

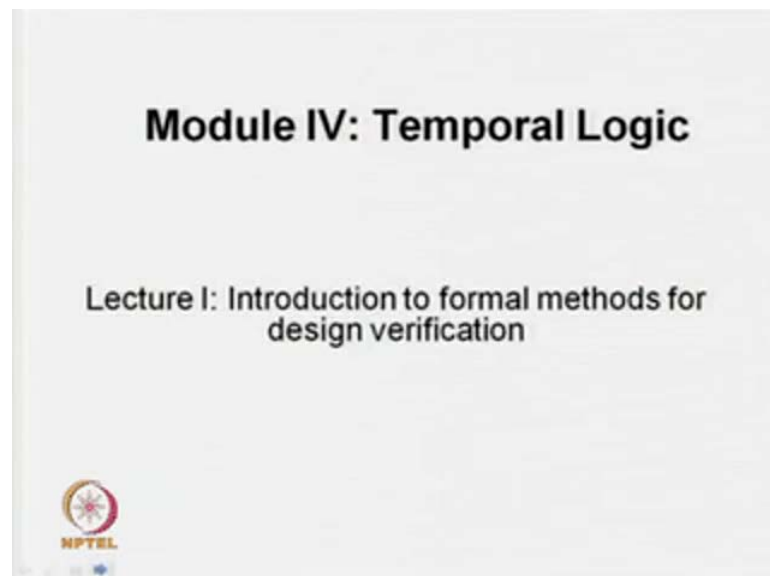
Design Verification and Test of Digital VLSI Designs
Dr. Santosh Biswas
Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

Module - 4
Temporal Logic
Lecture - 1
Introduction to Formal Methods for Design Verification

This course is about design verification and test of digital VLSI design. Basically, we are having 3 components in this particular course. 1 is design issues of your digital system, 2 nd one is verification and 3 rd part is testing of our design. In our last part, in the 1 st part already we have seen what are the design issues involve when we are going to design a digital system. In 2 nd part it is the verification and 3 rd part is testing of our product.

So, in this verification part what basically we are going to do; what need to be verified and why we go for verification. Without verification, whether we can design our circuit, we can develop our circuit or not we are going to look all those issues.

(Refer Slide Time: 01:12)



So, in this particular part we are going to look for verification issues and one of the components is our using logic of verification. But before going for the particular logic basically we are going to cover about temporal logic but before going for temporal logic we want to see what are the methods used for your verification and why it is need that. It

is basically some sort of introduction I am going to give and I will say why it is needed. Why you should go for verification.

(Refer Slide Time: 01:41)



So, if you look into it. In our design cycle basically we have going to get those following step; specification, design, implementation, testing, installation\marketing and maintenance. So, in our 1st part basically we are talking about this particular design issues; and when we go for design we need to know the specification of the system. That means as a designer; we are going to design a device and thought at we are having some intention that means my design should satisfy some specification.

First a fall we have to keep this particular specification just give a brief idea. What we are going to do in specification, why found specification will come just will give an example.


(Refer Slide Time: 03:22)

Design Cycle

Specification

Suppose we have to design the controller of a washing machine. There are certain aspects of washing clothes that the system has to take care of, like:

- The drier is activated after the wash not before ~~it~~
- Water is poured in before the detergent and it is drained before activating the drier.
- Cold water is to be used in soft wash where hot water in heavy wash, etc.



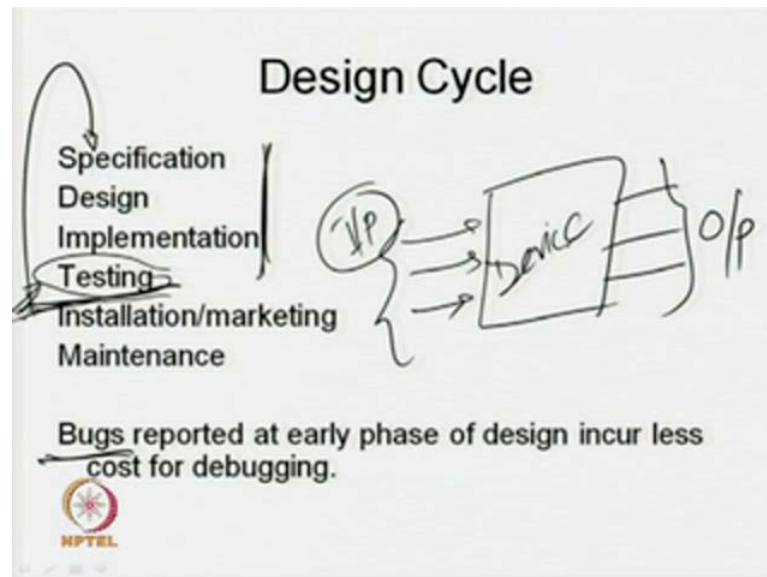
So, we are going to design a control of for our washing machine. What we can say about washing machine. We said that our design or ours control of our washing machine should satisfy some property. And, this property or specification can be in mention as that drier is activated after the wash not before.

So, because 1 st we are going to wash the cloth after that will activate the drier to dry our cloth. Secondly, we can give one specification something like that water is poured in before the detergent and it is drained before activating the drier. So, these are basically our requirement or we can say that these are the property that my controller should satisfy. And, another specification can say that cold water is to be use for your soft wash because we if we having the soft wash or sometimes we need to use hot water for our heavy wash.

So, accordingly it appropriate term we have to activate our heaters. So, these are the several components that we have an according to our requirement, we have to activate those particular component. Now, this is about your specification design. And, one design is completed then will go for the next page is our implementation. We are going to implement our controller. After controller is implemented than we need to test it whether we are getting a correct result or not but use here we are having this particular test thing. And, when we are doing it this test thing you have done after implementation that means after completion of this particular 3 phases.

So, that means we are going to test about the correctness of our product. Then, it will come for the installation and then maintenance. First, we have to install or may be market the product. And, once it goes to the market then what will happen we need a we need to have the maintenance space. We have to maintain it.

(Refer Slide Time: 07:00)



So, here we have one issue; that whenever we are going to design a circuit or going to design a system, it may not be correct in the very first. So, it says the bugs reported the early phase of design less cost for debugging. So, when we are coming to this particular testing phase we are testing it. And, how this will be test we know the input pattern. What are the inputs that we have to give? Along with that we know what the desirable are. So, basically we consider my design I can say that I am designing a device or this is my device. I can treat these things as a black box.

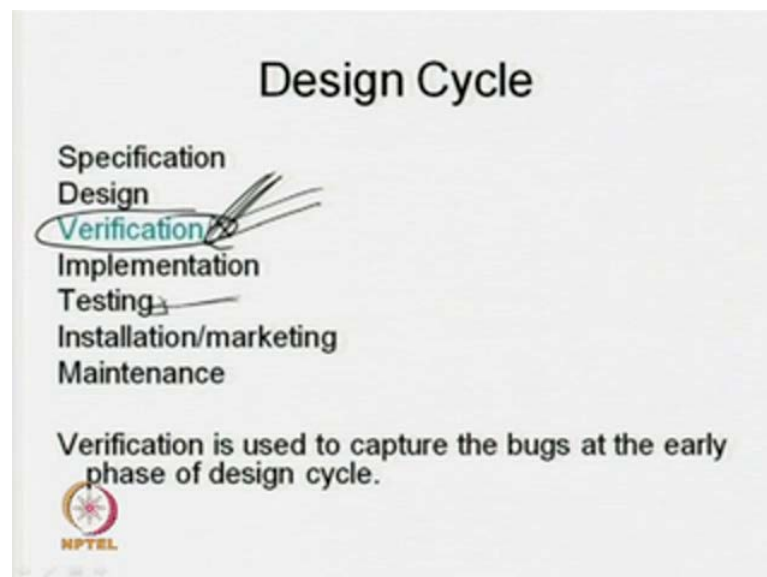
Now, there are several inputs are available for this particular device. And, according to this particular input behavior way we are going to get some output. Now, we know the output behavior of this device according to this particular input combination. What are the input combinations we have? So, during testing we are going to test for those particular input bear pattern. And, you see whether we are getting a desired output or not.

