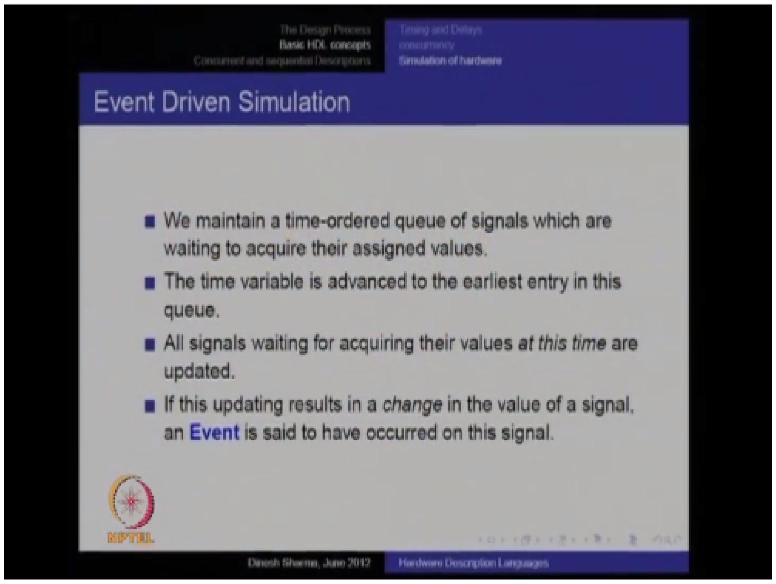
This kept builds up a detailed circuit from a hierarchical description, you flatten the design hierarchy. You create ports which are interfaces with other blocks, create signals and processes. For each instantiated component, you copy the component template to this particular instance. Remember you may not have just one flip-flop.

You describe a flip-flop for once and now you may place say 20 flip-flops in your circuit. Each instance is different. The description occurs only once but the actual copying has to be done at the elaboration level. You are elaborating the entire circuit. So, you take the template as the description and make copies of this template for each instance of the flip-flop, and you repeat this recursively till we are left only with behaviorally described atomic modules, whatever these can these may be.

For example, basic gates might be our atomic modules. Anything which occurs in library might be our atomic module or indeed single transistors might be our atomic module. The end result of elaboration in the flat hierarchy free collection of single nets connected to behaviourally described modules through defined ports. This defined ports describe which way the signal is flowing.

Is it an input to a module or an output to a module, so this is what elaboration will do; and finally we go to event driven simulation.

(Refer Slide Time: 45:07)



The Design Process | Timing and Delays
Basic HDL concepts | concurrency
Concurrent and sequential Descriptions | Simulation of hardware

Event Driven Simulation

- We maintain a time-ordered queue of signals which are waiting to acquire their assigned values.
- The time variable is advanced to the earliest entry in this queue.
- All signals waiting for acquiring their values *at this time* are updated.
- If this updating results in a *change* in the value of a signal, an **Event** is said to have occurred on this signal.

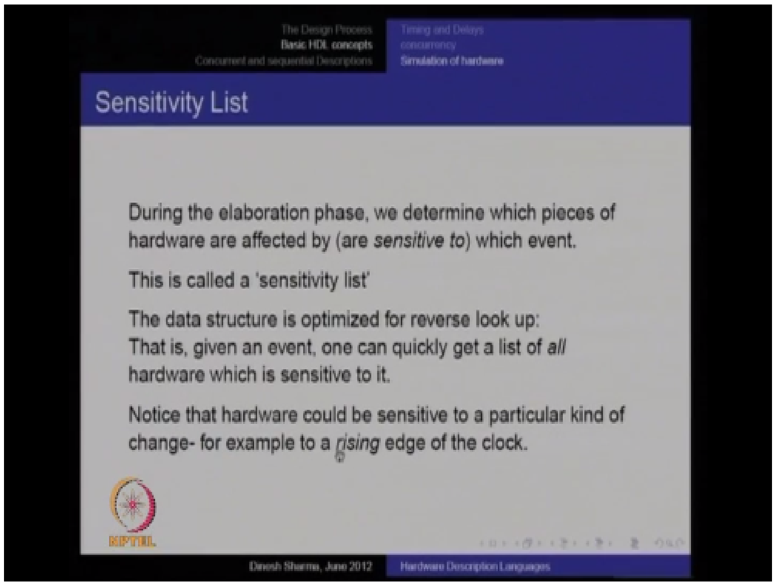
Dinesh Sharma, June 2012 | Hardware Description Languages

We maintain a time-ordered queue of signals which are waiting to acquire their assigned values. If you recall our discussion on delta times, now you realize how important that is because for the user, many events might have occurred at the same time but for the Hardware Description Language their time ordering is very important and therefore we maintain a time-ordered queue of signals which are waiting to acquire their assigned values.

