Synthesis of Digital Systems

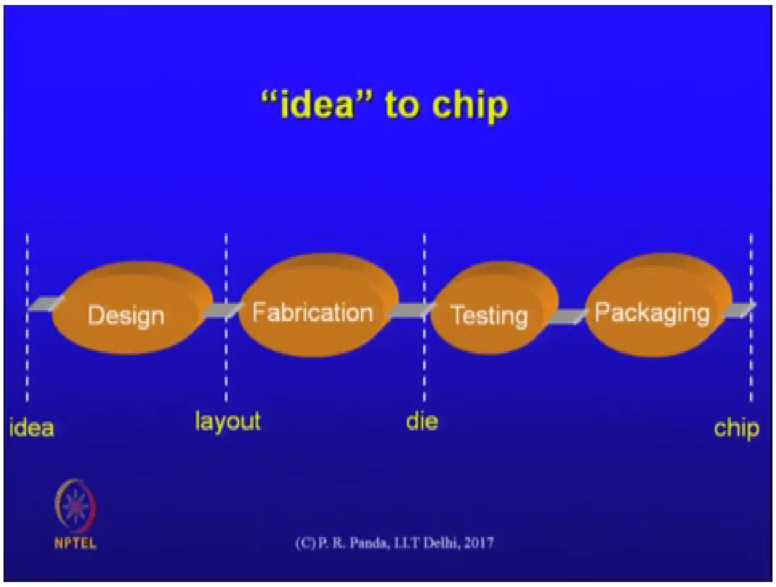
Dr. Preeti Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 02
Chip Design Flow and Hardware Modelling

So, let us start today with the setting of the context we had defined what synthesis was earlier. And before we go into the topic itself, let us talk a little bit about the overall chip design flow and where is it that design automation fits into the picture and within that overall flow where synthesis fits into the picture. So, that will be the topic of today's discussion.

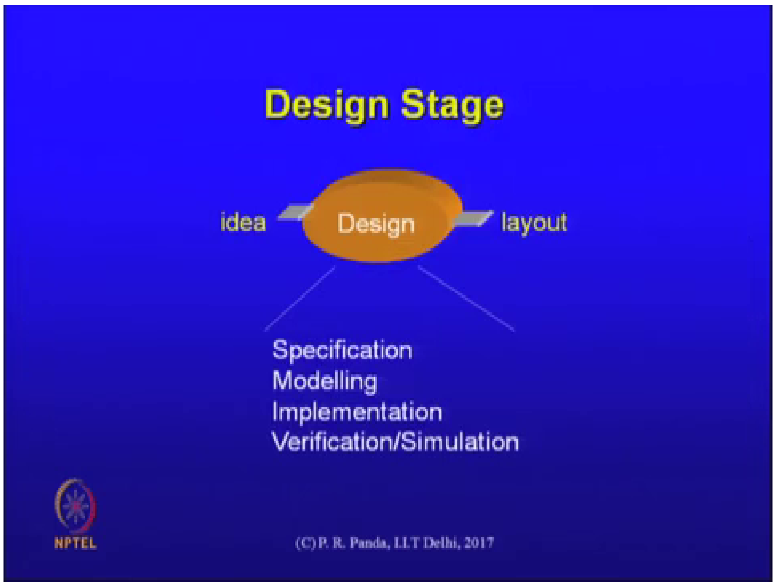
So, let us start with the high level picture of an idea transformed ultimately into a realization in the form of a chip.

(Refer Slide Time: 01:08)



What are some of the stages? Let us say the high level stages could start with the design phase which is an encapsulation of a number of different activities that go into what is called design. But at the end of it we have the layout which as we talked about the other day is a representation of the geometry of the system. There is a processing or a fabrication stage after that at the end of which we have the dies, those are ready for testing and packaging which ultimately results in the chip. So, if we were to abstract that flow into just 3 or 4 very high level stages it would be these.

(Refer Slide Time: 01:58)



Design Stage

idea
Design
layout

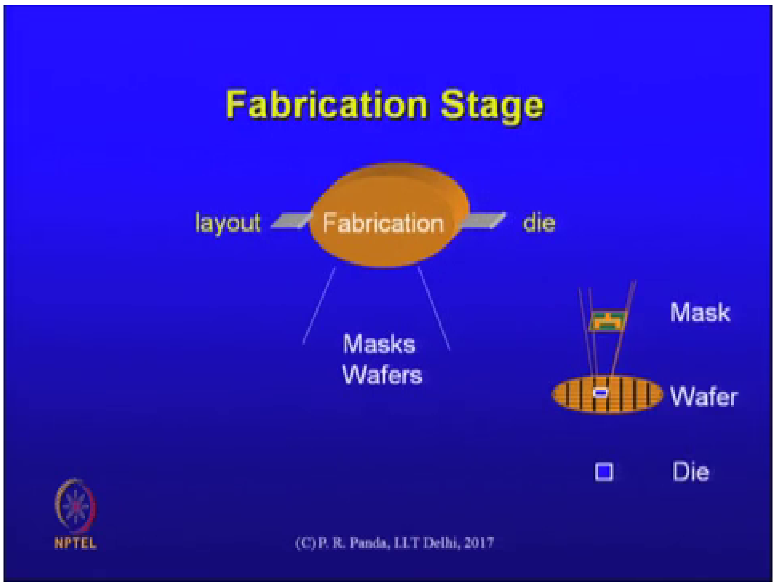
Specification
Modelling
Implementation
Verification/Simulation

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

What is involved in each of these stages? Design itself of course, is a very complicated set of steps we start with the specification of what the system should be doing, modelling of the system we will get into some more detail of what modelling means. Proceed to implementation of the system and following implementation there would be some kind of verification that we would have to perform just to make sure that this the resulting system is what was expected.

(Refer Slide Time: 02:33)



Fabrication Stage

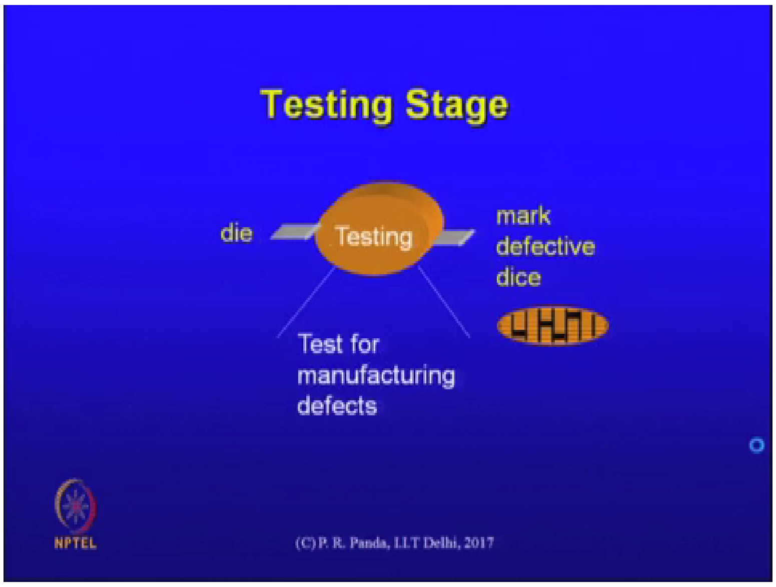
Masks
Wafers

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

That is design fabrication, again if it were to be encapsulated in just one picture it might be this that the input is a representation of what the geometry should look like and that leads to a series of steps starting with the creation of a set of masks and resulting ultimately in wafers which would be dissected into a set of dies that we are interested in.

(Refer Slide Time: 03:03)



Testing Stage

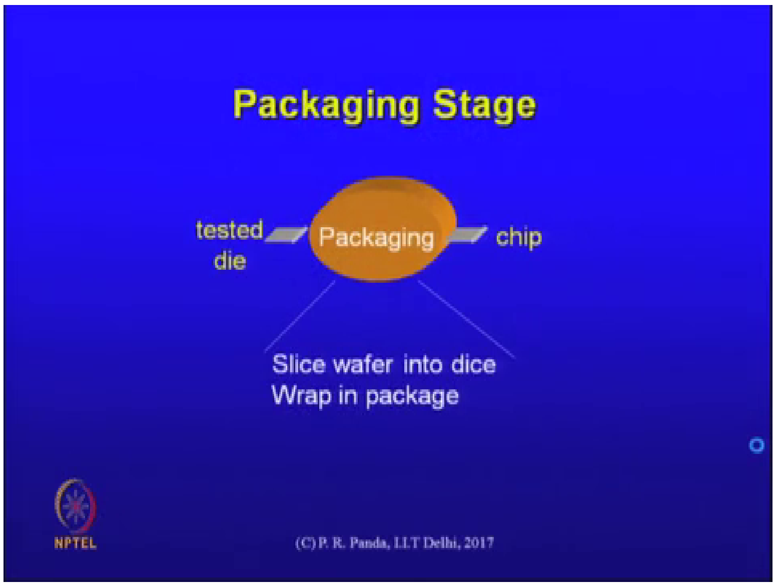
die
Testing
mark defective dice
Test for manufacturing defects
NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

Testing is the name given to a specific step in this design automation stages. As a verification step it is something that goes on all the time right from the time that we start with the modelling, but this testing is more specific it refers to a die being manufactured and then us running some tests to essentially check for manufacturing defects. Even though the design is perfect as far as functionality is concerned and our simulation results tell us that everything is fine then too there may be manufacturing defects that might still show up in the system these are essentially random in nature they hard to predict. So, you do have to run some simple tests on each of these individual dies just to make sure that there are no obvious defects.

So, that is this testing. This testing ought to be seen differently from the verification and simulation steps that we had seen earlier. Those are on a representation of the chip, whereas this testing is on the manufactured die. Even if everything is fine with respect to the design there may still be defects in the dies and we need to catch those in this testing phase.

(Refer Slide Time: 04:26)



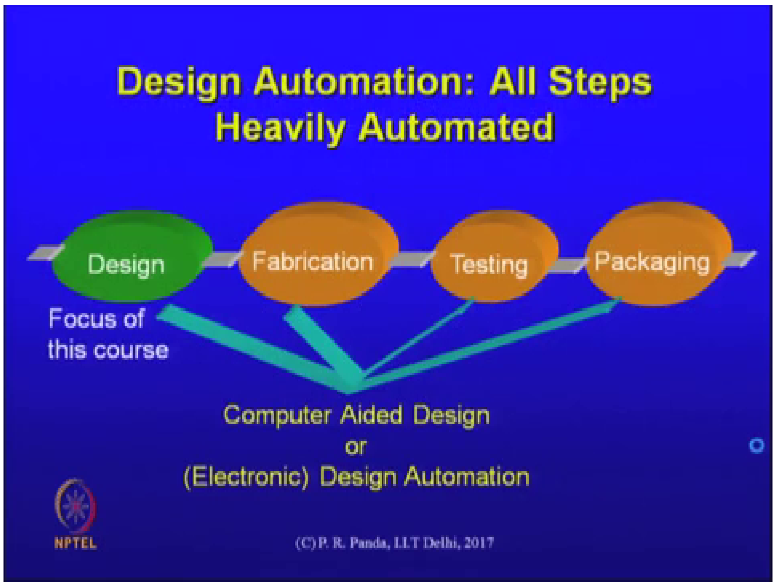
Packaging Stage

Slice wafer into dice
Wrap in package

Packaging once the die is tested and is validated to be ok, there is packaging step which ultimately results in the chip. So, these would be the 3 or 4 high level stages that we go through in chip design.

