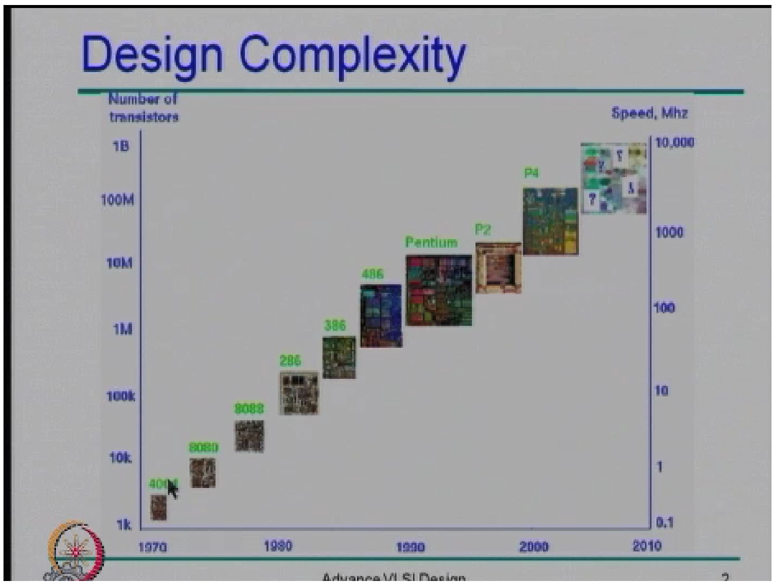
Advanced VLSI Design
Prof. Virendra K. Singh
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture - 39
VLSI Design Verification: An Introduction

Hello. Welcome to the lecture series on Advanced VLSI Design course. I am Virendra Singh from Electrical Engineering Department of IIT, Bombay. I will take you through design verification challenges and these couple of lectures. Today, it is the introductory lecture I will discuss about the challenges of VLSI design verification and in due course of time, I will take you through various challenges, their techniques to handle this and open research programs.

(Refer Slide Time: 01:03)



So as mentioned by Professor Chandorkar very early in this course that VLSI circuits have gone through various phases and if you look at one of the examples like microprocessor which was started early in 70s with 4004 and it has taken a long journey up to now current say Intel i7 processor. So 4004 was fabricated with 2300 transistors whereas now we have billons of transistors on the same chip.

So you can see the growth of the transistors hence complexity of design is increasing dramatically. So the number of transistors are increasing, number of input output are also increasing. Now here the big challenge is how to verify the correctness of the design. If you look at the VLSI design realization flow it starts from the customer needs. Customer needs may be in terms of like one customer may want to have, want to build a chip for microwave oven controller.

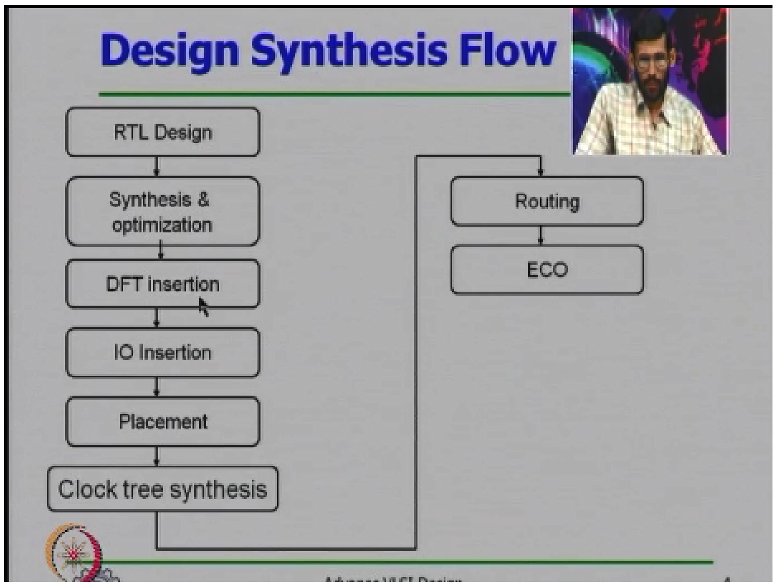
So he can give his requirement like if you put milk in the microwave oven it should operate for 10 minutes and run at 300 watt. If you want to cook rice, then it has to run for say 15 minutes at 350 watts and so on and so forth. So from this design engineer has to figure out what are the various requirements because the need is big in nature. Based on the requirement here the design engineer has to write some specifications which are more formal than the English like language.

Once you write these formal specifications so your requirement may directly come from the algorithm like you want to implement say image processing algorithm then algorithms are non-known and then your specification may be C code. So from that you have to synthesize your circuit so it has to go through various steps of synthesis which are clearly automatic steps. Once you synthesize the circuit, so do the place and route and finally you have GDS2.

You sent that to fab and then fab facility will fabricate your chip and give you the fabricated chip. Now you have to test each and every manufactured chip for that you need to develop some test vector that can test your chip in reasonable time. So these are the various phases. Now here when you are synthesizing your circuit you have specification as your golden reference model and then from the specification you have to go to RT level transformation.

From RT you have to go to gate level netlist from gate level netlist transistor level implementation and then you do the place and route and then finally the layout. So all these transformations have to respect the given specification. So now here what we want that here at every level we have to validate with respect to the laid down specifications.

(Refer Slide Time: 04:42)



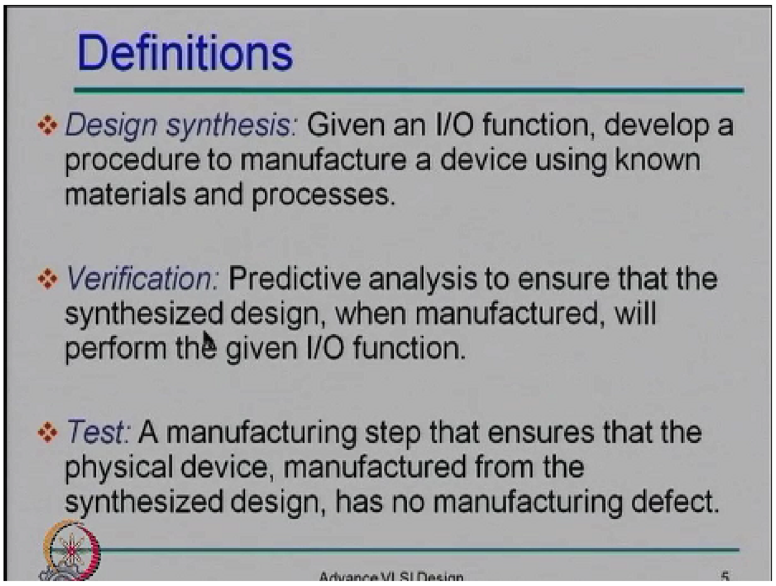
Design Synthesis Flow

$$\begin{tikzcd}
\text{RTL Design} \arrow[d] & \text{Routing} \arrow[d] \\
\text{Synthesis \& optimization} \arrow[d] & \text{ECO} \\
\text{DFT insertion} \arrow[d] & \\
\text{IO Insertion} \arrow[d] & \\
\text{Placement} \arrow[d] & \\
\text{Clock tree synthesis} \arrow[uuuuur] &
\end{tikzcd}$$

Now if you zoom in this process little bit in that case you can see that from specification you generate the RTL design that you are writing VHDL or Verilog. From there you have to synthesize the gate level netlist to optimize for various parameters. Parameters can be area, power, performance or testability. Then once you do that you have to insert the design for test points and you have to augment your circuit in order to make it easier to test after that you have to insert IO.