Now, we go to the earliest entry in this queue and the time variable is given a value which is equal to the earliest entry in this time ordered queue. All signals which are waiting for acquiring their values at this time are then updated to their values. This is when they acquire the desired value. So, therefore all these updates or transactions were kept in this time-ordered queue and then we take up one-time incident at a time.

Look up all the transactions which are placed for this time and these transactions now fructified. That means the signals actually acquire the value which they were assigned perhaps at an earlier time. After updating these values, we notice which values have changed and if this updating results in a change in the value of a signal, then we say that an event has occurred on this signal. This is very important because only if a signal has changed that we want to re-simulate a part of the circuit.

(Refer Slide Time: 47:05)



Sensitivity List

During the elaboration phase, we determine which pieces of hardware are affected by (are *sensitive to*) which event.

This is called a 'sensitivity list'

The data structure is optimized for reverse look up: That is, given an event, one can quickly get a list of *all* hardware which is sensitive to it.

Notice that hardware could be sensitive to a particular kind of change- for example to a *rising* edge of the clock.

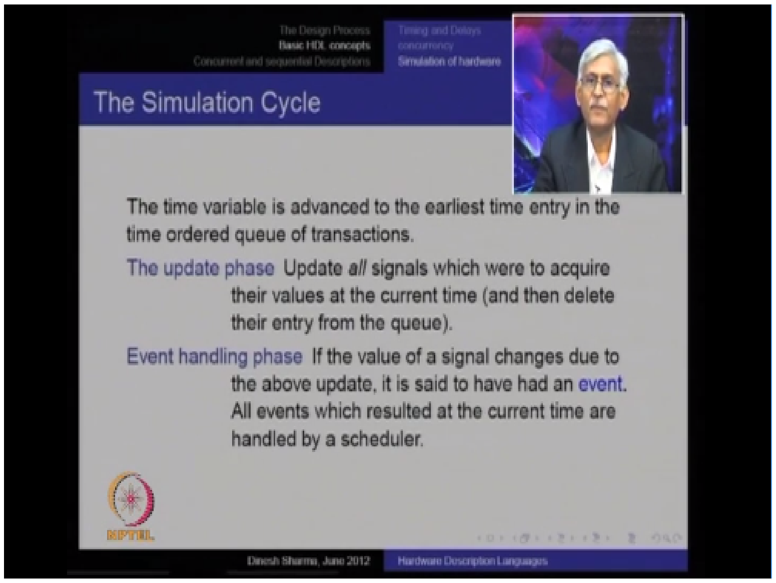
We maintain as I had said earlier a sensitivity list. During the elaboration phase, we determine which pieces of hardware are affected by or are sensitive to which event and this is called a sensitivity list. The data structure is optimized for reverse lookup. While describing we describe hardware and then the hardware is sensitive to certain inputs. But we keep the data in the opposite way, that means if you know that an event has occurred, we can go back and look up all the pieces of hardware which were sensitive to this event.

So, that means given an event, one can quickly get a list of all hardware which is sensitive to this event. Notice that hardware could be sensitive to a particular kind of change. For example, you could have a flip-flop which is sensitive to a rising edge of the clock and not a falling edge. So, therefore if the event is for a one to zero transition, the hardware doesn't bother about it, but on the other hand if the change is from 0 to 1.

Then, this hardware would like to reassign new values to its output. So, in short we have a sensitivity list, the sensitivities to particular events and then whenever an event occurs, we have an efficient data structure which then goes around looking for all pieces of hardware which are sensitive to this event which has currently occurred. When this event has occurred, then all pieces of hardware which are sensitive to this event are re-simulated.

The re-simulation will then result in further transactions. So, the simulation cycle consists of the following.

(Refer Slide Time: 49:04)



The Simulation Cycle

The time variable is advanced to the earliest time entry in the time ordered queue of transactions.

The update phase Update *all* signals which were to acquire their values at the current time (and then delete their entry from the queue).

Event handling phase If the value of a signal changes due to the above update, it is said to have had an event. All events which resulted at the current time are handled by a scheduler.

The time variable is advanced to the earliest time entry in the time-ordered queue of transactions. Then we entered the update phase in which all signals which were to acquire their values at the current time. So, we update their values and then delete them from the queue because after all we have updated their values. Then, we enter the event handling phase. So remember all updates are done before you enter the event handling phase.

When all the updating is completed, then we have a list of various events which have occurred, and it is only when we have prepared this list that we enter the event handling phase, and now if the value of the signal changes due to the above update it is said to have had an event and all events which resulted at the current time are handled by a scheduler. This scheduler will handle all the events in a particular way.

For this lecture, which shall stop at this particular point and we shall resume in the next lecture and see an actual example of a very simple circuit how it goes through the process that I have described somewhat abstractly here. It is only when we look at an actual circuit undergoing those actions that it will become quite clear to you how the transactions are handled in a time-dependent way.

How the time-ordered queue is maintained, how events occurred due to updating of signals and how are these events eventually had did. So, we shall do that in the next lecture. We stop here at this point in this one.