So, this is just to set the high level context.

(Refer Slide Time: 04:47)



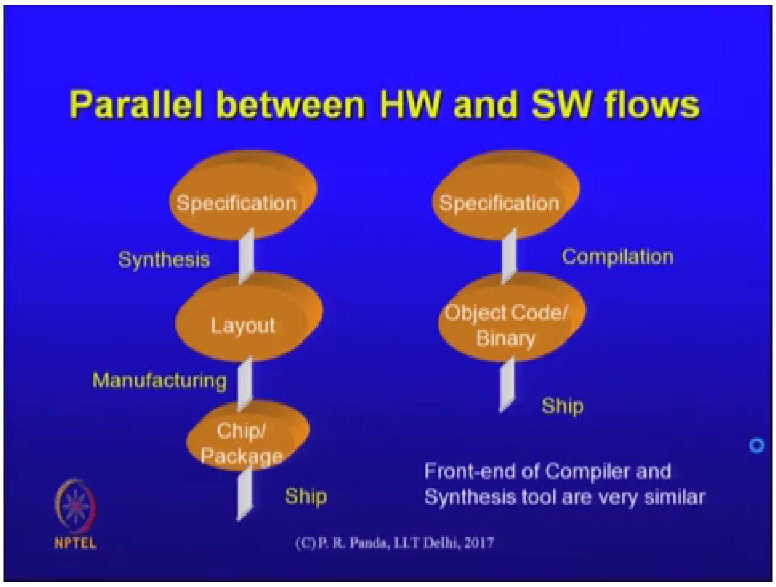
Design Automation: All Steps Heavily Automated

Computer Aided Design
or
(Electronic) Design Automation

The thing to note is that these steps are very heavily automated every one of these starting from design to fabrication, testing, packaging and so on all of these are heavily automated and over all forms this domain called computer aided that automation is what is called CAD, VLSI or electronic design automation. All the automation steps involved in, going through the design phase, the fabrication phase and so on all of these falls under what is called CAD VLSI.

Focus of this course is clearly in the design stage out of these stages, but also within the design stage it is more narrowly focused on the synthesis which is part of the design phase not everything, but that is the overall context.

(Refer Slide Time: 05:43)



Parallel between HW and SW flows

Front-end of Compiler and Synthesis tool are very similar

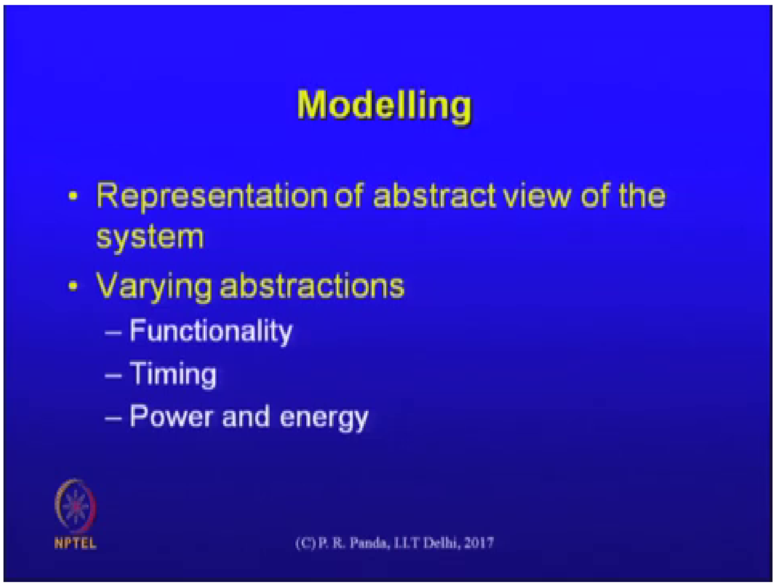
(C) P. R. Panda, I.I.T Delhi, 2017

This is a parallel that we talked about in the other class and let us also reiterate just to identify some of the differences. So, consider a hardware design flow chip design flow versus a software flow.

In both cases we start with a specification of the system. We go through a translation step that in one case is called synthesis in the hardware case; in the other case it is called just a compilation. The outputs are very different like we had seen in one case the output is a layout in the other the output is object code or binary consisting of a stream of instructions. In the software flow that is all that is there that is the end product if you have to ship that software to somebody then that object code is what you would be shipping. Here though in the hardware line the layout is not what you would be shipping typically that is not the end product that is not the chip that does go through another set of manufacturing processes before you have the chip. So, nevertheless at the front end of the compiler the software flow and the hardware flow are very similar, but the back end would be very different.

One thing to note is that what shipped here essentially you produce one true copy of in the software flow and you just replicate as many times as you need to deliver right. But in the hardware flow that is different the manufacturing step does lead to some uncertainties it leads to some defects. So, all chips that are manufacture are not necessarily correct. So, you have to throw out some of them and the ones that you evaluated to be are shipped to the customer ultimately. So, in the software flow you typically make as many copies as you want you do not expect defects there in the process of making those copies, but in the hardware flow you do expect defects in the process of making the copies.

(Refer Slide Time: 07:49)



Modelling

- Representation of abstract view of the system
- Varying abstractions
  - Functionality
  - Timing
  - Power and energy

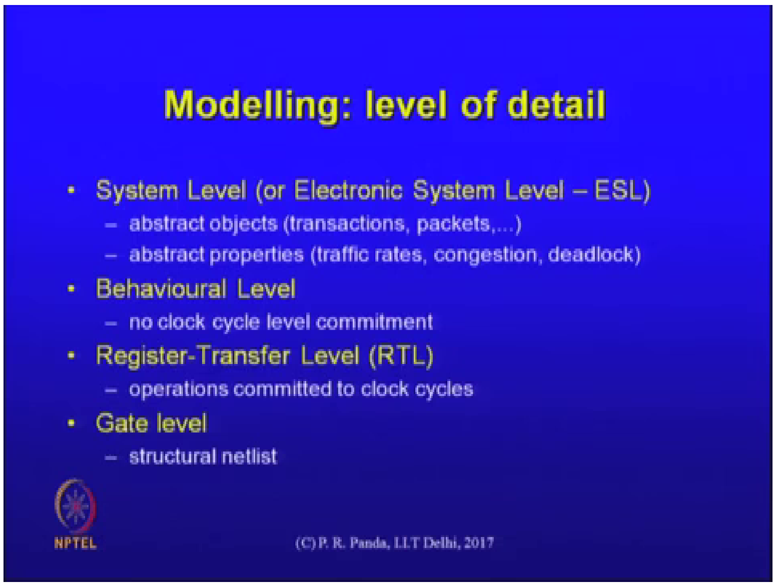
(C) P. R. Panda, I.I.T Delhi, 2017

So, that is again to reiterate the parallel between the hardware and software flows. Let us take a look at what is involved in the design stage.

We said that we had there was specification, there is modeling, synthesis simulation and so on. Let us start off with modelling what is involved in that. What is modeling? It is a representation of an abstract view of the system abstract as opposed to detailed, meaning that some aspects of the working of the system are captured properly not necessarily all aspects. We will talk about what are those abstractions, but it could be that we have functionality of the system being captured as part of the modeling. Functionality as opposed to let us say timing of the system. So, these are different and often orthogonal aspects of the modelling of a system, maybe the system is implementing some algorithm and you represent that algorithm in terms of some language or something like that that is the functionality that is captured.

As you indicate the functionality you do not necessarily give too much emphasis on how much time it takes to execute that functionality. Even though that is also part of the system, that is also part of the requirement of the system. You may be phasing out the modelling of the system in a sequence such that some aspect of it is captured in one phase some other aspect is captured at a different phase and so on. So, it could be that functionality is captured in one phase in that phase you do not care too much about the timing, but timing could be separately modeled and verified, other abstractions could be about power about energy and so on. So, many different parameters that we are interested in would need to be modeled in some ways and that is what we mean by there is levels of abstraction at which the modelling is done. Abstraction essentially means that we capture a subset of the essential features all of the features are not necessarily captured then.

(Refer Slide Time: 10:06)



Modelling: level of detail

- System Level (or Electronic System Level – ESL)
  - abstract objects (transactions, packets,...)
  - abstract properties (traffic rates, congestion, deadlock)
- Behavioural Level
  - no clock cycle level commitment
- Register-Transfer Level (RTL)
  - operations committed to clock cycles
- Gate level
  - structural netlist

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017

Let us look at the modelling in some more detail. What kind of modelling can we do? It could be at a system level or in electronic system level of abstraction. There we are talking about relatively abstract objects like transactions or packets or requests responses and so on. This is a little somewhat distant from a hardware view or pure hardware view of the system. So, such kind of objects and properties such as traffic rates or a congestion or a deadlock if there were some kind of a network that is being implemented then these are the properties that you are interested in and those transactions and modules and traffic and so on these are these objects that we are interested in. The description would be in terms of these objects and properties and the analysis of the system would also be in terms of those properties. These we can say are high level of abstraction forming the system level.

Going down one level of detail we can get to a hardware level, actually at this level of the system that we are talking you might not even have decided which parts of the system would be implemented in hardware and which other parts would be implemented in software. So, at that level of abstraction you might not have made that commitment yet. But at some point maybe between the system and this and the hardware or the software level of abstraction you would have made that choice and if you have decided to go down the hardware route there too you can think of a few different levels of abstraction, starting with the behavioral level we will go into some more detail on these hardware levels of abstraction in later stages of this course.

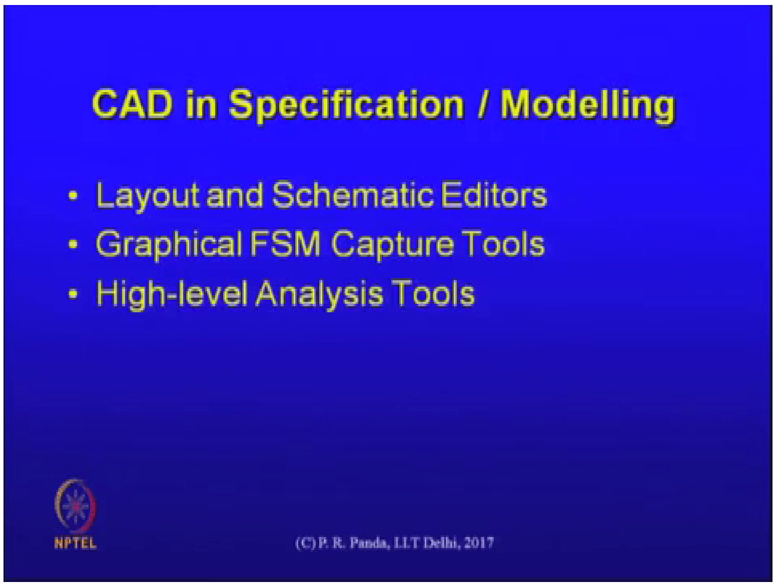
But behavioral level is one level of hardware abstraction where you capture some essential features of the design, but not necessarily all the details. One example of what you would omit would be you do not say what operation is to be performed when. So, that is what we mean by saying there is no clock cycle level commitment. So, there is a partial description of the idea of clocking perhaps the fact that there is a clock to the system that is indicated, the inputs to the system are indicated, the outputs of the system are computed so what the computation is those are clearly indicated, but which part of the computation happens when that is not necessarily indicated. When the designer specifies something at the behavioral level, that particular detail could be omitted.