So, in this particular case during testing we are looking for both testing actually 2 types of testing. 1 is your functional testing, 2 nd one you are manufacturing default because during fabrication, during manufacturing it some fault may come into fixer due to the

faulty manufacturing. So, in testing we are going to capture those particular issues also. And, if devices really your faulty than we have to discuss it. But if it is having some functional around and what will happen? We have to go back to our specification. We have to revisit our specification, we have to look for our design issues then we are going to redesign it or will revisit our design.

Once, we fix the bug basically it is a going back due to some bugs that has been reported. So, due to those bugs we are going to fix it and eventually again we follow this particular part; we may have to do several time. So, each says that since after implementation we are testing it; and we can capture the bugs at that particular point only after the test. So, we are going a long way to get our test. So, it says that if we can capture the device fault your design fault in this particular early phase then what will happen. It may reduce our cost because again another effect we are having time to market also. In some specific time we have to release our product to the market. So, that so you are thinking whether this particular test of think can be done prior to this coming to the implementation or not. So, for then in our design cycle we are in covering one more phase over here; which is we are talking about that verification.

(Refer Slide Time: 08:40)

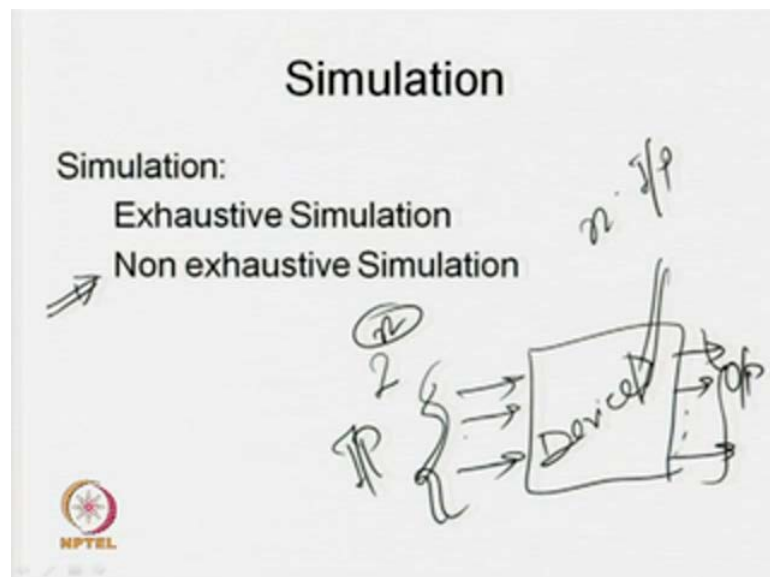


And, in this particular what of our course we are going to loop for those particular verification issues after always that we are going to verify. What we are going to verify and up to some why verification is needed. And, secondly there are several way of doing

the verification but here we are going to loop for formal verification on way. So, here what we can sudden verification is used to capture the bugs at the early phase of design; used is that we have placed the component verification just after design. By looking into our specification we have cover up with a design. And, after the design we know that for a specific purpose we are going to design a device. That designs those devices which satisfy some of the property or some of the specification. In this verification phase we are going to check for this particular correctness of those properties.

And, we can say that this is some sort of your property verification or we can say this is a functional verification. We are going to verify the function functional property of our defects. Once you satisfy that whatever design we are doing; it is going to satisfy our requirement then we will go for implementation; and after implementation we will go for testing. So, when we are coming to this particular testing point then basically it is remaining left with your manufacturing defect all way. So, we are trying to most after design issues or design bugs in this particular verification phase. Now, we are talking about here that verification. Now, how author define ways that we can do it, whether it was there or not in the early phase.

(Refer Slide Time: 09:03)

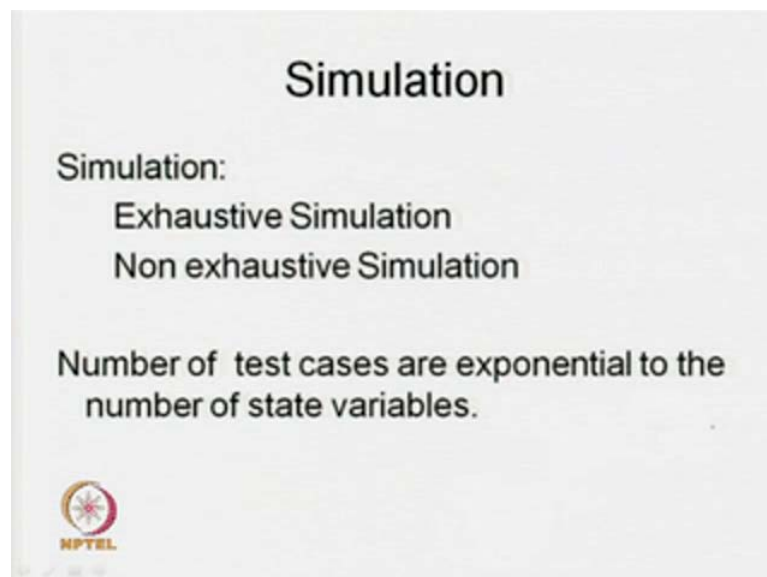


So, in most of the cases we are having that particular notion of having simulation. We do the simulation of our entered device, and check whether our design is correct or not. So, basically simulation happen.

Again, we think that this is my device I have designed it. And, we know that these devices are having several inputs. And, we not depending on the input pattern; we should get result output. So, we are going check those particular design output with respect to this input pattern. So, this is a basic simulations so we are going to have a simulation device. And, we are going to check it and if you find it is correct then we go for fabrication, but quite is it possible to go for simulation for all type of devices. If you look into that issue you will find at now the system or devices began more and more complex. And, we are having complex design part of devices because now due to the advancement of our semi conductor technology. You can put more and more component in a silicon way so we can go for complex design.

And, depending on our design the number of input is also increasing. So, the number of possible input cases may be you can say it is your two the power n if we are having n number of input signals. So, it is exponentially growing. So, for that it is not possible to similar for all possible combination. So, people are saying that we are going for a selecting simulation.

(Refer Slide Time: 11:04)



So, that is why it is talking about this non exhaustive simulation. In case of exhaustive simulation we try to test it for all possible combination. But due to the large number of input combination because that input pattern is more exponential in the number of inputs signal. It is not possible to tell the entire circuit for all the possible combinations. So, for

that we go for selective simulation and we said this is non-exhaustive simulation. So, for we said it is a number exponential already I have mention. So, in that particular case we are going to test it for some fixed number of input combination.

(Refer Slide Time: 11:16)

Non Exhaustive Simulation

Instead of using all possible combinations, simulation is done for some selected input combinations.

To find the appropriate subset is a complex problem.