After doing that you have to do the place and route and then you do the clock tree synthesis. Yeah you do the clock tree synthesis routing and then means after routing if you are not able to still meet your requirement you have to do the ECO electronic change order. So now in this process the bigger challenge is that you have to in all these transformations you have to make sure that they respect specifications now how do we do that.

(Refer Slide Time: 06:00)



Definitions

- *Design synthesis:* Given an I/O function, develop a procedure to manufacture a device using known materials and processes.
- *Verification:* Predictive analysis to ensure that the synthesized design, when manufactured, will perform the given I/O function.
- *Test:* A manufacturing step that ensures that the physical device, manufactured from the synthesized design, has no manufacturing defect.

Advance VLSI Design 5

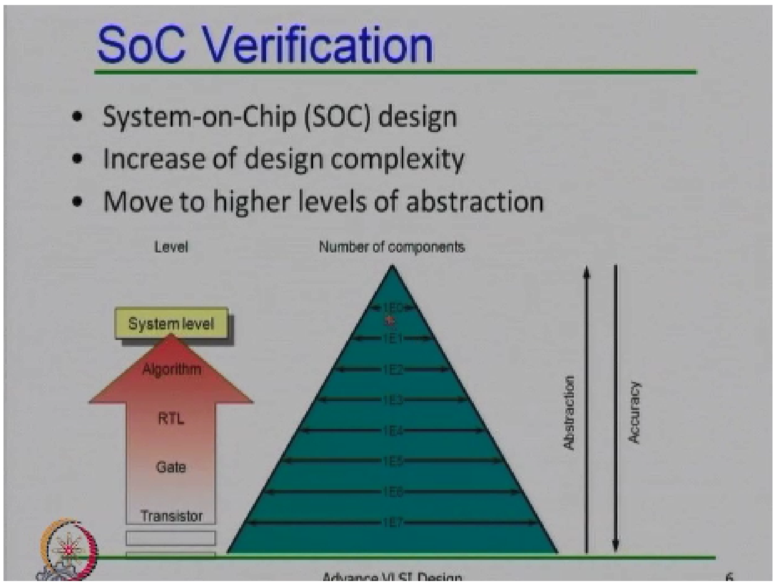
So let us go through first couple of definitions that we do use. The first thing is design synthesis. So design synthesis is a process. Design synthesis process is defined as for a given IO function development of a procedure to manufacture a device using non material and processes. Verifications is defined as predictive analysis to ensure that the synthesized design when it will be manufactured it will perform the given input output function.

The test is defined as manufacturing step that ensure that the physical device which is manufactured from the synthesized design has no manufacturing defect. So design synthesis essentially tells you that how I can obtain a given IO functionality because customer is concerned about the IO functionality.

He is not concerned about how you are implementing that whether you are designing sequential circuit or combinational circuit whether you are implementing that using CMOS or you are implementing using NMOS or you are implementing using TTL. So the customer wants given input output functionality. Then once you have done that you have to analyze that whether your synthesized design respect the specification that those who are laid down for the synthesis.

So that is why it is predictive analysis that can ensure the correctness of the behavior.

(Refer Slide Time: 07:39)



SoC Verification

- System-on-Chip (SOC) design
- Increase of design complexity
- Move to higher levels of abstraction

Level
Number of components
System level
Algorithm
RTL
Gate
Transistor
1E0
1E1
1E2
1E3
1E4
1E5
1E6
1E7
Abstraction
Accuracy

So now if you look at the complexity of the design. Now because design as I mentioned earlier that now current design do have billions of transistors. And that kind of design we cannot design as a flat circuit. We have to have hierarchy and so that means here the various steps. We have to go through this from this system level design to the algorithm like here you have a system that can implement various algorithm.

Like if you are processing image there are various algorithms like segmentation, tracking and so on and so forth. So now for a given algorithm you have to write the RTL from RTL you have to generate the gate level netlist from there you have to have transistor level implementation and then place and route and finally you have layout. So if you look at the complexity this is a pyramidal structure.

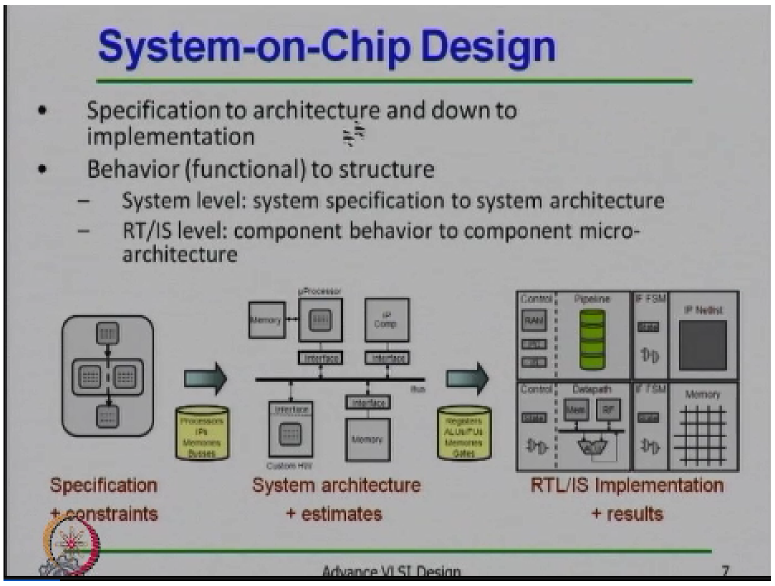
So at system level the complexity maybe few lines of C code. And then if you go to RTL you will have few hundreds or thousands lines of very lower VHDL code then if you go gate level netlist. You may have several millions of gates then if you go to transistor level implementation you may have hundreds of millions of transistors and so on and so forth. So now if you go down your complexity increases at the same time accuracy increases.

What this accuracy mean? Like when we design a circuit we design for given specifications and then here as an engineer we want to optimize our circuit for a few parameters. And those parameters are like area, performance, power and testability. So at every level of abstraction when you do the RTL level design or gate level design you have to estimate how much area it will consume, how much power it will dissipate, what kind of performance I will get.

And how easy it is to test. So now at higher level of abstraction the estimation is very crude.

If you go down and if you do the layout and if you abstract the parasitic then you may have more exact values of the power dissipation, area consumption and performance all these things. Now here if you go down your accuracy increases, but at higher level of abstraction the complexity is smaller you can handle it in better fashion.

(Refer Slide Time: 10:42)



System-on-Chip Design

- Specification to architecture and down to implementation
- Behavior (functional) to structure
  - System level: system specification to system architecture
  - RT/IS level: component behavior to component micro-architecture

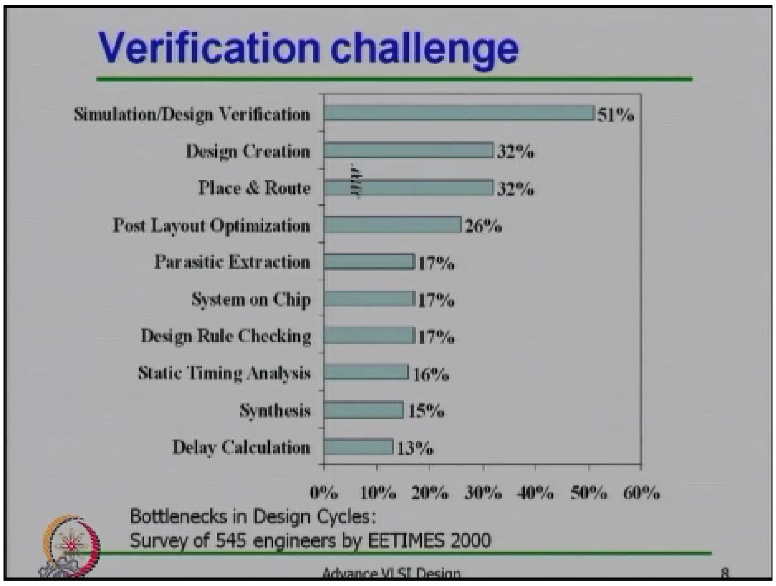
Specification
+ constraints
Processors
IPs
Memories
Busses
µProcessor
Memory
IP Comp
Interface
Bus
Custom HW
System architecture
+ estimates
Registers
ALUs/FUs
Memories
Gates
Control
Pipeline
IF FSM
IP Netlist
RAM
Datapath
Mem
RF
Memory
RTL/IS Implementation
+ results

Advance VLSI Design 7

So now conventionally here we start from the specification and we go down to the implementation. So you have specification those are few lines of C code we have. From specification you have to architect your system that may have like how many you need to have say processors, a DSP or GPU then you need some memory, then you need some blue logic or custom logic.