Next down we say that is what is called an RTL level or register transfer level of abstraction. At that level we say that we have made all the decisions of what operation will be performed in which clock cycle. It is an important abstraction change as we go down from the behavioral to the RTL level. The distinction essentially is that well there is several differences, but one important difference is that when we go down to the RTL level we have made up this decision of pinning down operations in to clock periods this is when each operation will happen. It is an important distinction. In the behavioral level we say what that computation is.

We might even indicate high levels of constraint saying all of this that I have indicated should be finished within 10 clock cycles. That is also a constraint that may that one may provide to a behavioral synthesis tool whose job it would be to take us from the behavioral level of abstraction down into the RTL level. But key elements that are not indicated by the designer are when each operation will happen. We can indicate the entire computation and we can say that all of it must be finished within a given number of clock cycles, but that is still not enough for us to produce the whole design we have to make some further decisions those decisions would be part of a synthesis system. Mapping of operations to clock cycles are exercises that have already been taken when we go down into an RTL level of description. So, that is an important difference in abstraction.

Beyond that there is a gate level when we go down into a more structural form what do you think the RTL representation looks like it. In fact, it is indicated in a hardware description language also, just like the behavioral level of abstraction. It is still functionality that is captured. So, you still have your programming constructs like if statements and loops and so on it is still an RTL design, but the description of functionality could well be specified in text form. But that is different when you go down further into the gate level of abstraction that is when you have structure explicitly built in and you have a structural net list that actually is the representation of our design.

(Refer Slide Time: 15:37)



CAD in Specification / Modelling

- Layout and Schematic Editors
- Graphical FSM Capture Tools
- High-level Analysis Tools

(C) P. R. Panda, I.I.T Delhi, 2017

So, with that as the different levels of abstraction let us quickly go through those steps that were involved in the design stage and indicate where is the scope for automation. We say that CAD and design automation pervades all through the chip design flow. Let us quickly go through just the design step and within that indicate what kind of CAD support is relevant.

The first step is that of specification and modelling that is when the designer indicates what is it that is the functionality of the chip. What kind of automation could be involved there? Clearly we need some mechanism to be able to specify that. So, depending on the level of abstraction you may think of different tools that could help in just this capture part. If layout is being specified or gate level net list is being specified there are these layout editors and schematic editors which you might already have used in various contexts that is part of the CAD that is involved there.

Little higher level of abstraction when it comes to finite state machines you can think of tools to capture those finite state machine you know what that representation looks like you could specify. How do you specify a finite state machine?

Student: State diagrams.

In the form of a state diagram or equivalently what other ways are there of capturing a finite state machine. Yes, a state table is a standard way of capturing, a finite state. Both of them are equivalent whether it is in diagrammatic form or a tabular form the information is the same and considering that these mechanisms are there for a designer to specify you can think of tools to capture either a table that state table or just a graphical tool for capturing the diagram.

Capturing goes hand in hand with the analysis once you have captured it there is some high level analysis tool that you can build into that same framework. What kind of analysis involves? Suppose that I have a tool for capturing graphically a finite state machine, what analysis could I perform on it? If I could perform some analysis that analysis could there, could also be another tool that is connected to this capture tool. What analysis could be performed on a finite state machine?

Student: Coverage functional coverage (Refer Time: 18:14).

Yeah, there are a lot of things that you can do once you have captured the FSM which is kind of an intent from the designer the sum of these analysis you have already done. It could be that you indicate to the designer that certain states are unreachable right; when will it happen that in an FSM a state is unreachable. You know what that FSM looks like right. It is just a set of states indicated by nodes in a graph and a set of transitions, would it happen that you have drawn a state which is not reachable.

Student: Yes, sir.

When will it happen?

Student: (Refer Time: 18:52) conditional.

Yes, the transitions into the states are controlled by a condition that you yourself specify. It is possible that we make some mistakes in the specification of those conditions that is also part of the specification. It could be that we have written that condition in a way that that condition is never true. You can write some expressions that never evaluate to true. So, some analysis one can certainly perform that just works of this specification that has been captured in the form of a final state machine.

Student: (Refer Time: 19:35).

Now, well this is an example of the analysis. In that particular analysis we said that we will analyze whatever you have provided whatever the designer has provided and give some feedback. That state is unreachable is some feedback that can be given to the disciple what other kind of analysis can be performed on an FSM. So, if feed this feedback is given to the designer he could possibly improve the design in some way, but what other things could we do.

Student: Complexity, complexity of n number of.

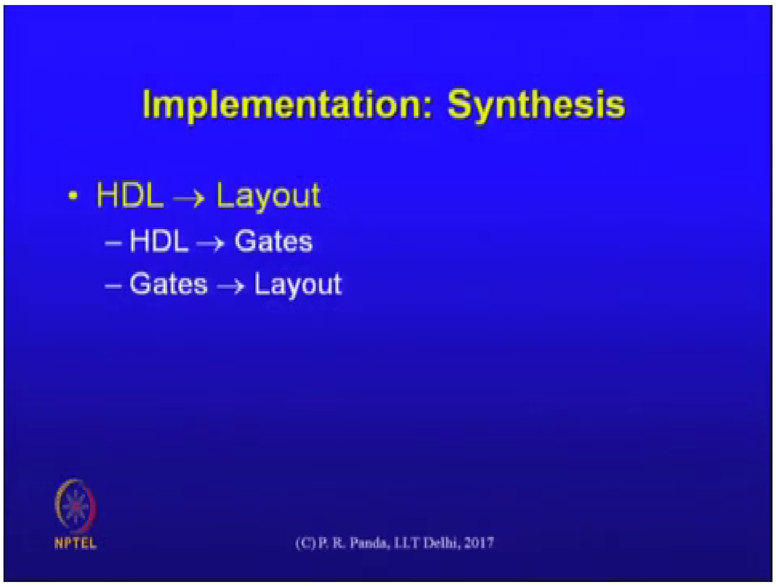
What do you mean by complexity of an FSM?

Student: If I could achieve the same functionality is in a simpler fewer number of states.

Yes. We have gone through this process of identifying redundant states. There is nothing wrong in it, but it is just that two states that you have indicated are actually redundant. This is not the same as unreachable you could reach one or you could reach the other, but the meaning would be the same. So, the tool could actually go through, you have gone through such an analysis earlier not necessarily automatically you have gone through the logic through which you can determine that two states are equivalent. So, these are examples of analysis that we could perform. It could be more sophisticated than that, but of course, this course is about such analysis that we perform.

So, as of now we are just pointing out that there is scope for automation at all these stages of the design flow. The more details we will get to as we talk about them.

(Refer Slide Time: 21:16)



Implementation: Synthesis

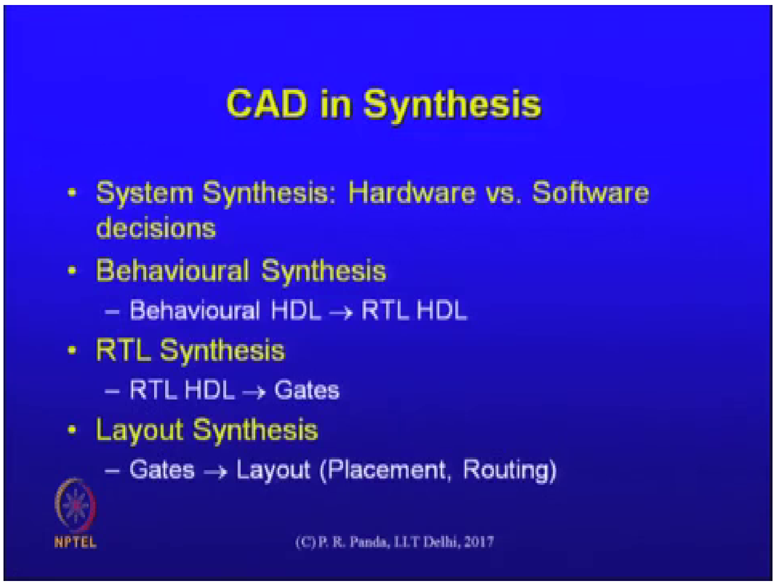
- HDL → Layout
  - HDL → Gates
  - Gates → Layout

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

That is about the capture at the very early stage one could introduce some design automation. Moving on to implementation you have captured the intent in the form of an HDL and you have to go further and ultimately we need to reach the layout we can think of reaching there through a few different steps. So, initially I go through some set of synthesis steps that take me down to gates and maybe at a later stage I invoke some other set of these physical synthesis tools to take me from the gates down into the layout. So, this step is what is called synthesis and is the focus of this course.

(Refer Slide Time: 22:00)



CAD in Synthesis

- System Synthesis: Hardware vs. Software decisions
- Behavioural Synthesis
  - Behavioural HDL → RTL HDL
- RTL Synthesis
  - RTL HDL → Gates
- Layout Synthesis
  - Gates → Layout (Placement, Routing)

NPTEL

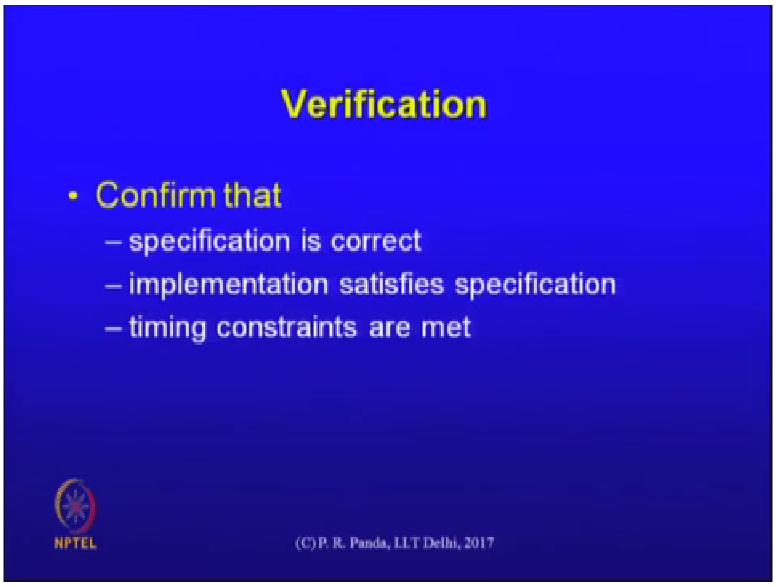
(C) P. R. Panda, I.I.T Delhi, 2017

What kind of design automation is involved in synthesis? First of all from the system level we could do some system synthesis that involves answering some important questions like hardware versus software implementation. The specification has been captured, but at a certain high level of abstraction we have not even indicated, the designer has not even indicated which module is going to be implemented in hardware or in software. In fact, we could introduce some tools there and some analysis there that would actually help make that decision of which modules should be taken further into the hardware flow. So, that is one.

Having decided to take a module into the hardware flow there are different levels of synthesis we can think of behavioral synthesis would take us down from that behavioral level of abstraction into the RTL level of abstraction.