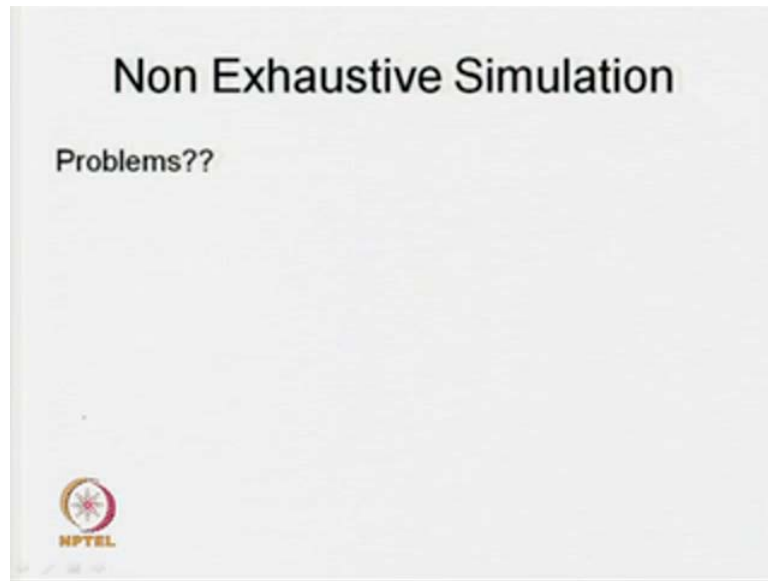
Test case generation

We may not cover all possible error cases.

NPTEL

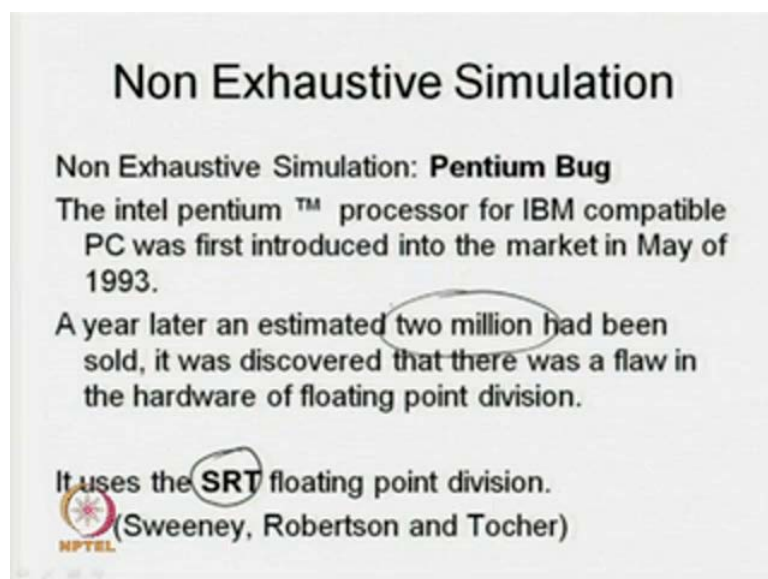
So, here instead of going for your all possible combination; we will go for some selected combination. So, just summarizing it to find the appropriate subset it is again a complex problem. So, we are having another research problem over here. How to find out a appropriate subset of those particular input pattern. This is come under this particular test case generation. Since, this is another domain; we are not going to look into this is over here, but just I am mentioning that this is a problem or we can say that ATPG automatic phase pattern generation. With the help of those particular techniques; we can find out what is the appropriate subset to test our design. So, since we have going for non exhaustive simulation. So, what happens we may not cover all possible error cases? It is quite obvious because it may happen that some of the combination may not be captured by the phase cases that we have given.

(Refer Slide Time: 12:17)



Now, from exhaustive simulation we are coming to non exhaustive simulation. So, in this case use a what may be the problems that we may face. You just think we are not considering all possible input cases and we are giving subset of it. Whether that particular subset is capture in all possible error combination. It is already I have mentioned it is another problem, another complex problem it is difficult to find it out. But still we are having some heuristic we does particular heuristic we try to get reasonably good or some set of test cases.

(Refer Slide Time: 13:19)



So, people are working with it. It is going fine but what problems are coming over here. Why we need to go from your non exhaustive simulation to some other domain. That, in this class I am going to cover about your formal verification. Why we need to go for formal verification. Necessarily we have some problems in non-exhaustive simulation and it is quite obvious. You not considering all possible cases.

Now, you just see now we are going to see what the problem is. That is a Pentium Bug I think you might have heard this particular term. What is Pentium bug? And, this is a major challenge that Intel has received in early 90's. So, what happen? Intel has released here processor Pentium series of their product in 1993. So, you know also it Intel is a big house is your processor design. And, if you look into their history line or time line you will find out they have started with Intel 386 as the basic processor. Then they have gone with 486, Pentium, Pentium Pro and up to Pentium 4 they have gone. And, you know that this is a apart compatibility.

Apert Compatibility is what about we can do in 386 something can be done in 486 but with so this is death timeline. So, after 486 they have release the product Pentium series. Again, it is apart comfortable what about we can do in 486 something can be done in you Pentium or be Pentium series. Now, say it is apart comfortable whatever we are doing in 486 something can be done in you Pentium but where is from the bug is coming.

You just select a year later an estimated 2 million had been sold. It was discovered that there was a flaw in the hardware of floating point division. You just see what is happening basically say they have release the Pentium it is apart comfortable. What about we can do in 486; something can be done in Pentium along with that we have some are the additional features.

Now, Intel design team has design it properly; and they have tested it; and they have release the product. But after 1 year an error has been reported for division only floating point division which was there in 486 also. So, while from this particular error has occurred. Now, you see that in your time line or during the design they have sense there division algorithm. So, earlier they have algorithm but in Pentium series they are look for a faster algorithm and they have use this particular SRT floating point division algorithm.

You see basically SRT is coming from 3 scientist names Sweeney, Robertson and Tocher. They have divided this particular division algorithm. Since, it is a faster algorithm so Intel has decided to use this particular division algorithm in the new Pentium processor. So, what is difference of basically what happens in this certain division? We have to use a look comfortable to find out the some input pattern looking into some input combination. So, they have implemented this look up above in their ram and due to an error in this particular one end error in one entry in this particular look table.

For some input combination it is giving error. So, this is the problem basically, this is the Pentium bug. So, we are having a look up table; we had some entries in the look up table but there is an error in one entry. And, whenever we are going to use that particular entry it is giving as a wrong result. But see Intel people have tested it. How they have tested it. They had gone for non exhaustive simulation. So, with non exhaustive simulation already I have mentioned that all it is not possible to capture all error because we are very much selective about our input test pattern.

So, some design flow or . So, this is the case where this particular designer or we sleep to. So, in after this particular Pentium bug so it is a Intel is a big business house. So, they can sustain such type of loss because what will happen since about 2 million chips they have previous . What they did? They call back all those particular chips and replace it by a new one and forded they have incur loss of 7500 million dollars. Since, Intel is a big house they can cope up with this particular error.

So, from this particular error all EDA components has lined in and they found at that non exhaustive simulation is not the solution. We were coming from exhaustive simulation to non exhaustive simulation but that non exhaustive simulation is not the right one to do it is not going to give an error free design. At that point people are thinking now what to do.

(Refer Slide Time: 18:06)



So, in that case this notion of formal verification is coming into friction. Is it not like that we are formal verification after Pentium bug it was here. We shall walk with this particular matter how to formally capture the design, how to formally specify of property, and how to check that the specification or the property or school in this particular design. So, this is basically formal trying to cap side it was here resources are working over here but industry people are slightly reluctant to use it. But after Pentium bug all people has think about it. Now, we should look for the alternative.