Now if you go down in each of these block you can write the RTL code for that and the RTL code will tell you that what kind of data path you will have, what kind of controller you will have and how data flow will take place. So this gives you the design flow. Now this is your specification when you are transforming from these specifications to the system architecture to the RTL you have to make sure that always here it has to respect the specification.

(Refer Slide Time: 11:45)


Verification challenge

Bottlenecks in Design Cycles:
Survey of 545 engineers by EETIMES 2000

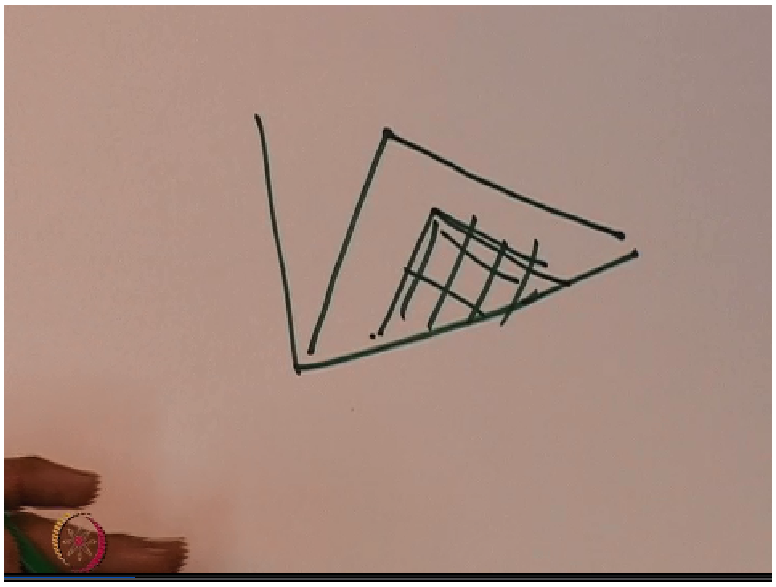
If you look at the time consumed by various activities in the design flow these activities are listed here. These are based on the time survey this is bit old which was done in 2000. If you look at it here you will see that most of the time is consumed by design verification. Then design creation then place and route and then here design rule checking, static timing analysis and so on and so forth.

Though here if you add these numbers it may go beyond 100 because couple of activities are being done in parallel. These are from the old generation designs now in current design the design verification time increase little bit from 50% to 60+%. And then here design creation shrink little bit from 32% to say 25% or like that because now here most of the time we are not designing system from the scratch we use IPs.

So the key observation from this slide is that most of the time we spent for the design verification so that means if this is the critical part in the design flow. Hence we have to have very efficient methodology to verify the design if we want to reduce the design time. And it is reported by couple of industries that if your design cycle escalates by 6 months the total revenue decreases by 30% that is huge.

So that means here the time to market is very, very important and you have to design, manufacture and ship the chip as fast as possible.

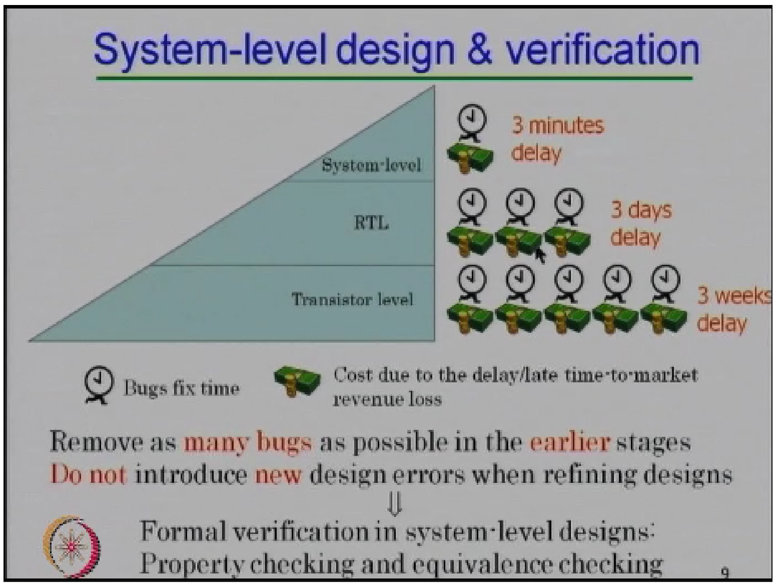
(Refer Slide Time: 13:55)



If you look at the revenue in that case here revenue model goes like this. Initially company gets very high profit margin and then slowly it goes down. If there is the escalation in that case here that will go like this. So now you will have this much revenue that can be earned from the product and that is understandable.

Because here initially your new products have an edge in the market then slowly your competitor will also launch the similar kind of functionality in the market and then you have to reduce the cost. So then profit margin decreases. As I said that key thing is we are spending too much time in design verification you have to reduce that time. So how I can reduce that time.

(Refer Slide Time: 14:50)



System-level design & verification

System-level
RTL
Transistor level
3 minutes delay
3 days delay
3 weeks delay
Bugs fix time
Cost due to the delay/late time-to-market revenue loss

Remove as many bugs as possible in the earlier stages

Do not introduce new design errors when refining designs

⇓

Formal verification in system-level designs: Property checking and equivalence checking

Now if you look at in the pyramidal structure as we discuss that the complexity at higher level of abstraction is lower whereas the accuracy is poor as well. So now if you design a system at system level if you get a bug in that case because the complexity is low you need less time to find that bug or locate that bug and the fixing of that bug. So say here at the system level fixing of a bug takes 3 minutes.

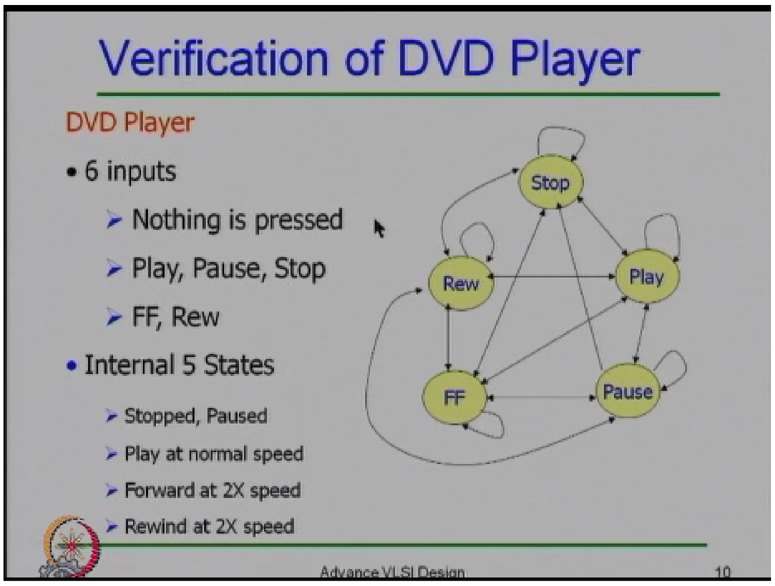
If you go down at RT level where the design complexity increases by order of magnitudes then the location of bug or detection of bug and fixing of bugs takes longer time because the design is more complex and so now here fixing of bug may take 3 days. If you go down further to the transistor level design here, you have millions of transistors. And if you happen to detect a bug the fix localization of that bug and fixing of that bug may take 3 days.