Remember the difference between the two. That is primarily the task of the behavioral synthesis. Once you are down here we would go through another set of RTL synthesis steps that take us from an RTL level HDL down into a gate level of abstraction. Layout and physical synthesis tools would help us in taking those gates down into a layout. What is involved here are steps like placement of the modules, and the gates and the routing of the wires between them. So, these are the kind of design automation steps involved in synthesis. So, even though we call the whole thing synthesis the kind of problems are very different that are addressed at all these different levels of abstraction.

(Refer Slide Time: 23:51)



Verification

- Confirm that
  - specification is correct
  - implementation satisfies specification
  - timing constraints are met

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

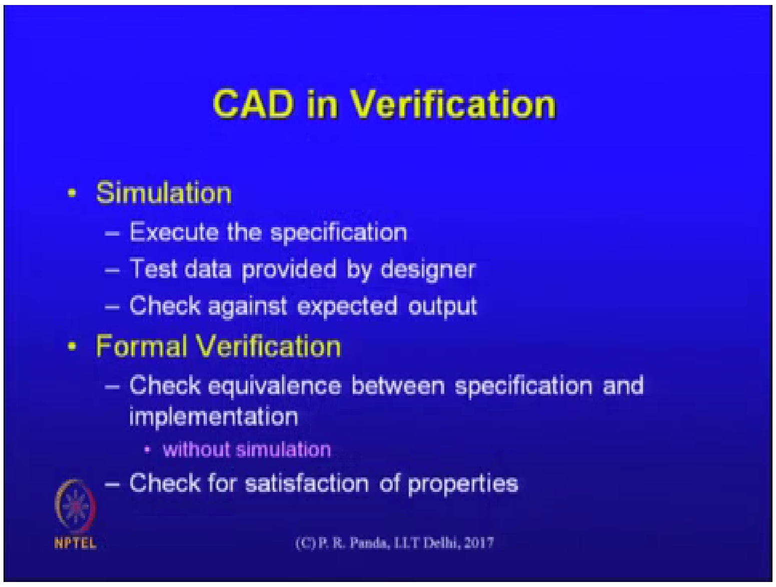
Verification, designing is not enough there has to be an associated step of confidence building in that design which is established through various verification mechanisms. All kinds of questions need to be answered after we have done a design with respect to whether we are satisfied that first of all the specification is correct. Specification is indicated by us in some way there could be bugs in the specification itself right. So, if you start off with specification that is faulty clearly; however, good a job you do of the synthesis tools and implementation tools the result of course, is faulty because there was an error in the specification early on.

So, we do need to confirm first that the specification is correct, if we are satisfied that the specification is correct then there is the question of being satisfied that the implementation that is generated at the end of this design steps actually satisfies the specification. For that same specification there might be many different implementations all of which are ok, they might even be equivalent with respect to each other, but like we said the other day all might not be equally good there is an efficiency criteria that is used by our synthesis mechanisms to finally, select one out of the many different possibilities. But it is a tool after all and the tools might have errors in them and therefore, whatever is the output wherever we are with respect to the level of abstraction we translated something from a higher level down to a lower level of abstraction and there is the need to answer this question of does this implementation actually satisfy the specification. Wherever we started from and whatever the next step was.

This is with respect to functionality, but it could also be with respect to timing. Timing constraints may be imposed by the system very early on as part of a design specification and the way we do the implementation we need to make sure that those timing constraints are met otherwise it is not a valid design. When the timing constraint is imposed it is not clear, but you could easily imagine one common scenario where at the very early stages of the specification you define that this is the functionality of the chip, but you also say that it must operate at 1 gigahertz or 2 gigahertz right. That is the product definition and it comes at very early stages. So, that is the timing constraint if you are saying 1 gigahertz is the frequency at which the final product will operate that is actually specified very early on and every step of the implementation that you are going through must be aware of that timing constraint and produce a an implementation that satisfies that timing constraint.

So, whether the timing constraints are actually met or not is part of a verification mechanism somehow.

(Refer Slide Time: 27:00)



CAD in Verification

- Simulation
  - Execute the specification
  - Test data provided by designer
  - Check against expected output
- Formal Verification
  - Check equivalence between specification and implementation
    - without simulation
  - Check for satisfaction of properties

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

What kind of design automation is involved in verification? How do you actually ensure that functionality is correct this is a difficult problem in general and while it is hard to establish in a fullproof way that the entire chip is designed correctly, you may actually divide the problem into divide the chip itself into smaller modules and ask more interesting questions with respect to whether those smaller modules are ok or not and also resulting interaction whether that is or not. So, at least in parts we ought to be able to verify the functionality of the system.

Simulation is a common mechanism having designed it and having implemented the system you execute the specification that test data is provided by the designer or a verification engineer and you check the output of the simulation against an expected output of the system. So, that is simulation. This is a common mechanism and normally chips that are designed you always take it through simulation, but often is not sufficient. Why simulation not good enough or why would it not help in establishing that the chip will work correctly under any circumstance that you can imagine.

There some fundamental limitations might be there in simulation what might they be?

Student: The word any.

Yeah, the word any what is the problem.

Student: Cannot come up with those many situations to stimulate (Refer Time: 28:41).

Yes. So, we are defining this as verification through execution. So, execution means that you are defined a system and you generate inputs and you observe the output of the system for those inputs, which means that to actually guarantee that the system is fine, you have to generate all possible input combinations and observe the response of the system for every possible input combination. Not just that it is not merely the combinations of the input the other thing that is that comes into the picture is that the sequence of inputs is you, not only have to produce every combination of the inputs, but every sequence of those combinations because our designs are sequential they are not merely combinational designs. But even if we were to restrict ourselves to combinational logic is it easy or difficult to generate all possible combinations is difficult, why, where is the difficulty?

Student: (Refer Time: 29:44) microprocessors you need to generate to give power to eliminate combinations.

Yeah, it should be obvious that the number of combinations grows exponentially with the number of inputs. Even if it is an adder that we are designing let us say just a 32 bit adder, how many inputs are there to such a system?

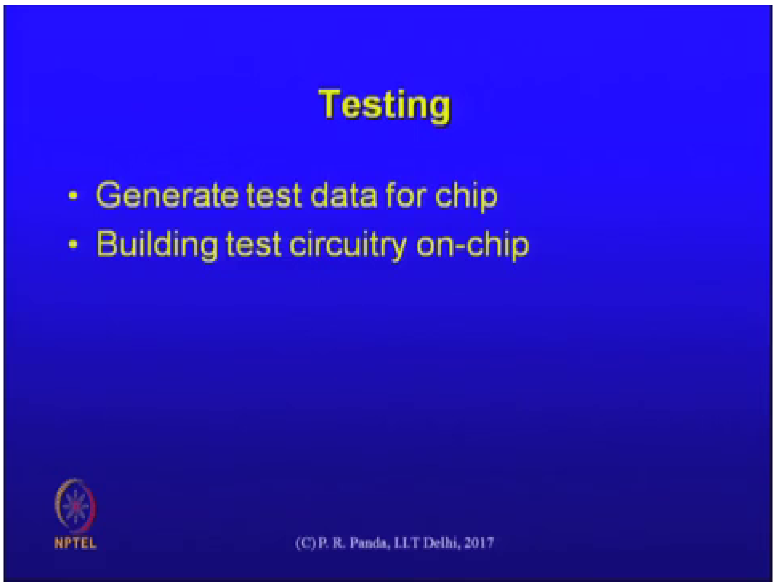
Student: (Refer Time: 30:07).

Yeah, there are two inputs of 32 bits each there may be a carry in also. So, there are 64 or 65 inputs and even if the numbers even, if the values of each input is constrained to be just 0 and 1 then too the number of combinations is 2 to the power 65 and there is no way of exhaustively testing whether that adder itself being a simple circuit that it is, is actually working correctly for all the input combinations. So, there is no hope of the simulation being exhaustive in any way. So, there is a need to expand the scope of verification beyond simulation and the formal verification is the mechanism that is somewhat nicely orthogonal. Its objective is somewhat different it does not rely on any test data provided from the designer. So, it does not do any simulation at all. But it for example, may check for equivalence between a specification and an implementation or it checks for satisfaction of certain properties.

We talked a while ago about entering a system in the form of a finite state machine and answering the question of whether a particular state is reachable or not. That would be an example of a verification you do not rely necessarily on any simulation input, you could look at that FSM if it is simple enough and you could analyze all those conditions that are specified on the transitions and verify theoretically by just looking at the your specification that certain state is unreachable. So, that is an example of a property satisfaction that we are looking for.

So, that is another kind of verification that is involved here. These are examples of design automation in the verification space.

(Refer Slide Time: 32:16)



Testing

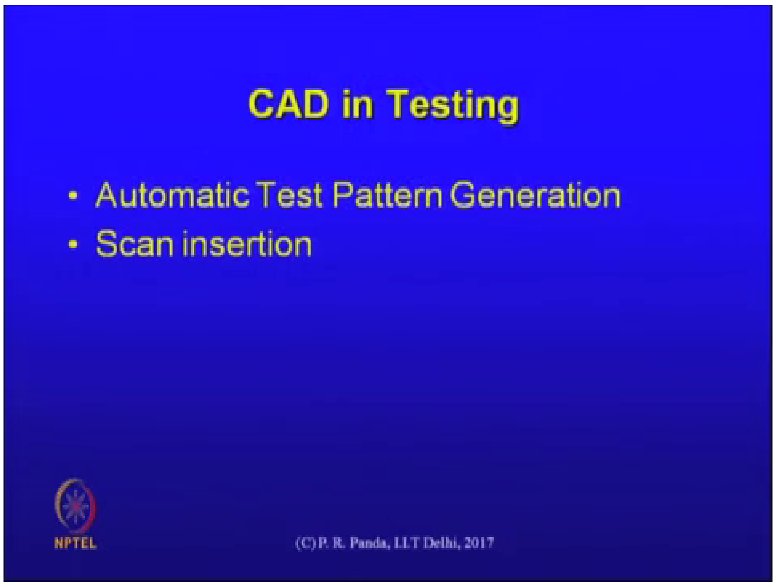
- Generate test data for chip
- Building test circuitry on-chip

NPTEL

(C) P. R. Panda, I.I.T Delhi, 2017

Move on to testing what is involved here you need to quickly check whether that die that is just produced is defective it suffers from some obvious electrical defects or manufacturing defects that is it. So, what is involved here? You generate some test data just like simulation, but you also have some hardware typically internal hardware that would check whether the internals are or not of the chip. So, some test circuitry is actually built on chip. So, these are examples of what falls into the testing domain.

(Refer Slide Time: 32:57)



CAD in Testing

- Automatic Test Pattern Generation
- Scan insertion

(C) P. R. Panda, I.I.T Delhi, 2017

What kind of CAD is involved there? A test pattern has to be generated these inputs that we said will provide manually at the simulation stage. This is another kind of execution testing involves a different kind of execution and there is an opportunity to automatically generate some test patterns, given the design to check whether there are any manufacturing defects.

There is an associated step called scan insertion. It turns out that to provide observability into the chip you need to tamper with the chip itself in some way. There is an issue with verifying that the manufactured chips are working which is different from verifying during simulation that your model is working ok, what do you think would be an, would be a difficulty. At the later stage when you already have the chip that is not there when you are running a simulation of a model of that chip.

Student: Parasitical (Refer Time: 33:58).