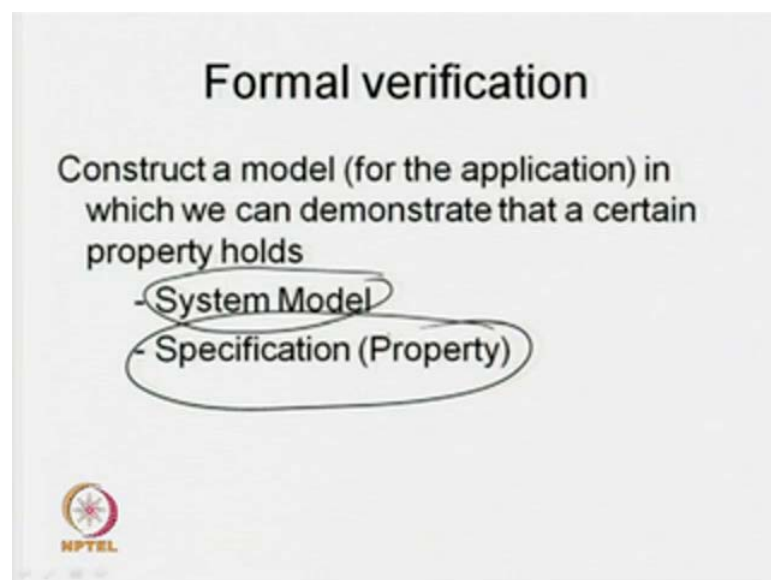
So, people are going for this particular formal verification and what are the issues that we can exist over there. Already I have said that it is the design complexities increasing day by day because in a small silicon area now we can put more and more components. And, indirectly it means that we can press more and more devices in same or small silicon area. That means we can place a complex circuit or complex device in a small silicon area.

That is why the complexity of the design is increasing. So, what we do in formal verification. Basically, we start from our abstract model. So, we know our design, we know our objective but in step going to the direct product or the direct design. What happens? We try to abstract out the relevant information and we are coming up with an abstract model. And, generally we deal with this particular abstract model. So, how to come out with an abstract model or that we need formal matter, we use formal methods,

formal mechanism. And, after we apply some verification techniques and that is why analysis is this is the formal verification.

So, Model helps us to build more complex system. Already I have mention that whatever is relevant, whatever is required we try to abstract it out we have a design out of it some sort of modular design can we think about it. And, for every case we can have a small model. We try to capture the design behaviors and we know what the properties are or what are the specification which is satisfies we will going to loop for the satisfaction of those particular property. So, again on the other a model is easier to understand than a whole system because we are coming down to a smaller peice of a whole system.

(Refer Slide Time: 20:27)



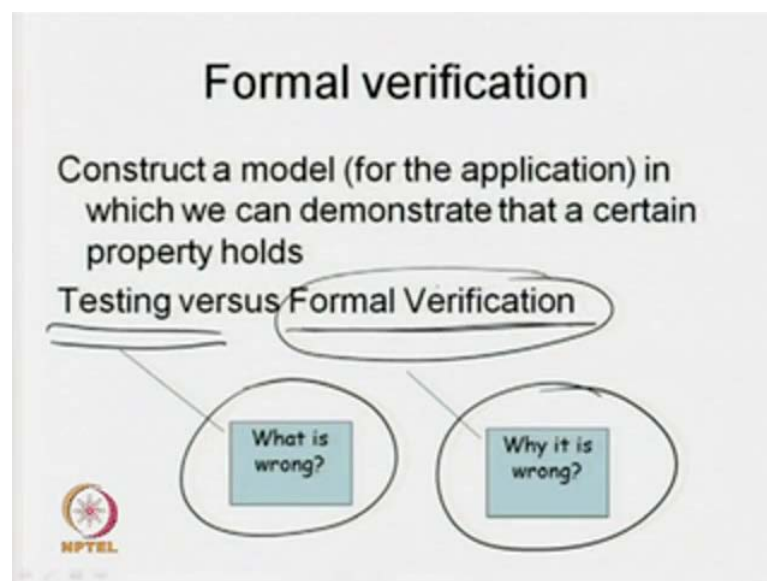
So, in this particular case we are going to construct a model in which we can demonstrate a certain property holds. So, we need 2 things. 1 is system model and 2 nd one is a specification of the property already I said that I am going to look for a model somehow we are going to keep our design and we set it this is our model. Another one we are have the specification or the property. We know that I am going to design a new device and that device should satisfy some of the property. Already one example I have given that we are going to design a controller of your washing machine and we said that it leads to satisfy some of the property.

Secondly, if you look into the formal verification of formal matters. It is not like that we can apply only in our designing of VLSI circuit or a hardware circuit. It can be

applicable in any design. One simple example I can give, say you are walking in a network in computer network. And, you are designing some new computer protocols that transfer protocol transport protocol. Now, in the most of the protocol what will happen one you send package or message forms sources to destination. What happens generally? That source expected it should get back the acknowledgment then only the sender knowing the message has been delivered properly in to the destination.

So, what is my requirement over here? I can say that when sender sends a message eventually it should get back the acknowledgement. So, this is we can say this is the property or this is the specification of the protocol that I am going to revision. So, I am telling you what is the specification but somehow formally we can specify or we have to capture this particular property. Similarly, that about your system model that we have that also formally we have to device. We have to design and eventually we are going to check whether this property is satisfied by this particular model or not. This is the formalism or the formal mechanism and we say this is the formal verification.

(Refer Slide Time: 22:45)

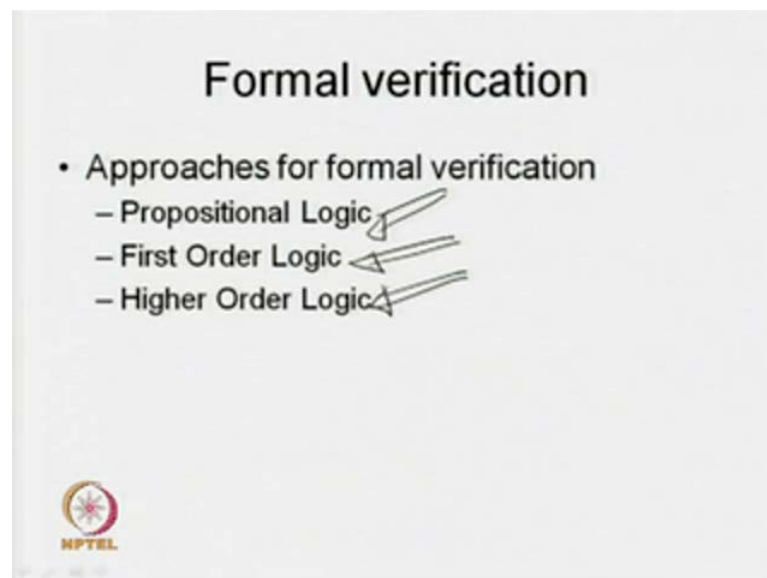


So, again now already we have the testing. And, now we are coming in the formal verification now. What is the difference between these two things? So, what is the testing and what is the verification. Basically, in testing we talk about what is wrong. So, this is basically testing. So, we have that is device, we have the product, we tested it and we said that this is the testing result. It says what are wrong in that particular case but in case

of we are verification or when we are going to do it; formally then we can just try to address this particular issue why it is wrong.

So, just it after your design phase we will do the verification. And, in verification will keep the feedback why something is going wrong, why it is wrong. So, when design team gets this particular feedback then what taken do? They can revisit their design, they can fix up the bug, they can fix up the error and they can rectified their design and come up with new rectified design. So, with this new rectified design again we apply this particular formal verification. And, when we are satisfied that now it is satisfying all my properties all my requirement then we proceed further then will go to the next phase. Basically, next phase is implementation of fabrication.

(Refer Slide Time: 23:49)



So, how we are going to do this formal verification. So, there are several approaches to go for verification or defining the system formally and specifying our properties and looking for a correctness of those properties in our design. Since, there are several mechanisms available processor there but in our course in this particular course we are going to look for the logical formalism. We will see how the logic is used for our formal verification. We may not go for the ordered issues.