So you can see the kind of time taken at different level of abstraction what it says. It says that we should detect and fix as many bugs as possible at higher level of abstraction possibly at system level or at RT level. And then when we go from higher level of abstraction to the lower level of abstraction we should not introduce new bugs or errors. And this is the key thing in the verification.

So we have to remove as many design bugs as possible at earlier stages and we should not introduce new design errors when we are refining the design. If you are going from the higher level of abstraction to the lower level of abstraction you are introducing or adding more and more information that process is known as refinement. Ideally we should not add or we should add zero error while we are refining though it is very difficult to make sure that there is no error introduced while you are refining the system.

So your formal verification can help you in this and we will discuss what are the formal techniques that can help you in detecting maximum bugs at higher level of abstraction and that make sure that newer error will not be introduced. Now look at how severe this problem is. Let us take a very simple example all of you might have gone through this at some point in time.

(Refer Slide Time: 18:05)



Verification of DVD Player

DVD Player

- 6 inputs
  - Nothing is pressed
  - Play, Pause, Stop
  - FF, Rew
- Internal 5 States
  - Stopped, Paused
  - Play at normal speed
  - Forward at 2X speed
  - Rewind at 2X speed

Stop
Rew
Play
FF
Pause

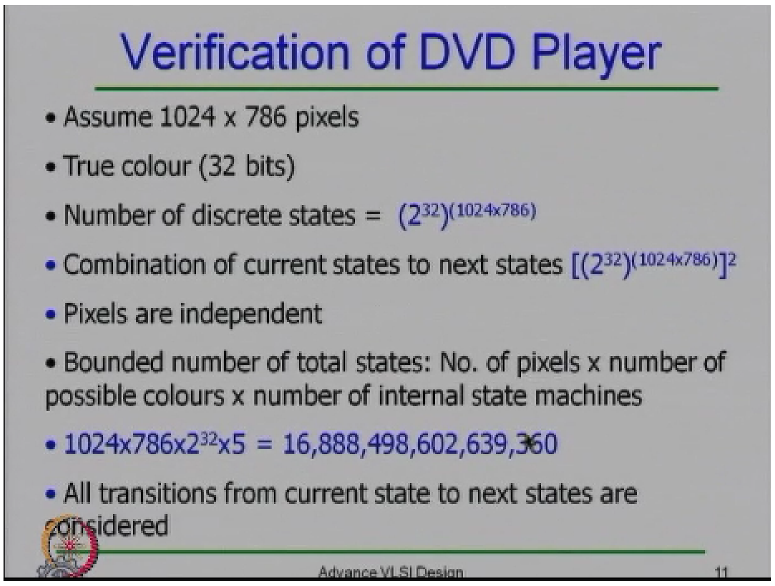
Advance VLSI Design 10

Let us take an example of DVD player which all of you have used say if you have DVD player that can have 6 inputs maybe play, pause, stop, fast forward, rewind and then if you are not pressing any button then it does nothing. Then in order to implement this here we implement a small finite state machine that can have say 5 states. Stop, pause, play at normal speed, fast forward at too excess speed, rewind at too excess speed and the finite state machine can be designed like this.

So wherein you have these 5 states stop, play, pause, fast forward, rewind. If you do not press anything in that case here it will stay in the same state otherwise it will migrate to the new states. So if it is in the stop state and getting play input here it goes to the play state. If you press a pause button it has to go to pause state and again here if you press the play button it can come back to the play state. So this is very small finite state machine.

Now let see how difficult it is to verify this small machine.

(Refer Slide Time: 19:35)



Verification of DVD Player

- Assume 1024 x 786 pixels
- True colour (32 bits)
- Number of discrete states = $(2^{32})^{(1024x786)}$
- Combination of current states to next states $[(2^{32})^{(1024x786)}]^2$
- Pixels are independent
- Bounded number of total states: No. of pixels x number of possible colours x number of internal state machines
- $1024x786x2^{32}x5$ = 16,888,498,602,639,360
- All transitions from current state to next states are considered

Advance VLSI Design 11

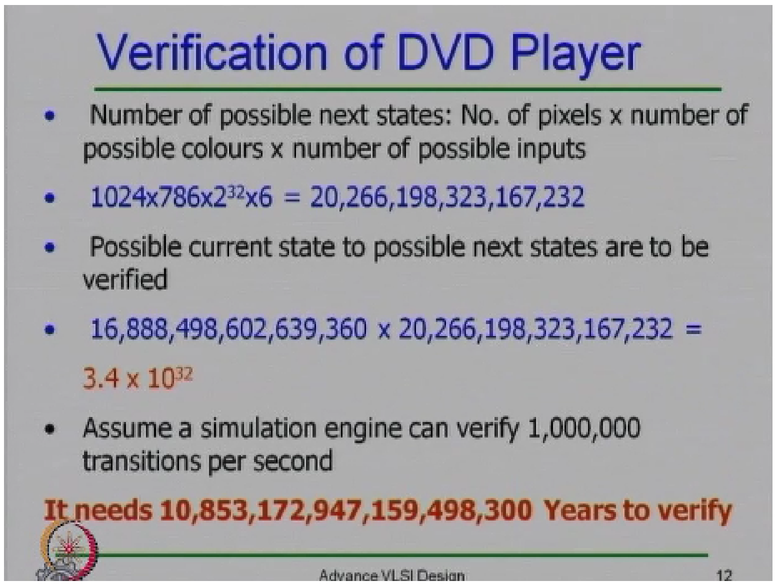
Let say you have a display that has 1024 x 786 pixels. Every pixel is represented by true colors so that means here you can encode that in 32 bits. So now if you want to find out the number of discrete states that you can have would be equal to 2 raise to the power 32, raise to the power 1024 x 786 how we get this because here every pixel can be encoded in 32 bits. So there can be 2 raise to the power 32 states and then here there are 1024 x 786 pixels.

So now here these are the total number of states we can have. In this here we assume that pixels are dependent on each other so that means one pixel can impact others. So now if I look at the state transition in that case. A state transition can be this many number of states to this many number of states. Now the combination of the current state to the next state would be square off this number.

As I said that here we are assuming that these pixels may have dependence. We can fairly assume that pixels are independent to each other that means one pixel do not affect another pixels. So if you assume the independence of the pixel in that case here the total number of states would be equal to the number of pixels into the number of possible color for a pixel and then here the number of internal states.

So 1024 x786 the total number of pixels one pixel is encoded in terms of 2 raise to the power of 32 states and then here there are 5 internal states. So these are the total number of states you can have.

(Refer Slide Time: 21:38)



Verification of DVD Player

- Number of possible next states: No. of pixels x number of possible colours x number of possible inputs
- $1024 \times 786 \times 2^{32} \times 6$ = 20,266,198,323,167,232
- Possible current state to possible next states are to be verified
- 16,888,498,602,639,360 x 20,266,198,323,167,232 = 3.4×10^{32}
- Assume a simulation engine can verify 1,000,000 transitions per second

It needs 10,853,172,947,159,498,300 Years to verify

Advance VLSI Design 12

Now if you press one button it can go to another state so now here the total number of state transitions would be equal to the number of pixels into the number of possible colors into number of possible inputs that you may get. So now 1024 x786 x 2 raise to the power 32 x 6. So these are the total next state. So now here what would be the total number of transitions I can have.