Parasitic effects are there in the real chip, but it is not as though it is not there at in the model that is being simulated that depends on what is it that you have modeled for. If your model is very simplistic and it did not take some of those effects then indeed you do not observe it, but we can do our modelling taking those effects into account. Like when we already have the layout there is enough information for us to answer many other questions.

The disadvantage of simulation is that you do not have enough time it just takes too much time and having the real system the testing is much much faster than simulating a model of the chip. So, that is an advantage that is what is better about doing the testing at of a manufactured chip well, what is the disadvantage?

Student: Sir, we are using practical component. So, we cannot know for sure how they will exactly behave I mean 100 percent we cannot predict the way.

Yeah, sure we already argued that even if our design process was perfect the manufacturing process may introduce defects. There is an example of an imperfect system. The job of testing is precisely to point out which ones have defects and which ones do not have defects.

Student: Some of the inputs may end into a situation where that chip is damaged.

Chip could be damaged for various reasons.

Student: (Refer Time: 35:36) is what you just said in the.

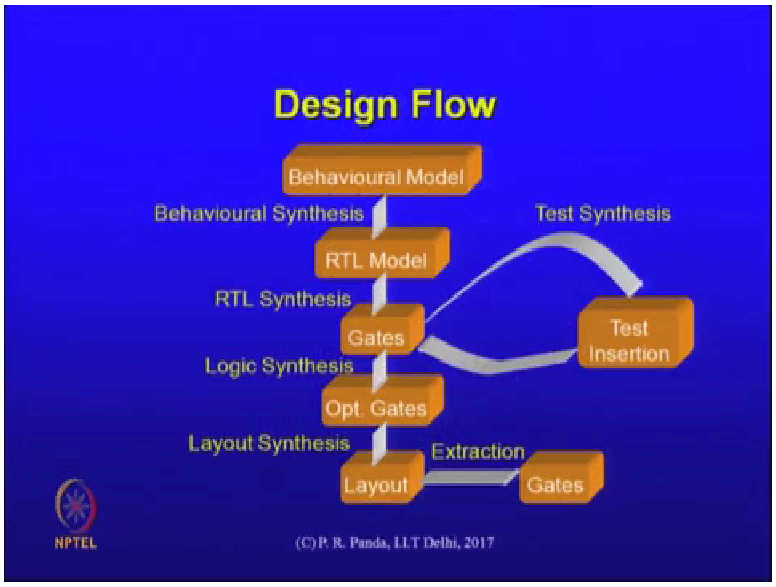
Observability, see the problem with the monitoring what is going on in the chip as opposed to monitoring what is going on in different parts of the component that is being simulated is that during the simulation as part of the simulation process you have access to your whole design right you could click on some output of a flip flop and check the value that is there currently at the same at that time instance.

The chip itself has a very limited number of external pins, it is not so easy to go to that particular flip flop and check what is the value. So, on the other hand we do need to do some elementary testing of that nature in this phase to make sure that the system is ok, right. So, to aid this there is a standard procedure that we follow that that is what the scan insertion is about we introduce the testing related infrastructure elements into the chip, its purpose is just to aid in the debugging aid in the visibility. So, conceptually what happens is that and large parts of the memory or interesting parts of the chips memory are connected together in a form of long shift register and ultimately we are able to scan out the elements of that shift register and indirectly observe the data that is there inside a chip. So, such mechanisms need to be inserted as part of the design automation for testing that is what we call scan instruction.

Student: Sir, the debug should also be (Refer Time: 37:12).

There is an interesting phase called post silicon debug, debug to just like the simulation we are debugging as we perform simulation, but that is debugging on the model of the chip. The testing while I did not point it out here indeed involves other interesting part the kinds of debug that you would perform on the chip that is actually produced yeah for simplicity it was left out, but indeed that is another interesting step. So, here we are just talking about observing, but after observing what kind of analysis can we do and also can you form out some of that analysis on chip into validation related hardware more sophisticated validation related hardware on chip. Then merely this can change that we would be inserting it is a very interesting modern area somewhat outside the scope of the synthesis, but we are talking about it just to understand the overall scope for computer aided design.

(Refer Slide Time: 38:19)



Design Flow

(C) P. R. Panda, I.I.T Delhi, 2017

Student: Sir, (Refer Time: 38:18) all chips (Refer Time: 38:20) is that done by the designer or does someone else do that job.

So, who designs the on chip test circuitry? Actually this overall flow should tell us something about what is happening where. So, here is a simplified version of the design flow in which we have labeled some of those steps. Let us say I start off with a behavioral model. So, this at this stage the assumption is that we are talking about a hardware implementation. Remember at the higher level at the system level that itself was not was not decided, but behavioral level the behavioral model of hardware we would go through a behavioral synthesis step to generate an equivalent of an RTL model.

Let us say there is an RTL synthesis step that takes us from the RTL model into gates this is more structural like we are seen it is a structural net list. This is where we may need to manipulate the design to insert our scan circuitry the test related circuitry. Essentially it may involve changing some of the existing flip flops into a more sophisticated kind of flip flop that can also be configured as the shift register that we were talking about. So, some of those flip flops not all because there is an area overhead associated with that insertion, but in conceptually you have inserted some memory elements to help us in tracking later on. So, that is what is performed as part of the design step all of this is technically performed by the designer.

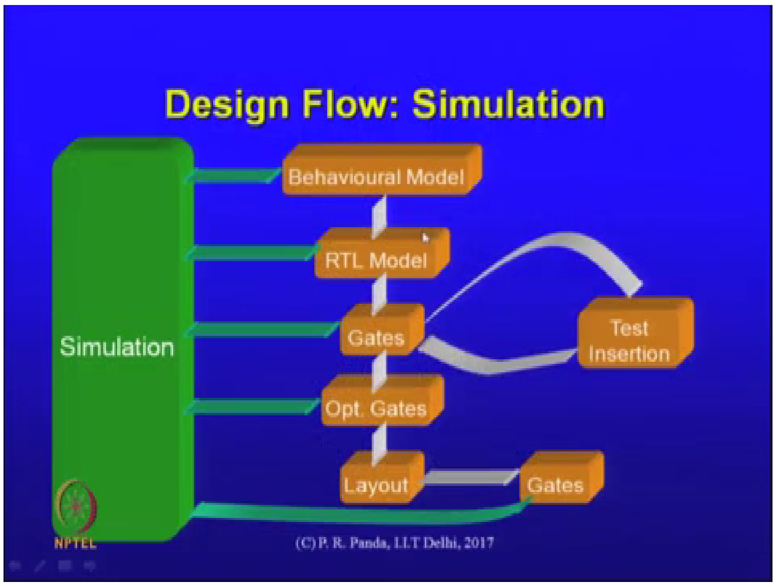
So, it is just that this step that test insertion is what is called a test synthesis that would be among the different synthesis steps that you would be performing. A gate level design then that is not the end of it there is this logic synthesis step that we can go through to optimize a bunch of gates this is not. So, what is indicated here is not necessarily optimized it is one form of representation of our design. A set of logic synthesis steps would take us into a more optimized form of design of that same gate level net list. Then we would go through a physical or a layout synthesis to generate the layout for us that would then be what is input to the next stage of fabrication and so on. There is another optional step here we could extract gates from the layout that was generated why would we do this.

What happens here is you look at the layout and extract from it a gate level circuit. You should be able to do it this is a digital design anyway. So, one should technically be able to extract these gate level back from the layout, what would they use be?

Student: (Refer Time: 41:21).

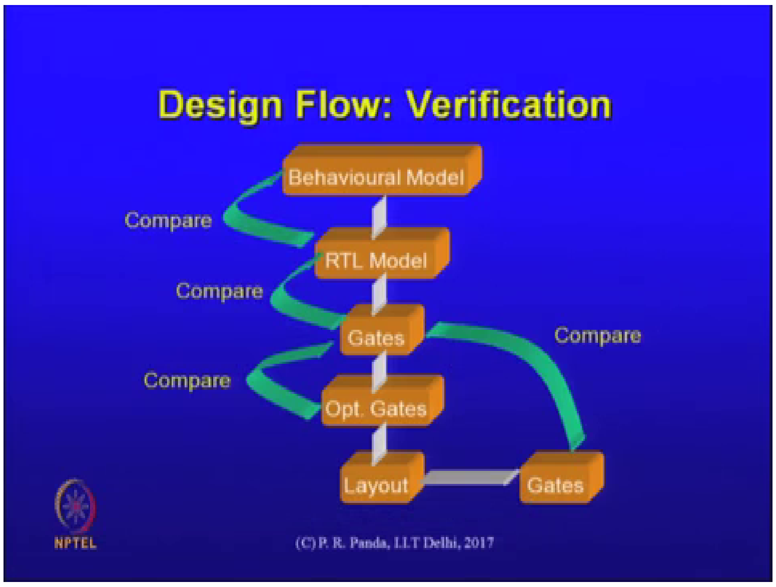
Yeah, it could help in establishing that their layered synthesis was among other things there was no problem in the layout synthesis. In every one of those steps since we will be using tools heavily it is certainly possible that errors are introduced at various stages. Verification is somehow ingrained in this design process we are not really separating them out, simulation is something that is relevant at every step of our design flow starting with the behavioral model, the RTL model, each one of these is still a model this optimized gates and so on these are gate level netlist and which should be able to simulate them for correctness.

(Refer Slide Time: 41:54)


Design Flow: Simulation

The extracted gates we could also simulate that for correctness just to make sure that everything is fine.

(Refer Slide Time: 42:10)


Design Flow: Verification

Other examples of verification would be a comparison formally, informally, manually or automatically and so on of each of those steps each is a translation step as we go in the forward direction it is a synthesis step that is taking us from a higher level of abstraction to lower level or more detailed level, at every step there is possibly a need to compare the two make sure that everything is fine.

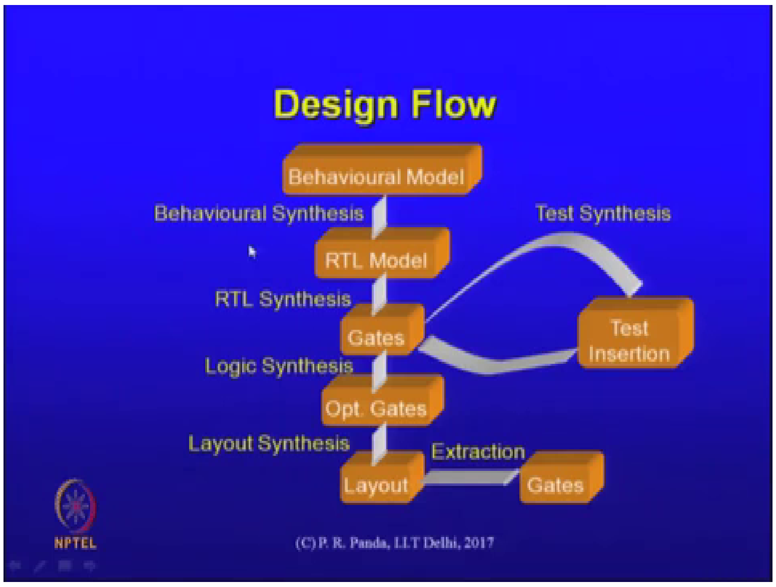
So, we are just indicating this as comparison all of them do not necessarily mean automatic comparison. Even the forward step the synthesis not all of those steps are necessarily automatic some of that synthesis s just happening in our minds. For example, going through from a behavioral level into an RTL level forms what the context of the behavioral synthesis that we indicated, but all of that could well be just steps the design and goes through and comes up with and enters the design directly in an RTL level of abstraction.