So, if you look into this particular issue; when you come for logic. We know that we are having 3 different type of logic. One is propositional logic which is basic one. Then we look for a fast order logic and higher order logic these are basically predicate logic. We

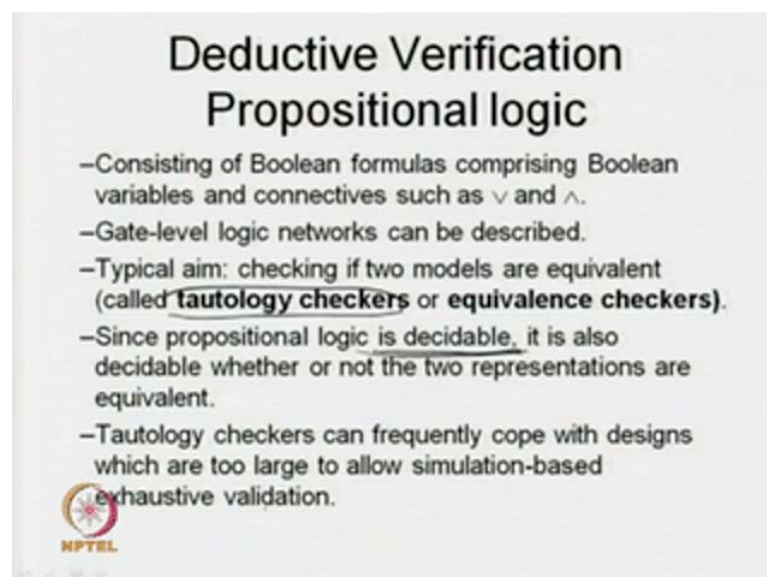
define predicates and we try to reason about this particular predicate. So, in propositional logic you see that these are basically or we are dealing with the declarative statements. And, we are going to check with a something can be derived from some other given declarative statements or not.

So, like that we go for first order logic, these are predicate logic than we go for the high order logic. So, in this way we can go up for higher order logic and will see. Now, what can be done with propositional logic or when we need to go for the higher logic? And, what are the difficulties we are going to phase if you go for the higher order logic.

Since, if you look in to this particular order that fast them showing the propositional, then first order, then higher order. So, that means we are going for basic logic do more and more complex logic. When we are going for a complex logic basically what we can say, simply I can said that you are having more expressive power. Basically, what we cannot express in your propositional logic something can be express in our first order logic. Secondly, again if something cannot be express in first order logic that thing can be expressed in the higher order logic.


So, the expressive power is more. Since, we are having more expressive power so logic becomes more complex; and reasoning on this logic again become more complex. So, if we need going for more expressive power then we have to go for complex logic. And, that design that logically in finds will be a complex one.

(Refer Slide Time: 26:20)



Deductive Verification
Propositional logic

- Consisting of Boolean formulas comprising Boolean variables and connectives such as \vee and \wedge .
- Gate-level logic networks can be described.
- Typical aim: checking if two models are equivalent (called **tautology checkers** or **equivalence checkers**).
- Since propositional logic is decidable, it is also decidable whether or not the two representations are equivalent.
- Tautology checkers can frequently cope with designs which are too large to allow simulation-based exhaustive validation.

 NPTEL

So, first one is propositional logic. What we can do the simply this going to give an idea that what can be capture in propositional logic. And, what we can use just these are some introductory things that I am going to tell you and after it we will go for how we are going to do the verification.

So, in propositional logic we are going for a deductive verification. What we have over here consisting of Boolean formula comprising Boolean variables and connectives and or etcetera. So, we are going to work with Boolean formulas. And, Boolean formulas will be constructed with the help of Boolean variables and with connectives. All we know that these are WFF in propositional logic which is basically well formed formulas. So, we have to construct well formed formula and we are going to work with this particular WFF.

Another note, what about if you are going to look for a digital device or digital circuit. It can be always with the help of Boolean formulas. So, it is a set of Boolean formulas which is going to give the behaviors of our system. So, this Boolean formula can be again treated as our statements in our propositional logic. So, that is why I am saying the gate level logic network can be described with the help of this Boolean formula of propositional logic. And, in typical aim in case of your deductive verification what basically we check with a 2 models are equivalent or not.

We call these are tautology checker, because in our design phase we are having an abstraction. First we come up with a model then will make it more refine we will go for another level. Now, after that we have to check whether we are giving together equivalent translation or not. So, we can look for such type of equivalent checkers. So, these are basically sometime we can say tautology checkers is also here or equivalence checker.

Again, we said since propositional logic is decidable. Now, we have to see what a mathematical theory is over it. Since, it is decidable. So, we can say that it is decidable whether two representations are equivalent or not. Again, tautology checkers can frequently cope up with the designs which are too large to allow simulation based exhaustive simulation. So, we have problem with the exhaustive simulation. So, now tautology checkers can cope up with those particular problems that we faced in a exhaustive simulation.

So, in a way saying that what we can do so basically digital system of VLSI system can be treated as a combination of your Boolean formulas. And, this Boolean formula can be map to a proportional logic step. Hence we can use the formalism propositional logic to reason about those particular formulas. So, this is one here but we know the expression power is less. So, whatever we can do that is also restricted.

(Refer Slide Time: 29:06)

The slide is titled "First order logic (FOL)" with handwritten symbols \exists and \forall to the right. It contains three bullet points: "FOL includes quantification, using \exists and \forall .", "Some automation for verifying FOL models is feasible.", and "However, since FOL is undecidable in general, there may be cases of doubt." The NPTEL logo is visible in the bottom left corner.

The next level we are going to talk about first order logic or first order predicate logic. We know that this is basically what happens? We are going to use 2 quantifier basically these are there exist and are all with this particular quantifier we caps we try to capitalize set of statements. With a for all x something is to or there exist in x something is to 1. So, with the help of this thing that means it is having slightly more expressive power. Secondly, we are having 1 issue since FOL is undecidable we have said in general. So, we are may be case of doubt.

So, when you use first order logic for you are designing about system. After doing it, we may some doubt in some cases because it is undecidable order manual intervention is required. And, in this case some automation for verifying FOL models is feasible always not true automation is not that possible for all cases sometime we get automation also. That means your manual in intervention is always required when you go for first order logic or higher order logic.

(Refer Slide Time: 30:10).

Higher order logic (HOL)

- Higher Order Logic allows functions to be manipulated like other objects.
- For higher order logic, proofs can hardly ever be automated and typically must be done manually with some proof-support.
- Interactive theorem provers require a human user to give hints to the system.

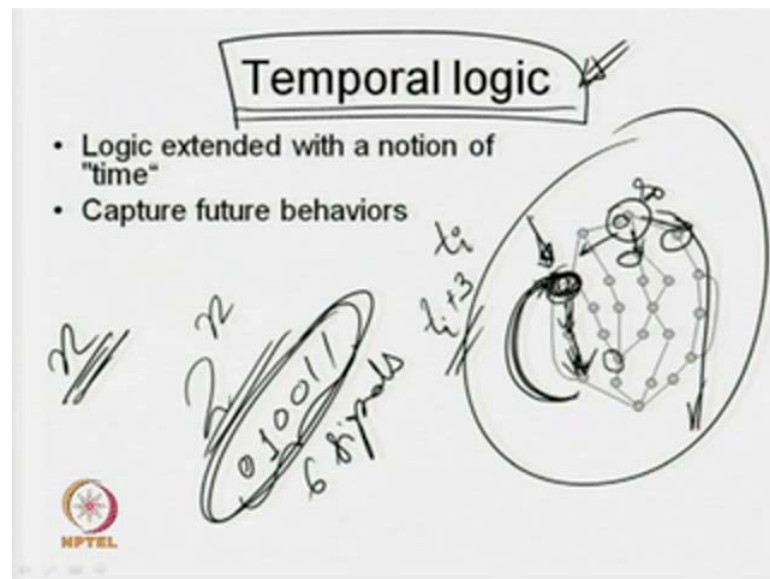


So, next is higher order logic. Again I said that this is more expressive and so that logically becoming more complex. So, in that particular case what advantage we are getting. What is expressive that is we are getting. It says that it allows function to be manipulated like other objects. So, that function can be treated as an object that thing cannot be cap side in your first order logic but higher order logic we can cap side. That means now it becomes more expressive. So, expressive power is more. So, that means now we can express many more things with high order logic but since it is complex. Now, listening with high order logic is also complex.