So total number of transitions would be because you can go from any current state to any next state these are the next states and these are the number of current states. So current states multiplied by the next state would be the total number of state transitions we can have and we have to verify this system for all the state transitions that the number of state transitions would be 3.4 x 10 raise to the power 32 this is huge number.

I assume that we can verify one million transitions per second using very fast simulation tool. In that case here it may take several trillion years to verify this design. So that means whatever design we created today that would be ready to manufacture after several trillion centuries which is impractical. So now means if you want to exhaustively verify your design you need this many years that is impractical.

What we want is this should be verified in reasonable time and reasonable time cannot be centuries. So what could be the reasonable time? If you look at the design cycle time it is upon somewhere from 6 months to 2 years or 3 years and assume that 60% time goes to the verification. So in that case here it can be we say few months to year. So now here your reasonable time is few months to year.

Now an exhaustive verification time is trillions of centuries. So you have to reduce this time from trillions of centuries to few months. It is a huge reduction. Keep in mind that we want similar kind of confidence in our design. So that means the kind of confidence we can have by applying exhaustive simulation vectors we should get from a limited number of simulations vectors this makes it very complex.

Now here I guess this gives you a flavor how complex the design verification process is, how many order of magnitude verification time we have to cut down. If I recall the statement by Intel India at VLSI design 2011. He said that today it is the design verification or validation engineer who is most important person in the entire design flow. He said that it would be the design verification engineer who would be able to buy some real estate in metropolitan cities.

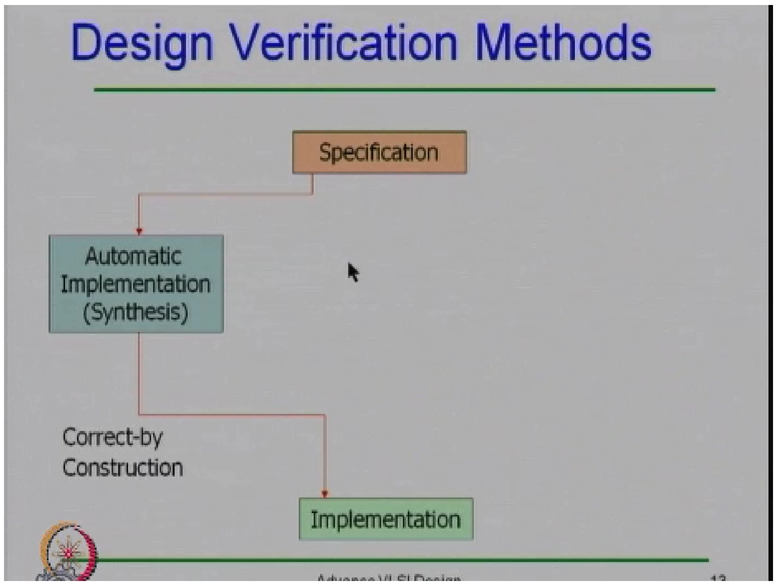
That means here he is the person who would be earning the most money and because now as I said that our current design process is IP based. So now here we are integrating more and more IPs which is making your design more complex and then here it is very, very important to visualize the corner cases and now here I will come to a point what are the corner cases how important it is to visualize the corner cases.

Now here let us start how your design verification flow goes. So you have specification those are created from the customer’s requirement and you want to implement your system or circuit and this implementation should respect the specification so that means there should be equivalence between your specification and implementation how I can do that. I mentioned that there are couple of synthesis steps.

It has to go from specification to implementation. So like RT level synthesis then gate level synthesis, transition level synthesis then place and route and finally you get GDS2. Now here industry want this automatic process. So they want that this should be push button so that means you feed specification and then here it should produce the design.

Now one of the ways that whatever transformation you are doing to gate level implementation, RT level implementation, from specification and then from RT to gate level, gate level to transistor level. In all these transitions if you can make sure that these transformations are correct you do not need to verify this.

(Refer Slide Time: 28:02)



Design Verification Methods

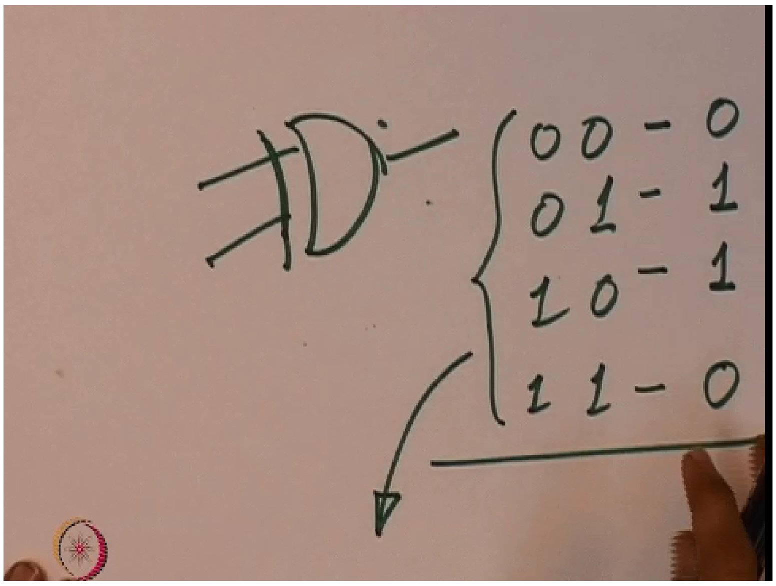
So if you believe that your automatic implementation or synthesis process is correct you do not need to verify every design. You need to verify only once implementation of synthesis tools and so this process is correct by construction. So now here I can completely eliminate the cost of design verification which consume lot of time that is in terms of man hours, it contributes to 60%, 70%.

This is anyway beautiful way. Now what are problems? Problems are as follows. The verification process of a software piece of code is even harder problem than hardware verification. Why because the design space in software is much bigger and complex then hardware. In hardware you write the bit vector from 0 to 4, 0 to 5 means you need to use as many bits as you need.

But in software piece of code you instance it one variable say int i, this explores a design space of 2 raise to the power 32 because this I can take any value and now here the verification of the entire software is extremely difficult or rather I can say that it is next to impossible. Hence you cannot rely on the tools that you are using for the synthesis. Hence you need to verify each and every created design.

If you can make sure that this synthesis process is correct you can completely eliminate the design verification cost. So now what are the options if this option is not available. One of the options is you have to simulate your design and what the simulation mean?

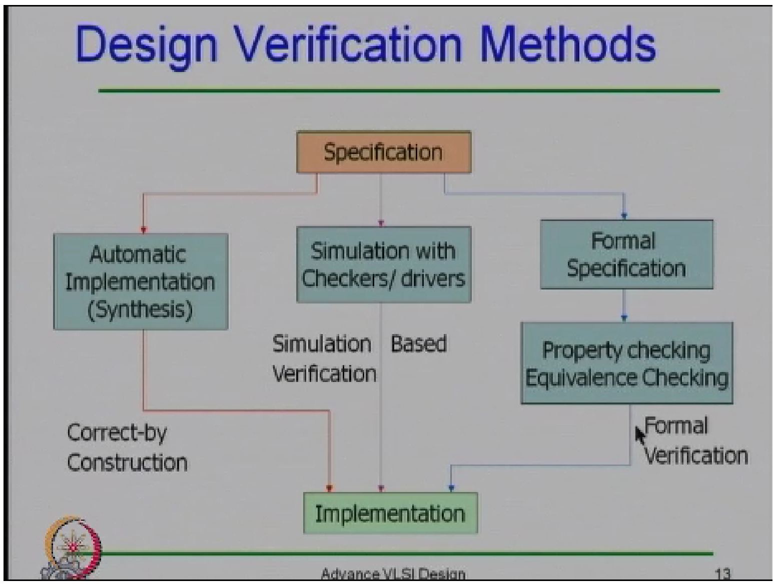
(Refer Slide Time: 20:26)



Simulation I can simply say that you can say you implement one XOR gate what you can do is say there are several possible inputs so if it is two input XOR gate input can be 00,01, 10, 11. So you can apply this couple of inputs and then see whether you are getting the correct behavior or not. So say from 00, 00 you should get 0 for 01 it should be 1, for 1 it should be 1 and for 11 it should be 0.