Student: (Refer Time: 43:12) high level synthesis which you are mentioning.

Yeah.

Student: All are similar or dissimilar is it with the RTL to get (Refer Time: 43:22).

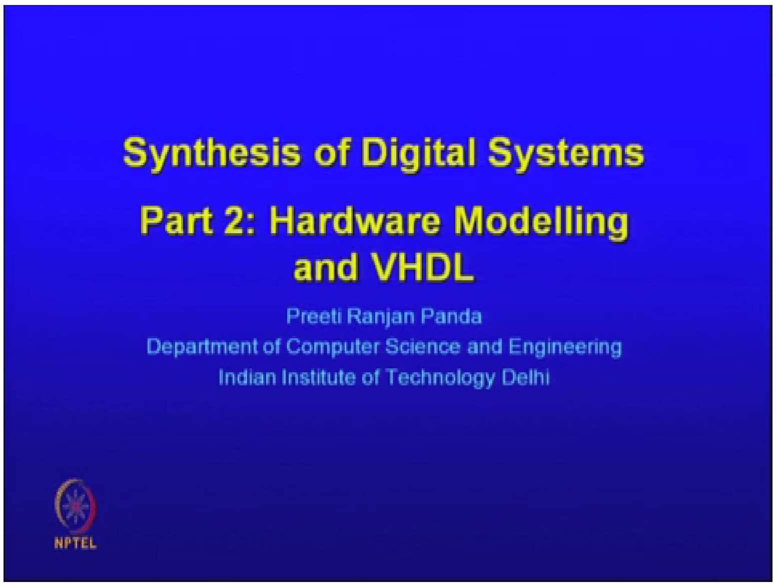
(Refer Slide Time: 43:25)



So, we indicated that there is a behavioral synthesis and there is an RTL synthesis flow these are different just because the levels of abstraction are different the components, the objects the properties and all of these are different in some ways of these components. Therefore, the kind of activity that is involved in those synthesis steps would be different, but that is of course, the topic of this of this course. We will talk about behavioral synthesis, but we will also talk about RTL synthesis, logic synthesis and so on. It turns out that the formulation of the problems they care about and the solution mechanisms are all very different from each other.

With that introduction let us move forward to the modelling part we did say that that was the first step in the design flow before going into the modelling.

(Refer Slide Time: 44:24)



Synthesis of Digital Systems

Part 2: Hardware Modelling and VHDL

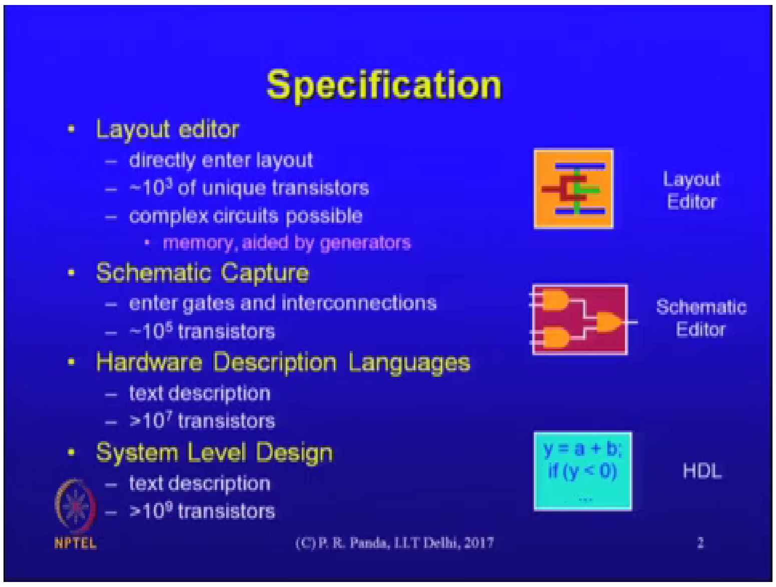
Preeti Ranjan Panda
Department of Computer Science and Engineering
Indian Institute of Technology Delhi

NPTEL

So, hopefully we got an idea about where design automation is relevant and of course, the conclusion is that you break down the design flow into a very large number of steps. There is automation that is involved at every one of those steps we will be concentrating on that synthesis step here, but some motivation has been set for design automation at every one of those steps. So, the first step was the modelling of the system. And let us now, take closer look at what is involved in the hardware modeling.

Of course, we will rely on our prior knowledge of programming languages and standard programming constructs are assumed to be already known we will. What will point out here by way of hardware modelling are those features that are different in some ways in a hardware modeling. There are lots of things that are common and there should not be any need to talk too much about the common features.

(Refer Slide Time: 45:34)



Specification

- Layout editor
  - directly enter layout
  - ~10^3 of unique transistors
  - complex circuits possible
    - memory, aided by generators
- Schematic Capture
  - enter gates and interconnections
  - ~10^5 transistors
- Hardware Description Languages
  - text description
  - >10^7 transistors
- System Level Design
  - text description
  - >10^9 transistors

Layout Editor
Schematic Editor
y = a + b;
if (y < 0)
...
HDL

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 2

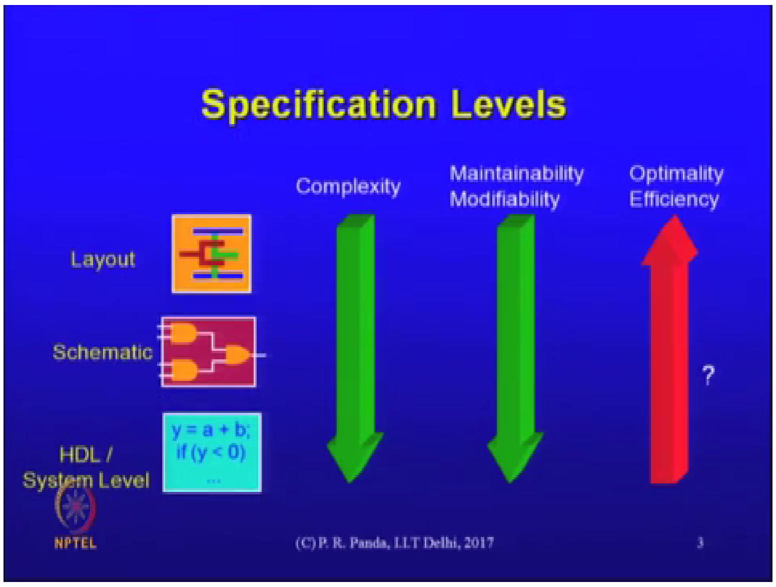
How do you specify your designs? We already talked about some of those capture related tools that are involved it could be that we specify using a layout editor, which means that these individual geometries, individual wires and transistors all of them are actually drawn by us manually on an editor.

You could do that it limits of course, the complexity of the system that we can design in some ways. So, maybe it translates to a small number of unique transistors that could be designed in this way. That does not mean that complex designs are not possible memories are actually very large circuits and individual memory cells, individual decoders, IO and so on those are hand designed typically right. That is an example of a chip and a large system where you actually may be entering this, the design at a layout level. So, not that complex designs are not possible, but it is possible when the system that is being designed is regular enough that you can use generators in some automatic way. So, essentially the memory system is an example where you would be designing maybe one cell by hand, but it is designed in a way that you should be able to replicate that design millions of times and you still have a working system. So, but the complexity of the individual blocks that we design is still limited there.

You could move to schematic capture where you enter gates and interconnections and so on. So, the difference between that layout at this schematic and that other schematic here is that this is actually representing the layout is representing geometry whereas, this is still a little abstract. You draw a wire like that you do not really intend that the wire should look that way in the layout, it is just a logical view of the system. So, this is the way older systems used to be designed and it is possible to have the medium complexity designs being captured in this way. The more modern mechanism for specification capture is typically through hardware description languages it is a text language and usually the specification very nicely scales up to up to very large systems.

So, as you can see here that syntax is something that is similar to what we have already been used to in the form of in the form of programming languages. So, description is text and you could enter very large systems this way system level of design that the difference here being that some of the components are software components and so on there too it is not very different the entry could still be text. You could still have sophisticated graphical tools that aid the design entry in various ways, but overall the standard ways of capturing specification at the higher levels of abstraction like HDL or system level is typically through text.

(Refer Slide Time: 48:54)



There are some interesting questions first to answer before we go into capturing specification. We said that the complexity as you go down or you go up the level of abstraction or layout of schematic and an HDL and system level, you can expect the complexity of the system that can be described using those specification capture mechanisms to increase as you go up the level of abstraction, up from layout capture all the way to text level HDLs.

There are other advantages to doing it this way what about maintain ability or modify ability of the designs. So, somehow that is related to the number of objects that you are entering. So, that number is small then it is easier to maintain, the system modify, the system understand the system. So, that too would improve along with this transition of the specification capture level.

There is an associated and an interesting question what about optimality, what about efficiency of the design being constructed. Would there be circumstances where it is more efficient to enter the design at the layout level than at a gate level or at an HDL level? Yes, why?

Student: (Refer Time: 50:21) then we go for the gate level layouts it starts from there and then.

Is there a context in which it makes sense to enter the design at the layout level?

Student: (Refer Time: 50:33).

Yes. In fact, we already talked about. That memory is an example where you actually enter the sense of a memory cell through a layout. It is so important that you can afford to spend all that time to design the system at the layout level.

Student: If we want to optimize the area beyond limits of the technology then you would (Refer Time: 51:01) like memories.

Right.

Student: (Refer Time: 51:04) based highly optimized.

Yes, it does depend on the context the answer is not. So, obvious which one is more efficient, it is not obvious. So, a modern, high performance processor if the next generation processor if you are designing what level of abstraction would you be designing it out of this?

Student: HDL level.

At the HDL level. Would it make sense to capture parts of it in layout?

Student: Yes.

It might. So, the circumstance of our particular design might be such that what do you mean by circumstance; maybe there are timing constraints imposed on some subsystems that are not realizable if you enter it at the HDL level.

Of course, the remember that they are when we say HDL captured system level and so on we did say that we are talking about digital systems here whereas, there might be components that are analog and so on in a bigger system which you do need to take recourse to other mechanisms of the specification. But there is one thing to be acknowledged about moving to higher levels of abstraction, moving to higher levels of abstraction somehow the assumption is that we will use automatic tools to take us down into the layout. Ultimately of course, we do need a layout, but if I capture it at the HDL level, capture the specification at the HDL level then I will use some automatic tools to take me down to layout.

If I were to capture it at the layout level then I am not using that kind of automation there are other automation that is involved, but I am directly entering the design at the layout level, so what can we say about the comparison of the efficiency which one would be more efficient. If I had the choice of entering a design whatever it is let us say it is a multiplexer, if I had the choice of designing it at the HDL level there are some constructs that can help me specify a multiplexer in an HDL. I also know how to design, I also know the transistor circuit and I can actually enter it that transistor circuit at the layout which would be more efficient?

Student: Efficient in terms of (Refer Time: 53:22).

Yes. So, let us define the metric of efficiency area.

Student: Layout (Refer Time: 53:29).

Layout should be more efficient why?

Student: So, that these are the exact picture of how everything could be routered and placed on a particular chip (Refer Time: 53:40).