So, for higher order logic proofs can hardly ever be automated and typically must be done manually with some proof support. So, this is the finding that we have. So, it is hardly can be automated because now it is a complex automation is not possible but partial automation may be possible. So, we can say that interactive theorem proves require a human user to give hints to the system. Now, we are having interacting theorem provides interactive we are going to check from one sentence to the another sentence or one statement to the another statement, for some step it may go in an automated way.

That means we are having some formal way of doing in but after it human intention is required that means we have to give some input. So, that my proofs directs to the right direction. So, that is why you are saying that we have to human user must give some hints. That means we have to give some inputs to go into the proper direction, when we look for that design with this particular higher order logic.

So, this is basically that, you know about these things propositional logic, you know first order logic, you know 2nd order logic, and 2nd order onwards we said is a higher order logic just simply I am telling what we can capture with this particular logic. And, in formal verification when we are going to apply those things. What we are going to use in this particular lecture? I am going to tell about it and what will need for that. We need some more things but that will be based on these 3 logics only. That is why I have mentioned about these 3 logics.



So, here we need one more information, this is about time. So, you know that we are working with our digital signals and we are going to have a controller to walk with our digital signals. And, we know that this system is going to behave defined to a at defined point of time that means timing will come into picture. Say you know example, you think about that washing machine say you have start a machine initially it has to give pour the water after that we should give you the detergent then it will go into the wash mode.

So, after sometimes around say you can say that time like 10 minutes or 15 minutes depending on the wash style. What happens we have to drain out the water? So, you after sometimes we have to drain out the water; again put trace water into this particular washing machine. Just see in what have lead that timing is coming into picture. So, somehow we have to capture these particular timing issues. We need some formalism; we need some mechanism to specify those particular timing issues. So, for that, we cannot do with these particular or basic logics like propositional logic or predicate logic

for that we have to go for some other logic. And, this is basically temporal logic that we are going to talk about it.

So, this is temporal logic. And, the method verification method that we are going to discuss in this lecture based on this particular temporal logic. So, what is this temporal logic? Already I have mentioned that I can say that it extend the notion to time actually. So, we can capture time and we can capture the user behavior. Now, you see that what is a system? Already I have talked that somehow I have to capture the design issues we are come up with a model. In most of the cases we are going to get a finite model on there.

If you slightly removed that timing issues, because in case of timing issues it may be turn up to be reactive system which will repeat the something for several times. So, in general we can say that we are going too worked with our signals. We are having a fixed number of system and most of the cases we are going to get final step machine. So, that steps phase is always final. And, what will be the step phase? If we know the I am working with n variables or n control signals the we do not know the number of different combination we are going to get is 2 to the power n .

That means, what big may be your n , always we are going to get a final steps. And, every structure going to talk about some configuration. That signals values either 0 or 1. We are going to talk about digital system. So, that is why I can think about such type of your model. So, we can said that initially my system is here say this is say that this is the state as 0. Then, depending on my input behavior or depending on my input signal or the system where we are it can take go in 3 different way.

So, again said is depending on some condition either I am coming to this particular step or we are going to this particular step. If I am coming over here that means my execution in follow this particular point. So, this is the way we can capture our design. And, here you see that this step I am in a particular time; now here the time is different. Again, you see that here I am going back to this particular step, say this is the execution phase and going back to this.

Now, say this particular step basically, this is decide or this is going to give the binary encoding of my this particular n input signal because this step I am going to define with this particular signals on their other this signal is high or signal is low. Basically, if I am using signal either signal may be high or low. So, this may be one particular pattern sat if

I am going to have 6 signals. So, if I am going to this is the step, where the pattern of my signals is this one. Now, say I am going to follow this particular path and when I am reaching this particular step. Depending on my situation I am going back to this particular configuration.

Here again this my input signal pattern in same. That is why coming to same behavior but when you look into timing issue. Then what will happen? When you fast encounter this particular is step at that time that say time is said t_i . After this 3 step, say if consider that every state is have 1 unit. So, 1, 2, 3 next time when I have gone back to this particular step that my time is t_i plus 3. Now, this time is important. And, in our case what will happen or issue is to get this particular timing notion and you can do this thing with the help of this temporal logic.


So, that is why I going to just keep you brief idea about temporal logic. And, this is about this module is basically talk about a temporal logic. And, we are going to have a series of capsa. And, I am going to give the idea about this particular temporal logic.

(Refer Slide Time: 37:40)

Temporal logic

- **Branching vs. linear time:**
 - **Linear time**
Models physical time
At each time instant, only one of the future behaviors is considered.
 - **Branching time** (at each time instant, all possible future behaviors are considered).
 - Models different computational sequences of a system.
 - Nondeterministic selection of the path taken.

The slide includes three diagrams: a straight arrow representing linear time, a branching arrow representing branching time, and a state transition graph with multiple paths.



So, what basically we are having. So, I have said that we are having 2 timing behavior we are going to capture. We are going to use temporal logic for that we are having 2 way of doing it. 1 we are going to talk about the branching nature of time and 2 nd one is a leaner nature of time.

So, in case of your branching nature, then what happens? Basically, you say that we just thing that time progressing one direction only but in case of your branching. So, this is linear nature sorry I talked. So, in linear case the time is progress in one direction only but in case of your branching time can branch out in several direction. Now, just come back to this particular model. If I am going to look for from is particular step I can go into 3 different direction. So, depending on my input pattern or depending on my system configuration it can follow one of these 3 particular parts.

So, that means I can say that the nature of timing having a branching nature at that particular point. So, it branch out in 3 different steps but if you concentrate on a particular part over here; and thus say that we are interested for this particular execution part than we may not look for other issues. So, in that particular case we say that this is the linear time and we are going to capture it by this particular linear notion. So, this is basically about timing issues. 1 is you are linear and your branching. And, in case of branching when we are going to talk going to reason about this particular system. Then, we think about the non deterministic issues of this particular branch timing. We can think any one of this 3 possible continuous combination.

(Refer Slide Time: 39:24)

The slide is titled "Temporal logic" and contains the following content:

- Discrete vs. continuous time
 - Discrete time
Used by most temporal logics, mostly using natural numbers to model time.
 - Continuous time
Using real numbers

On the right side of the slide, there are two diagrams:

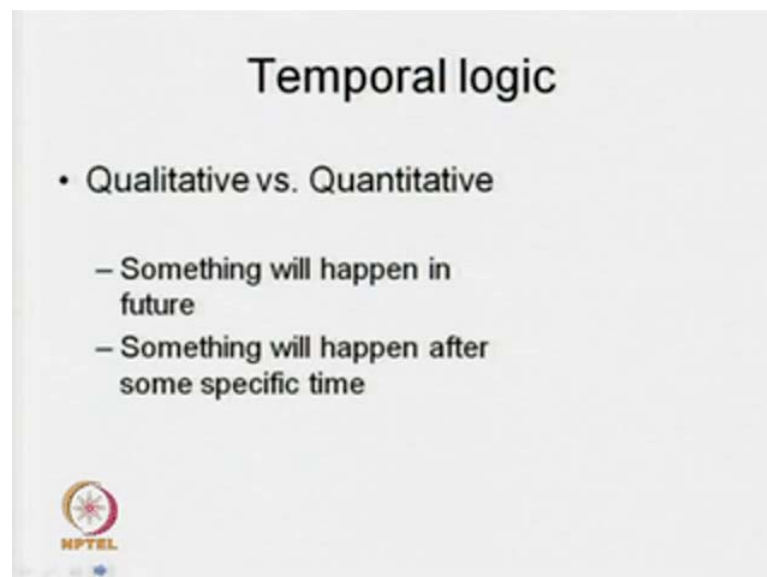
- The top diagram is labeled "N" and shows a horizontal axis with discrete points marked 1, 2, 3, 4, 5, followed by an ellipsis, representing natural numbers.
- The bottom diagram is labeled "R" and shows a horizontal axis with a continuous arrow pointing to the right, representing real numbers.