So now here one way is that you can exhaustively simulate this and as I said that exhaustive simulation is not possible. So now here what you do is you have to take a subset of this exhaustive input pattern and stimulate for that and based on that here you make a decision whether your design is correct or not.

(Refer Slide Time: 31:33)



Design Verification Methods

Specification
Automatic Implementation (Synthesis)
Simulation with Checkers/ drivers
Formal Specification
Simulation Based Verification
Property checking Equivalence Checking
Correct-by Construction
Formal Verification
Implementation
Advance VLSI Design
13

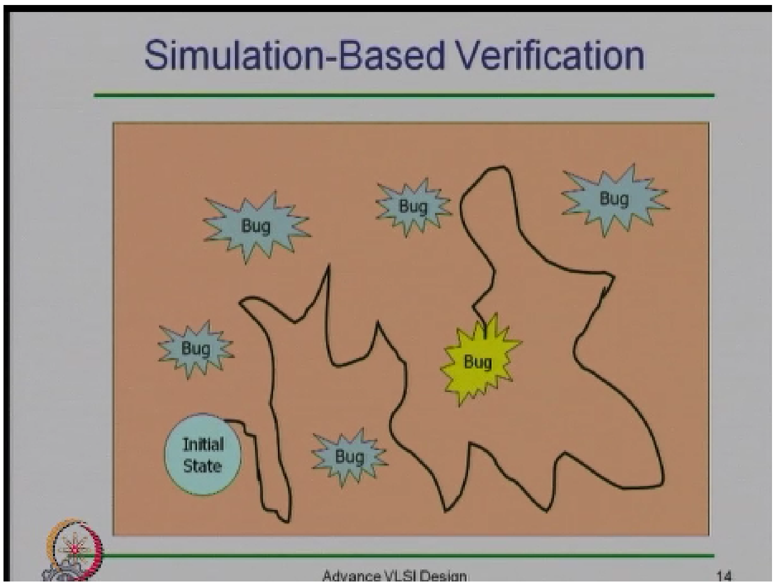
So for this you have to built some checkers and drivers this is called as simulation based verification because here we are not exhaustively simulating we cannot have the 100% confidence in our design and it is very time consuming. So now here again this is not the complete method because we cannot simulate exhaustively. So then here what are the other alternative. Other alternative is we can use mathematics because ultimately you are going to built a circuit which follows the Boolean Algebra.

So this is you have mathematical expression for the function that you want to implement. So now here if you can specify or you can write the specifications in terms of mathematical formula so let say we call formal specifications how you can write that I will come to that point. And then you can reason about that whether your implementation always respect the specification or not and this is known as formal verification technique.

So in hardware verification here the simulation based verification is referred as simulation based verification and this formal technique is referred as formal verification whereas in software verification domain the simulation based verification is referred as software testing. And formal verification is referred as software verification. So sometimes these terms are confusing when you talk with the software people.

Okay let us look at little bit more about what are the challenges we have with the simulations based verifications and what are the challenges we have in front of formal techniques.

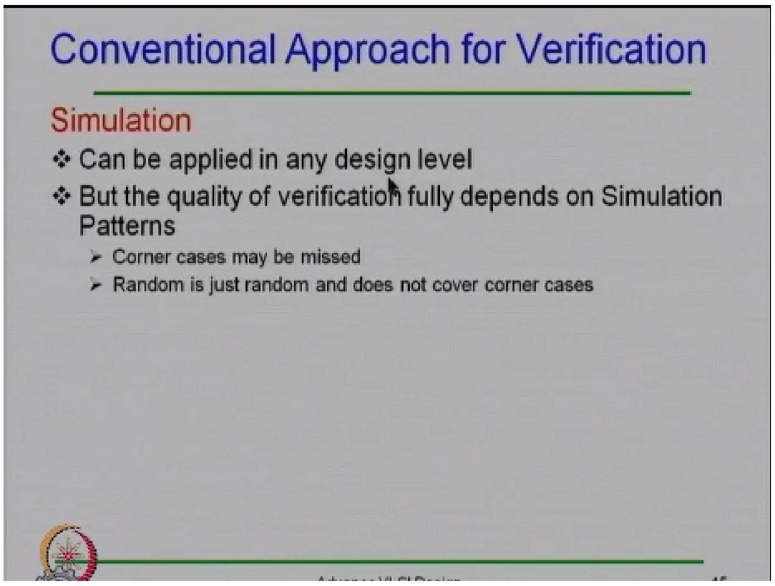
(Refer Slide Time: 33:48)


Simulation-Based Verification

So simulation based verification as I said that we cannot exhaustively test so now here this is your total design space and then there are couple of bugs in your design. So you start from say some initial state and then you start to traverse your design in some way and you hit a bug. Once you hit a bug then you localize that bug, fix that bug and then again you start your simulation based verification and then you may hit another bug and this way you keep on doing.

So this way here if you happen to hit a bug you can say that there is a bug otherwise you believe just that there is no bug, but it does not give a guarantee because you did not explore the entire space.

(Refer Slide Time: 34:38)



Conventional Approach for Verification

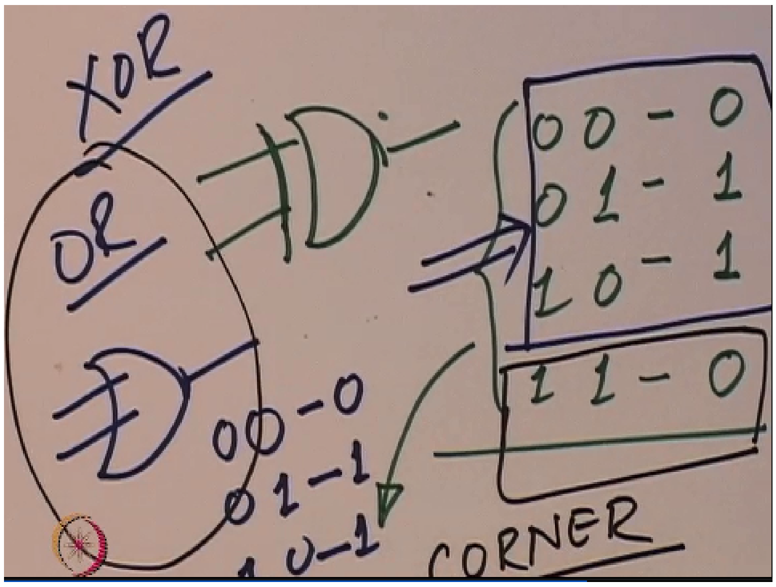
Simulation

- ❖ Can be applied in any design level
- ❖ But the quality of verification fully depends on Simulation Patterns
  - ➢ Corner cases may be missed
  - ➢ Random is just random and does not cover corner cases

So now here the simulation based verification so there are couple of good things about the simulation based verification. One thing is this can be applied across the design level so that means at system level, at RT level, at gate level, at transistor level, at any level of abstraction you can apply this technique, but as I said that simulation based verification cannot simulate exhaustively the entire design.