So, let us move up from a multiplexer to a little bit more complex let us say it is some small finite state machine that you decide or you can think in terms of equivalent number of gates maybe there are 10 gates. So, is that better to, do you think that the design will be more efficient if you directly enter it at the layout level area we are talking about as the efficiency metric then if you were to specify it there at the HDL level.

Student: Since layout would be custom made (Refer Time: 54:11).

Yeah.

Student: (Refer Time: 54:12), but HDL the automated processed is more generalized.

Right.

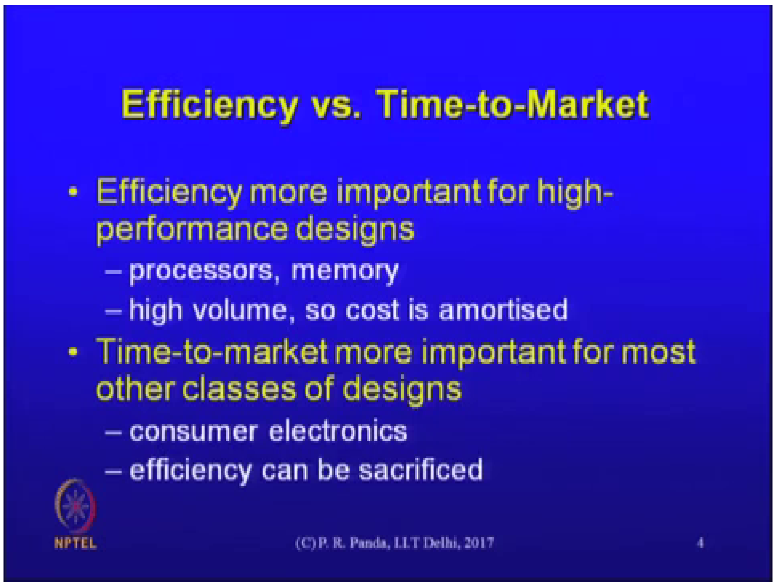
Student: So, layout should be more efficient.

So, there is an argument that if you draw that layout yourself if you get to control everything right all the placement of the individual transistors, the connections between the transistors you expect that you can do better than a tool that ultimately has the same logic in its mind it has the same transistor diagram in the mind, but maybe it is its realizing it differently from the way you would realize it. So, there is an expectation that we can design a more efficient system if we were to capture it at the lower level of abstraction at the layout level.

Student: (Refer Time: 55:04).

Yeah, it is yeah. So, while that expectation is there is an implicit assumption in that expectation, implicit assumption is that it that design is small enough that you have full understanding of it therefore, if it is the 10 gates maybe it is still we could possibly generate a more efficient system. It is not clear that that argument scales if the design is very large it might be hopeless to design the layout. Given enough time maybe you could still do it, but unfortunately that much time is not there the chip ultimately has to go into the market right. So, there are limitations with respect to how much time can be spent therefore, the design methodologies are usually such that there is a trade of that is involved in the efficiency versus time to market, that always effects a product design and it affects the important decisions of what level of abstraction I should capture my specification at.

(Refer Slide Time: 55:57)



Efficiency vs. Time-to-Market

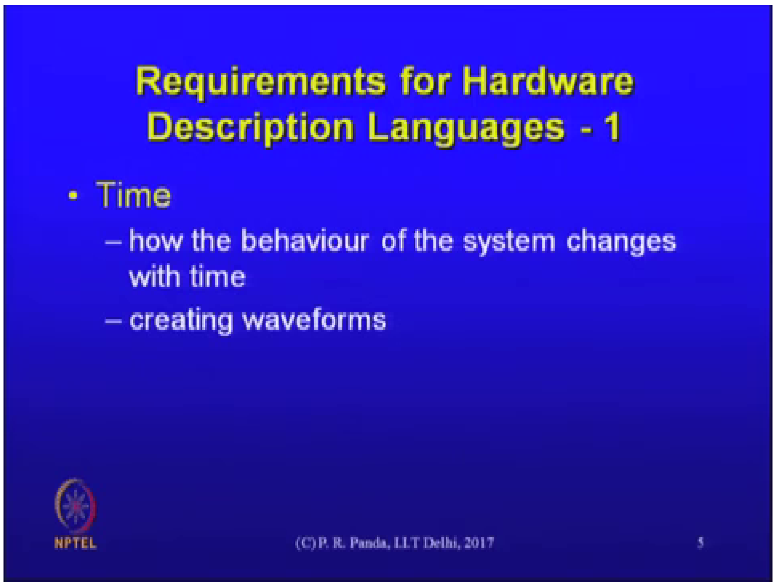
- Efficiency more important for high-performance designs
  - processors, memory
  - high volume, so cost is amortised
- Time-to-market more important for most other classes of designs
  - consumer electronics
  - efficiency can be sacrificed

NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 4

That efficiency is typically more for a class of designs like a high performance processors and memory and so on where the area is very important to, you memory is an example where you want to design that single cell in a very efficient way right. You would you cannot afford to lose even a little bit of space because that single cell is going to be replicated so many times that it makes a difference to the ultimately to the cost of the design itself. So, there would be some class of chips like high end processors memory and so on where you can afford quite a bit of manual intervention in the design. So, that leads to higher costs in a very natural way but typically the expectation of volume is high enough that the cost is amortized over the very high volume that it is spread over.

But otherwise the time to market is typically a lot more important for most other classes of designs involving consumer electronics and so on. There is always an aggressive deadline and that deadline means that you have to make intelligent decisions with respect to how much manual intervention is involved and what the methodology should be and that leads to various decisions being taken for example, you would intervene at a gate level or a layout level only where it is necessary where for example, you cannot obtain that kind of performance or area if you were to use higher levels of abstraction. So, in that process some efficiency can be sacrificed it is alright it is expected that some efficiency would be sacrificed in the interest of meeting time.

(Refer Slide Time: 58:04)



Requirements for Hardware Description Languages - 1

- Time
  - how the behaviour of the system changes with time
  - creating waveforms

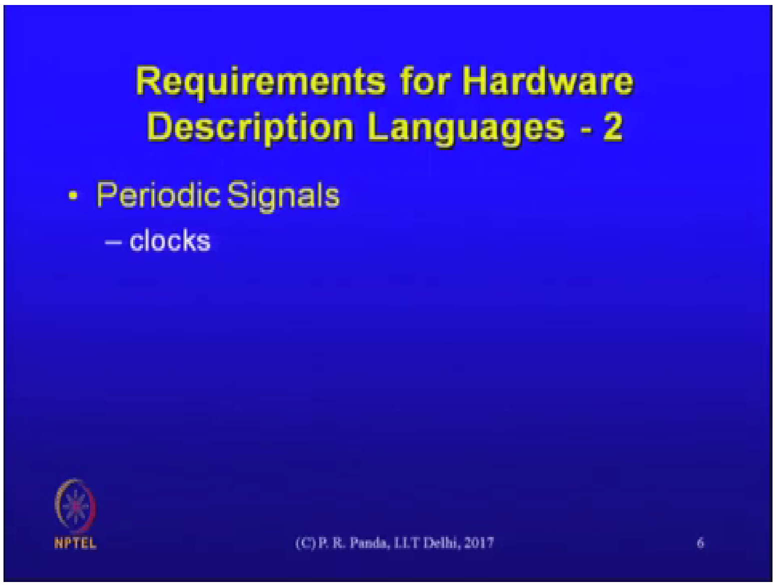
(C) P. R. Panda, I.I.T Delhi, 2017

5

So that just sets the context for hardware description languages as it turns out in modern systems even in high end systems, high performance processors and so on large parts of the system are indeed written in an HDL and a system level description. So, there will be some components that you will still design manually perhaps, but that the overall system just because we need to do other things with it we need to be able to simulate it, verify it and so on the HDL based methodology is very widespread. It would be rare to find an example of a modern chip where HDL was not involved at all. So, let us quickly go through some of the requirements of these HDLs.

In ways that are different from what is the normal expectation and normal features in a typical programming language. So, essentially we need to sit back and think about what aspects of hardware do I really need to capture that are not present in an obvious way in a programming language. What are those? So, let us start off with indication of time the. How the system behaves with respect to the progression of time is a very important component of a hardware specification. So, I need some language related help. So, we are just trying to indicate what are these requirements so that we can provide language constructs to capture those requirements. So, time is essentially I should be able to create waveforms, I should be able to observe waveforms and so on. So, some mechanisms I need for specification of time and responses of the system to time.

(Refer Slide Time: 60:00)



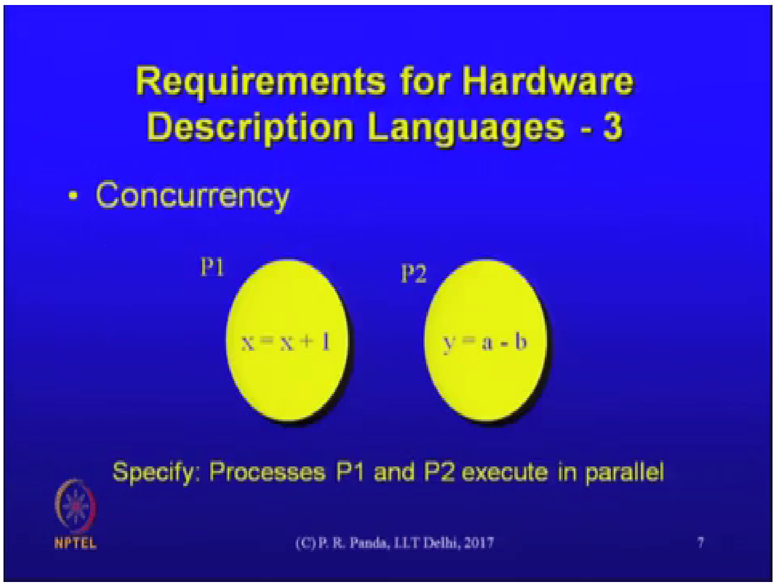
Requirements for Hardware Description Languages - 2

- Periodic Signals
  - clocks

(C) P. R. Panda, I.I.T Delhi, 2017 6

Through that the creation of periodic signals as we know the we need some mechanisms a hardware sequential hardware is described in terms of clock and I should be able to model clocks in my design in some ways.

(Refer Slide Time: 60:16)



Requirements for Hardware Description Languages - 3

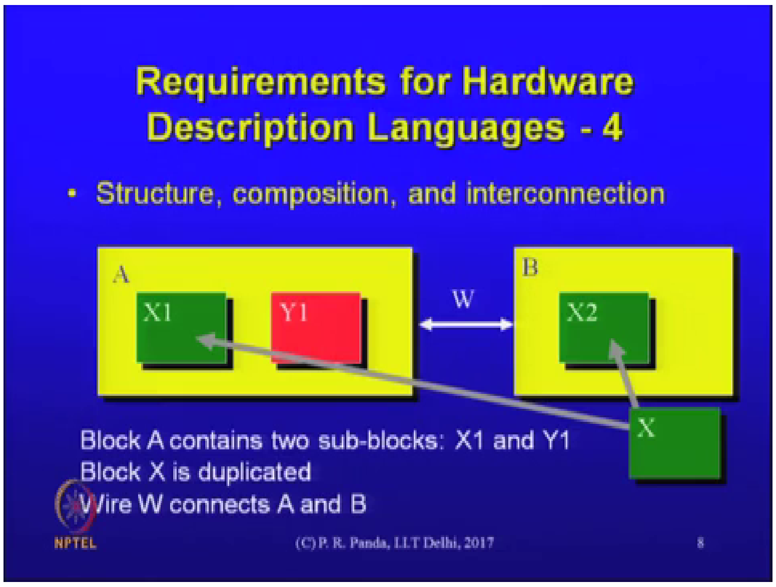
- Concurrency

Specify: Processes P1 and P2 execute in parallel

(C) P. R. Panda, I.I.T Delhi, 2017 7

The idea of concurrency we actually pointed out the other day you have a large net list of gates. In fact, all of them are working simultaneously right; all the gates are working are active at the same time. As long as the inputs are changing the outputs will change if all the inputs change to all the gates in the design then all the gates are active and all outputs may change in response. So, concurrency is very fundamental the way we think about hardware and I should be able to explicitly model that in my design. So, process is just a terminology that is used in these languages to capture that effect, but I should be able to say that there is some operation or a bunch of operations that proceeds concurrently with another bunch of operations. So, two processes P 1 and P 2 execute in parallel I should be able to say that in a simple and elegant way.