The NPTEL logo is visible in the bottom left corner of the slide.

Again we are having another issues which is called time. How we are going to capture the time. 1 is discrete in nature and 2 nd one is continuous time. So, basically you want to talk about or digital system, we basically walked in a discrete domain. So, time can be

captured in a discrete wise and we can use the natural number system to give our time. So, I can said that the tic 1, 2,3,4,5 like that in this discrete way we can define our time and we can say that this is discrete in nature. And, in case of continuous timing we are going to walk with the real number system, where every timing in possible and you may try to design about every timing in sense. Now, by thus I am saying that either you can use the discrete time or you can use the continuous time. You know about the properties of real numbers and you know about the integer or natural numbers. From here it say but you can visualize that designing about the continuous time will be a the complex one. Yes, indeed It is real a complex issue but designing about discrete systems slightly easier. So, but some system we need to design with real number also. So, continuous time remaining also.

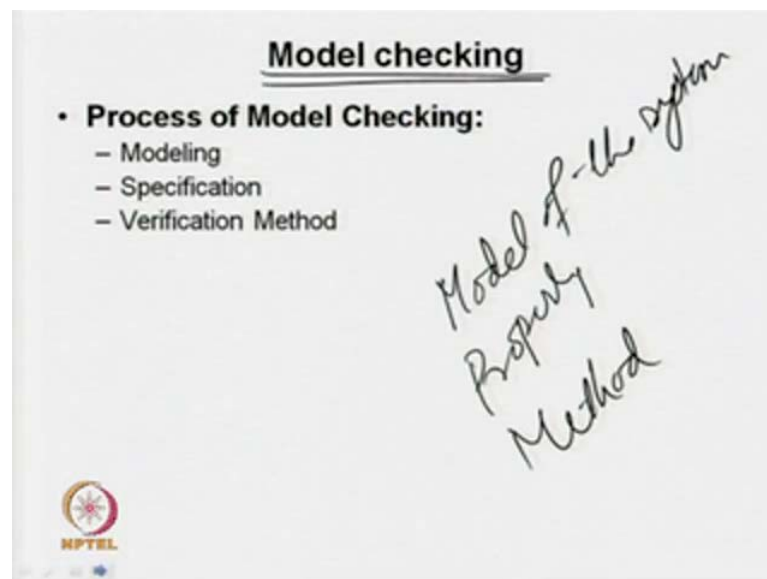
(Refer Slide Time: 40:36)



Another issue we have going to capture of timing; another issue we have is about qualitative designing, quantitative designing. Capture in the time a qualitative way and quantitative way. So, in case of qualitative way we just talk about the behavior of the time. We say that whether something is happen in now or something is going to happen in future. Just say that I am talking about the particular issue about that NATO protocol when send a message; eventually it should get back the acknowledgement. I am talking about eventually it should get back the acknowledgement. That means in future sender is a accept expecting the acknowledgement.

This is some sort of qualitative timing behavior. So, I am saying that, I send a message now in future I should get it. But in case quantitative designing we specified the quantum of time also. In quantitative reasoning what we say that now if we send a message now whether after 5 minute of time senders will get back the acknowledgement. So, this is say quantitative, we have quantified the future behaviors said that after 5 minutes of time whether senders get the acknowledgment or not. So, these are the issues that we have to look in to a chain temporal logic. We are going to look for those issues.

(Refer Slide Time: 42:00)

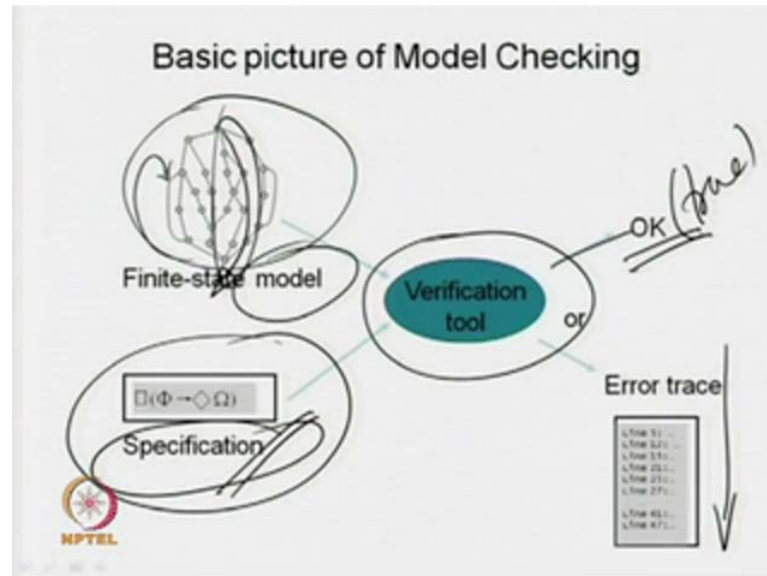


Now, we are going to look for already have mention that in this particular lecture we are going to look for a particular mechanism, formal mechanism to specify our system. And, we are going to look for this particular model checking process. Now, just I am going to talk about this particular model checking. So, view to what is the required thing in our model checking. So, we are having the basically 3 components 1 is your modeling, 1 is your specification and 3rd one is your verification method. That means, we need some formalism to give the model of the system.

So, we kept giving the model of the system then we have to somehow specified property. And, we need a method to say whether these properties are too heavy in this system or not. So, basically in model checking we are having these 3 components. Since, we are going to look for the of the property or specification, it is also termed as a property

verification. We are going to prepare properties about systems. So, it is properties verification.

(Refer Slide time: 43:24)



So, basically we are having these 3 components in our model checking. Now, we just see that what we are going to do this model checking. So, what are the basic components of model checking? So, we are having verification tools; so this is our model checking tool. And, we are having 2 different inputs to this particular model checking tool. 1 is your model and 2nd one is your specification. And, I am talking about this model as your finite states because already I mention that generally we are going to get a finite number of states about design. Because it depends on the number of variables that we have. If we are having n number of variables the total possible combination is 2^n and we are going to get 2^n different states.

Now, when you are talk about the time. Now, present instants I am in a state but in future I can come back to the particular state again but still excluding that particular time. We are having the fix number of states. But it turn what happened we can say that now we are having more number of time but again we are going to represent this whole system with the help of this finite state model. And, this is basically going back issues is going to say this is the another timing instants.

Now, this is the model our systems say we are going to design some controller or we are going to design some circuit. First of fall we have to capture this design with the help of

this models, some models. We will see in what way we are going to capture. After that secondly what will happen knowledge seeking is nothing but the property verification. Somehow we have to specify our property or we have to give the property. So, this is the some in Greek notation we have written something and we have said this is the specification we have given to it.