If you cannot then what are the problems? As I said that you have to pick up the subset of the exhaustive set.

(Refer Slide Time: 35:26)



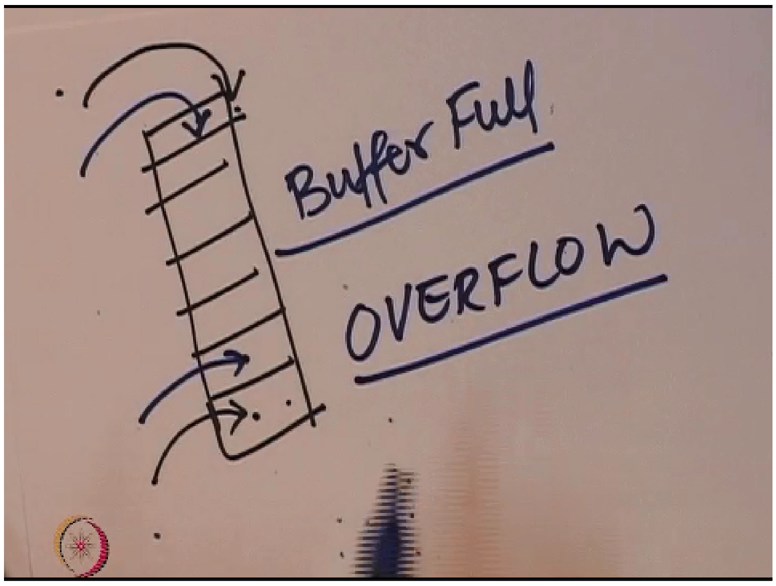
Now here say we were talking about this XOR gate. So say here these are the four possibilities 00, 01, 01, 10, 11. Now if I pick say the 75% of cases that is still a big numbers. Say as I said in DVD player it takes several centuries. If you take 75% cases still it will take several centuries. So even 75% is very big number you cannot pick that. Assume that we pick 75% cases. These are the 75% cases I am picking.

Now assume that by mistake in place of writing XOR I have written only OR and then here it implemented OR Gate. Now when it implements OR gate I look at that and I take these three cases 00, 01, 10. This will give me 0, this will give me 1, this will give me 1 and there is a exact match with XOR gate. Hence, though I have implemented OR gate I will say that XOR gate is implemented correctly.

So this 75% cases are not sufficient. When I pick these cases, I have to pick in such a way that here we can distinguish these. So now here what can distinguish the OR gate and XOR gate this input. So that means here when you are picking a fewer cases you have to at least pick this case. This is the case when you have OR gate. This may have different kind of matching with the different kind of gates.

So these we are saying corner cases. So it is very, very important to visualize the corner cases and that is why the people say that now we in the industry we need the most intelligent people in the verification so that they can visualize the corner cases.

(Refer Slide Time: 37:45)



And other examples of corner cases I tell you say you want to implement FIFO. Say you have 8 entries in the FIFO. So now you are storing some value in the first entry and then you store some value in the second entry so now here how I verify that. I will write and I will read that and if I get the same value in that case it is verified. Now when you are writing to the first location or second location or third location here it will behave in the same way.

So once you check for the first location means it will behave for the third, fourth, fifth. When it come to the eighth location what will happen. Once you have written in eighth location it should generate one specific signal that is buffer full. So that means here after writing eighth location you have to check additionally whether buffer full signal is generated or not that is one of the corner cases because that is different from other cases.

Now even if generate the buffer full signal when you want to write one more value so the ninth value it may possible that here this ninth value what it should say when you are trying to write the ninth value. It should say that buffer is full you cannot write. So that means here it should generate overflow signal. So that means here when you are trying to write ninth value it should generate the overflow signal this is another corner case.

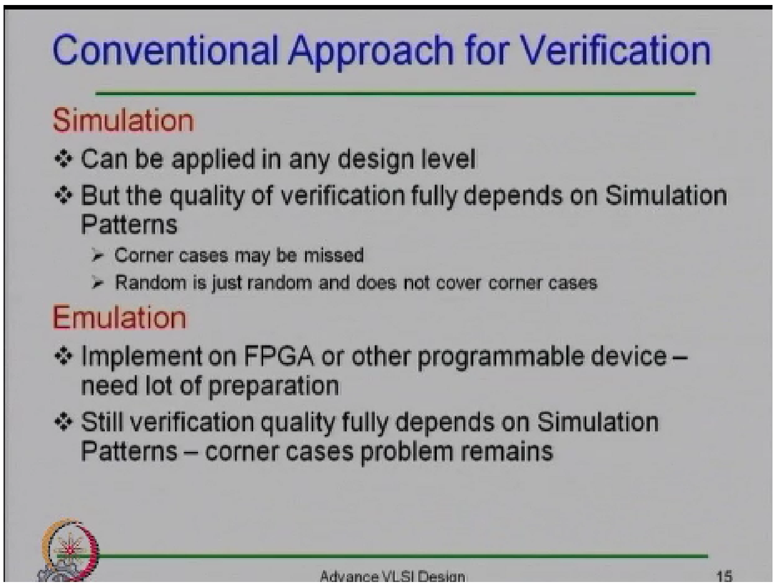
It may so happen that even if it generates the overflow signals, but this can also rewrite that location. So that means here the eighth value is rewritten by the ninth value and you have spoiled your earlier written value. So you have to check that. So that means here whether it has overwritten that value or it still continue to store the eighth value these are the corner cases. So when you use the simulation based verification you have to visualize these kinds of corner cases. So that is very, very critical.

Now here as I said that here most of the time we pick up some of the random values and then we simulate for those values and then we top of this with some of the corner cases. So again here these random values are random and this does not cover all the corner cases and we cannot visualize all the corner cases. Other problem with the simulation based verification is the simulation speed. Can you think of how fast or how slow the simulation process is.

If you look at the simulation process flow whether it uses the compiled code simulation or event driven simulation. The simulation speed is something 1 to 2 hertz. That is very, very slow. If you compare with the actual device speed. Device can run at fairly high speed say gigahertz. Now your simulation runs 8 to 9 order of magnitude slower than the actual device and now you can compute that even if I run my simulation for 6 months how many vectors I can apply to this design.

In order to expedite that the other method which is being exercised in the industry that is emulation.