(Refer Slide Time: 61:12)



Requirements for Hardware Description Languages - 4

- Structure, composition, and interconnection

Block A contains two sub-blocks: X1 and Y1
Block X is duplicated
Wire W connects A and B

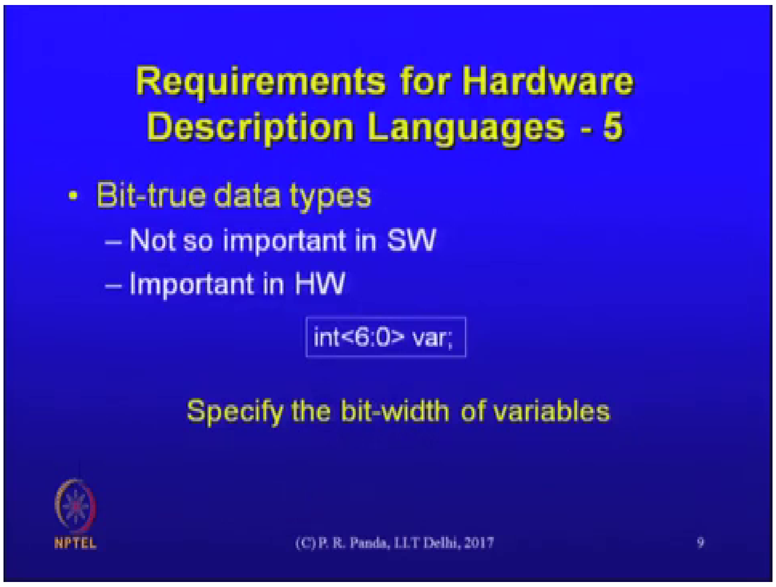
NPTEL (C) P. R. Panda, I.I.T Delhi, 2017 8

A
X1
Y1
W
B
X2
X

I should be able to specify structure composition interconnection these are the fundamentals of a net list creation. We had already seen what their structure would be like. So, what kind of structure I have here the idea that a block A consists internally of two sub blocks X 1 and Y 1, I should be able to specify that. I should be able to say that this X 1 and X 2 are just two instantiations of the same block that is X. That is again something that we all have already been using in whatever hardware we have been designing. So, we do need that ability, we should be able to say among other things that there is this wire that connects these two blocks.

So composition I should be able to specify that this block consists of these other sub blocks I should be able to specify interconnection, these pins of one block are connected to those pins of the other block I should be able to specify that.

(Refer Slide Time: 62:15)



Requirements for Hardware Description Languages - 5

- Bit-true data types
  - Not so important in SW
  - Important in HW

`int<6:0> var;`

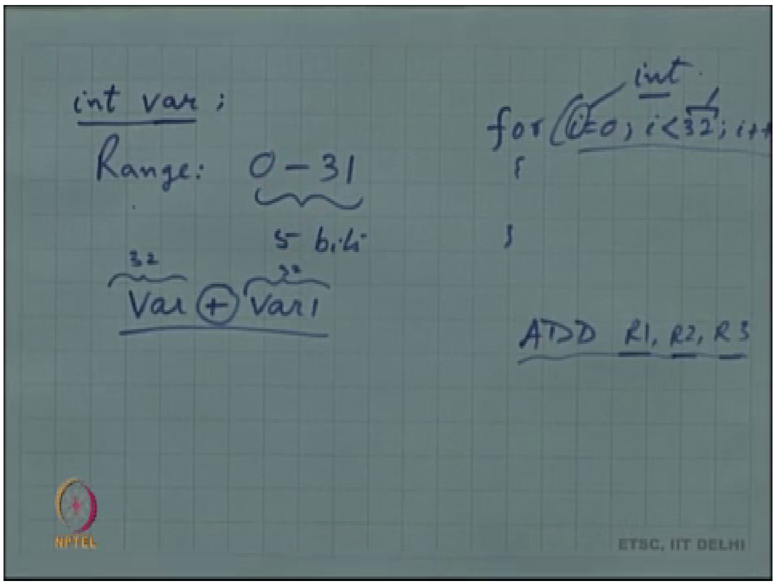
Specify the bit-width of variables

(C) P. R. Panda, I.I.T Delhi, 2017

9

Quickly moving on of course, we will get back to how we specify that. So, as of now we are just trying to indicate some requirements in a hardware specification that we might not necessarily have used in the software. All though will when we talk about these constructs we will realize that some of them do overlap with software constructs too, but let us talk about bit true data types. This refers to the idea of specifying bit width of variables. I say something is an integer, but in addition to that I also have this information that this integer let us say I have an integer called variable.

(Refer Slide Time: 62:55)



I know the range that that variable can take. So, the range is let us say 0 to 31 that is the range over which that integer value might vary, but not beyond that. Remember this is a specification of the hardware and ultimately there is some hardware circuit that will be generated that is inferred from that specification right. So, what I am saying is that for the hardware specification it makes sense for me to somehow indicate that possible range as part of the specification itself, why?

Student: (Refer Time: 63:45).

For using less memory, where is the memory being inferred from?

Student: (Refer Time: 63:51).

Yes. That integer needs to be stored somewhere and if I have this valuable information that the range is 0 to 31 this means that 5 bits.

Student: (Refer Time: 64:03).

Are sufficient for storing that integer, right. As opposed to this if I had just left it in a programming language saying int bar what would the inference be?

Student: (Refer Time: 64:18).

Of storage, actually we do not necessarily indicate that we just say integer, but programming languages make certain assumptions about the width of that integer compilers are obliged to obey the rules of the programming language. So, in C when you say int a 32 bit integer might be assumed. Such a thing might not be a good idea in a hardware description language.

So, let us talk about softwares, I did say that I will talk about some of the parallels between compiler technology and synthesis technology as we discuss these in detail, but it actually shows up even at the early stage of specification.

In a C program if a variable had a similar property how would you declare it? There are bit fields that can be used to actually specify explicitly the range. So, the idea that you are interested in an object that essentially is 5 bits, but do you do that usually? In fact, you could not even recall how you do it, which means that you do not really you are not using it. Why do you not use it is it not important?

Student: (Refer Time: 65:30) it is an important. Sir, memory is not that much of criteria because in computer systems we have large chunks of memories.

Yes, this is an important the memory related argument is important. If we are generating, if we are generating hardware from that specification then the 5 bits as opposed to 32 bits that are going to be used they make a lot of difference. This argument is with respect to memory it is not merely that. So, let us say this is a variable right. Then we have var. Let us say I had some other variable you had an operation on two variables both of which are constrained to be in the range 0 to 31. What is the other inefficiency that creeps in if I did not put this valuable information that my variable was 5 bits wide?

Student: (Refer Time: 66:19).

Yeah. So, both of these need to be stored right. Storage and a default assumption will be made about 32 bits right, in the absence of that specification. But note that what about this addition how will that addition happen? In hardware you have to instantiate an adder circuit to perform the two additions. Now, that addition also will be inefficient, you need a 32 bit adder to add to 32 bit adder, as opposed to 5 bit adder that would have been enough. If I had this valuable information that the possible set of values is restricted to 0 that range of 0 to 31. This is an important optimization element.

Sometimes there may be enough information in your program or in your hardware description that an analysis tool can actually automatically infer that the range is limited to 0 to 31. It could sometimes right. Can you give an example of a code that you write from where we could infer that the range is limited?

Student: 5 to 0 or 1 some of the bits (Refer Time: 67:36).

Yeah. I am just talking about let us say you are in a C program there is something that I could look at the way you have written a loop and infer that 5 bits are enough or you could equivalently infer that the range is limited to 0 to 31. How? The way you write a for loop, this is the only thing that is happening these are the only updates that i plus plus is the only updates that is happening. There is actually enough information in there for us to infer automatically for a tool to infer that the range is 0 to 31 not beyond that. So, such of course, if you write such a loop the synthesis tool could possibly automatically make that inference. Sometimes though, but all the time that information might not be there.

Sometimes it could be that the designer has awareness about the system maybe that value instead of being hard coded like that it comes as an external input somehow. And you know from the external working of the system that the value is limited to 32 bit otherwise the system might not be able to capture it.

So, in general there may be a need for us to specify some language support to the designer where the designer is able to say not merely the type of the data, but also the bit widths. Such a thing is not very important in software, but it is very important in hardware it controls in a direct way the quality of the hardware that is being generated. We already saw that, it controls the sizes of the registers sizes of the memories that will store those variables, but also it has lot of implications depending on what operations you are performing sizes of those ALUs and so on multipliers everything might get influenced by such a, by that knowledge and that knowledge is critical right in hardware.

So, let me stop here, we will continue with what are the other requirements of the HDLs and then we will introduce the HDL language later on.

Student: Sir, what is the meaning of bit true?

Bit true just says that you specify the exact bit width. As opposed to an int that we specify in a programming language so we rely on the compiler to use some default widths bit true means that we are specifying exactly how many bits. It is a lot more essential in hardware. In fact, there is a different reason why you do not use bit fields in the software context you just leave this as int.

Difference is that the architecture of the processors ultimately it is going to run on some processor that architecture is optimized to certain data widths right. So, that data bit might be 32 bits for example, of the processor. In ways that it does not really help you whether your data is the range is 0 to 31 and therefore, whether the bit effective bit widths of your data whether there are 4 or whether they are 20 or whether there are 32. In fact, the processor is designed in a way that it does not really take any advantage of your effective data being narrow. Meaning let us say you go for an add instruction, in a processor.

This is assuming that register has a certain width now right, what is that width? It is that 32 bit. So, the fact that our data is fitting within 5 bits is not really helping the processor is designed in a way that that add instruction may take one clock cycle even if the bit if the operand was 32 bits wide. So, a narrower operand is not really helping us. Specification that the operand is narrow is not really helping us in that context therefore, it is actually correct to use that int data type. That is actually the fastest way. So, you do not get any faster by having 5 bit data in a processor. But in a hardware implementation you do get faster. If you should do get smaller, but you do get faster it is efficient in almost every way whether its area or power or delay or so on. So, it is a lot more critical when you are generating hardware therefore, in the synthesis context the specification the knowledge of the right bit widths is a lot more essential. In the software context because of the way processors are designed it tends to not really matter.

So, let us stop here with this.