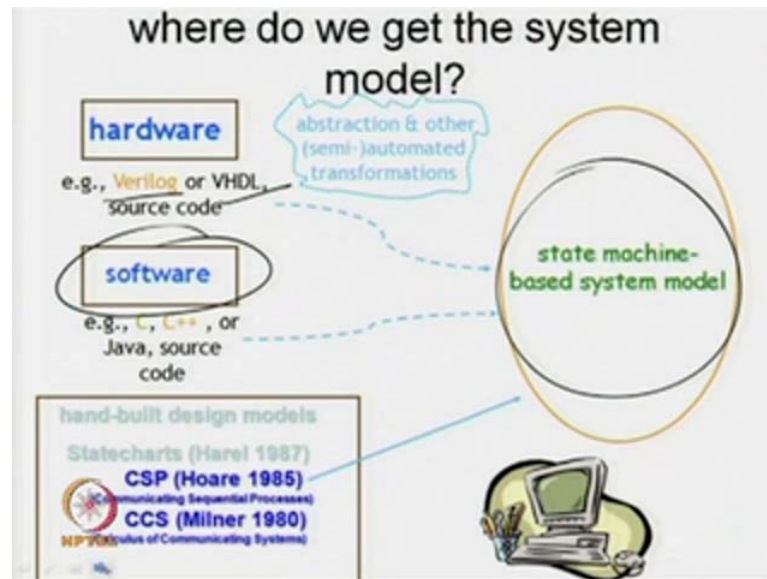
Now, in both these component will be even as an input to my verification tools. And, after that now this specification tool is going to check whether the given specification is proving in this model or not. So, what is the output of this particular model of a tool verification tool or a model checking? If this property is indeed true in this model then it will just give you the output or it will said at that true. That means the given specification is true in my model but if it is not true; just said that we have come out with a design. And, we have trying to check whether this particular specification is true or not.

So, the verification tool try to check all possible combination and it finds that this specification is not true over here. So, that my specification tools is simply it will not simply said that it is false. Along with that it will give me some information this is basically it will give me this particular error trace. So, in this particular error trace what happens it give me the error trace in such a it says that if you follow this particular path then the given property is not true. That means the design theme is getting some feedback on this particular verification method.

Now, they can concentrate the design issues for those particular executions trace all this. So, basically it will happen correct. So, our bugs are concentrate over here some sort of focusing the bug and design theme can now look into the issues related to details but it does not over will debt error or not present in the other section. It may present other places also because when designer going to fix this particular bug, then next time it may say that in some other patrols but up with this particular model seeking application by applying it repeatedly eventually we can capture most of the error and we can fix it.

So, it is somehow giving the not only saying that it is false but along with; it gives some indication also where the possible error is.

(Refer Slide Time: 47:10)

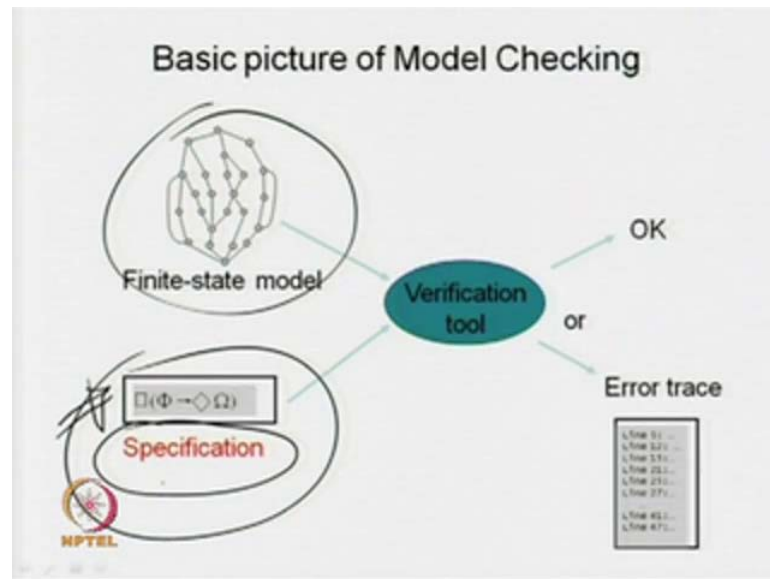


Now, where from we are going to get this particular state base machine I am saying that we need a model. So, in the previous slide I said that these are the 2 inputs. 1 is specification, 2 nd one is model where from we are going to get this particular model. So, just see that we are looking for this particular model.

So, different people are designing the system in different way either it may be hardware or it may be software also. And, already I have mention that since I am going to talk about one method verification method it can be applied for all design cases, it may be hardware or software. So in case of hardware what happens we describe our system in RTL level with the help of some HTL language. Hardware language like verilog or VHDL so we can have those particular verilog and VHDL.

If it is software then we can have C, C plus and like that. And, some other design can be done with the help of that which is define by your CSP, CSS. So, these are different way we are having a presentation but from that we can capture our state based machine. So, we need to have formalism to get eventually we should get this particular states machine for our model checker.

(Refer Slide Time: 48: 28)



After that this is the things now we have to look for this specification how we are going to give it. So, this is basically we are going to use temporal logic this specify our property the symbol box diamonds are having their own notation, own meaning.

In this particular model of temporal logic, we are going to basically talk about or going to discuss about this particular notation. What those particular notation means, and what we can specify. So, basically now in this case what happens? You see that what you have seen in that we are going to apply for a particular verification technique, which your model is checking. It needs 2 component. 1 is model and one is your specification. We will go to look into how we are going to give the model and how we are going to give the specification. So, next class we are going to discuss about the temporal logic and it is going to see about what is syntax and semantics of temporal logic; and what we can specify.

What is the expressive power of the temporal logic? So, next class we are going look into this issue.

(Refer Slide time: 49:36)

Questions

1. What are the problems with simulation based validation method.
2. Why Formal methods did not get acceptance in industry earlier.
3. What are the advantages of using formal methods for design verification.
4. Why it is difficult to use HOL in verification.
5. Try to find out major system design failure like Pentium Bug.



So, after going through this particular lecture you just see that some simple problems that you can think about. Simple question that first question that I am putting it like that. What are the problems with simulation based validation method. All ready I have given you some idea about what happens in simulation what happens in your exhaustive simulation and non exhaustive simulation. Why cannot go for the exhaustive simulation always. So, these are something like that you can get some more information in your book or some net; just see that what are things why it is not visible at present. Due to the advancement of your technology also it is not visible because in a small silicon space; you can put more and more devices.

2 nd just problem, I have already mentioned in my lecture also. Why formal methods did not get acceptance in industry earlier? Because, it cannot be automated secondly human intervention is required that means we need people that particular domain from automated theorem proofing domain because you have to guide that theorem while doing this things. So, that is why there are slightly reactant and they tried with you simulation base method. But Pentium bug has holds the people to go for these particular formal matters.

3 rd questions, just I am saying that what are the advantages of using formal methods for design verification. I am not mention probably but in the due course we are going to talk about it but you just think or you with just try to explore information in net and like that. 1 basic thing is that we are going to use formal mechanism to specify our system or to

model our system. We use some formal mechanism to give the property, the formalism that we are going to use our having a predefined syntax and predefined semantics.

So, ambiguity will be removed, when will pass through the design teams. So, this is a design it is going to the next step verification team or it will go to the implementation team. Then, it will remove the ambiguity basically formally we have defining it so all are having predefined syntax and semantics. Why it is difficult to use HOL in verification already I have mention you can look into it and some you are if so all verification matters and you can find some more idea.

Try to find out major system design failure like Pentium bug. So, it is Pentium is not only the bug that has been reported. After going into the public some more issues also there some other devices are also there which has reported bug after it is design, after it is going to public. You will get some information and try to compare this particular information and make a report. And, see what are the failures that we have to the design error. And, what problems that human beings are facing due to those particular error. So, that problem fine we can think about like that you are going to write a report about it and try to collect that particular information. We are having several design error fault in our history. So, with this I am winding up my lecture today. So, next class we are going to talk about or we are going to discuss about the temporal logic about it syntax and semantic.

Bye, bye.