(Refer Slide Time: 42:00)



Conventional Approach for Verification

Simulation

- ❖ Can be applied in any design level
- ❖ But the quality of verification fully depends on Simulation Patterns
  - ➢ Corner cases may be missed
  - ➢ Random is just random and does not cover corner cases

Emulation

- ❖ Implement on FPGA or other programmable device – need lot of preparation
- ❖ Still verification quality fully depends on Simulation Patterns – corner cases problem remains

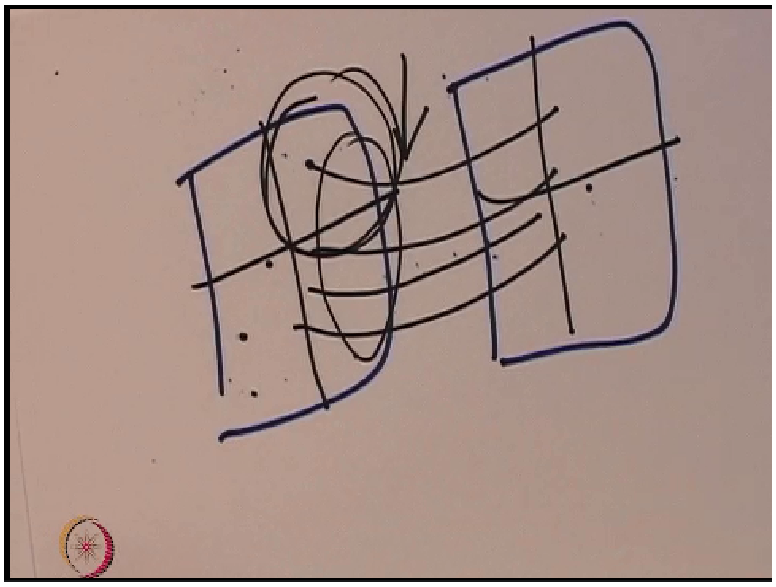
Advance VLSI Design 15

In emulation we try to implement our design on reconfigurable fabric and FPGA is one of the reconfigurable fabric. So now we implement our design on FPGA. Now FPGA can run at much higher speed like 100 megahertz or so which is significantly faster than your simulation on general purpose processor. This looks very interesting and very fast may be say 2 order of magnitude slower than your real chip still it is 5 to 6 order of magnitude faster than the simulation.

Again your exhaustive simulation needs several centuries. So 5 to 6 order of magnitude speed up is not sufficient to go for exhaustive simulation so still it is based on the corner cases how good you are in visualizing those corner cases, but I comfortably said that you implement entire design on FPGA and this can work say 200 megahertz to 200 megahertz. Now what are the challenges can I do that.

If you have small design you can do that, but now say you want to verify the current processor you design new processor that you want to verify. So entire design you cannot implement on the single FPGA. So you need to divide.

(Refer Slide Time: 43:46)



If you need to divide the design so now here say I divide design in two blocks say 100 million transistors or gates I am implementing on this, 100 million gates I am implementing on this. Earlier when I was fitting everything in one here I need the IO pins is equal to the IO pins of your design. Now when I split this there are couple of internal signals which are crossing from one partition to another partition.

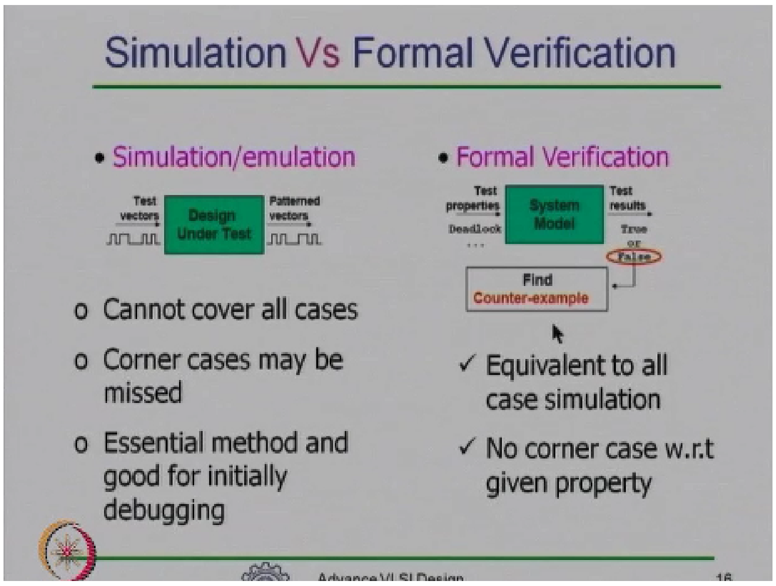
And as you know that there are large number of internal signals and now here these signals may across to millions or several 100 thousands. You do not have 100, 1000 IO pins. So in order to do that what you need to do. You need to again further divide this you need to again further divide this and now here you are limited. So even though you have large numbers of reconfigurable devices available on your FPGA you cannot make use of that because of IO pins.

And now your simulation or emulation speed is determined by the IO speed that is much slower in some kilohertz 100s of kilohertz that makes it 2, 3 order of magnitude slower than a single FPGA. So now here the real emulation speed that you can achieve could be 100s of kilohertz. That is still 4, 5 order of magnitude faster than the simulation, but 4, 5 order of magnitude slower than the real chip.

Again I said that here this emulation you cannot apply exhaustive simulation vectors so you have to rely on the corner cases which are visualized and you know famous Pentium bug which was reported by Intel when they were dividing 2 numbers there was a change or inaccuracy in 8th or 9th decimal point and it did cost about 500 million dollars to Intel. So when I mention the emulation Intel heavily uses emulation for their design verification.

And now they use several hundreds of boards. So FPGA boards to verify one design. So now here this is your simulation or emulation is not enough so you have to go for formal techniques, how formal techniques behave.

(Refer Slide Time: 46:57)



Simulation Vs Formal Verification

- Simulation/emulation

Test vectors
Design Under Test
Patterned vectors

  - o Cannot cover all cases
  - o Corner cases may be missed
  - o Essential method and good for initially debugging

- Formal Verification

Test properties
Deadlock
...
System Model
Test results
True or False
Find Counter-example

  - ✓ Equivalent to all case simulation
  - ✓ No corner case w.r.t given property

Advance VLSI Design 16

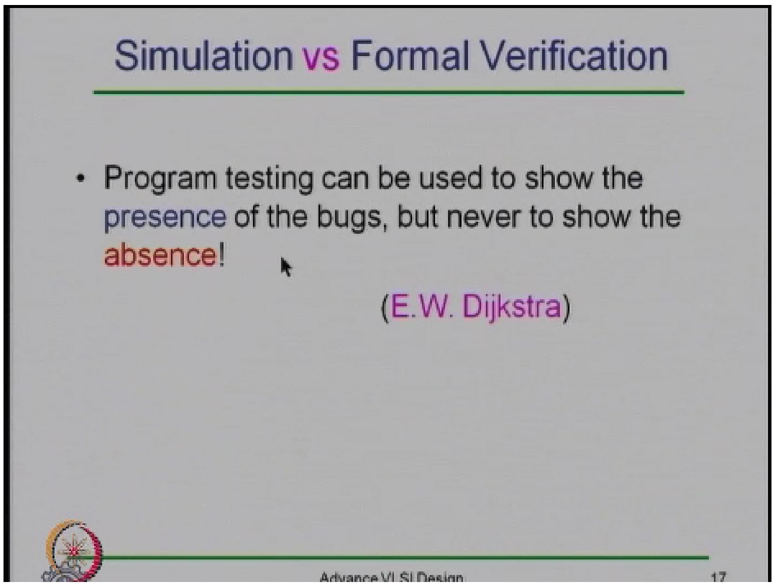
So now here what is the difference in the formal techniques and simulations based verification. Simulation based verification here you know this is you have to apply some test stimuli or vectors you see the response whether if it is correct then you say design is correct otherwise not. So now here you heavily rely on the corner cases, but this is essentially and very good for the initial design debug.

Whereas formal method here you need to supply the specification in terms of some mathematical formula or logical formula and you have to specify some of the properties. Like here for example if you are arbiter you want to design a arbiter you may have a property that your arbiter should not give access to multiple masters at the same time to use the common resource or if you have designed a traffic light controller, traffic light controller should not give green signal to cross roads these are the properties.

So now you supply this to your formal verifier will tell you whether this property is valid for that design or implementation or not. If it fails, then here it gives you a counter example. The counter example means here it will give you a trace of input under which you get the wrong result that helps in debugging or localizing the bug. So now here the formal verification technique is equivalent to all case simulation with respect to a given property.

So that means here there are no corner cases this is always correct with respect to a given property.

(Refer Slide Time: 48:52)



Simulation vs Formal Verification

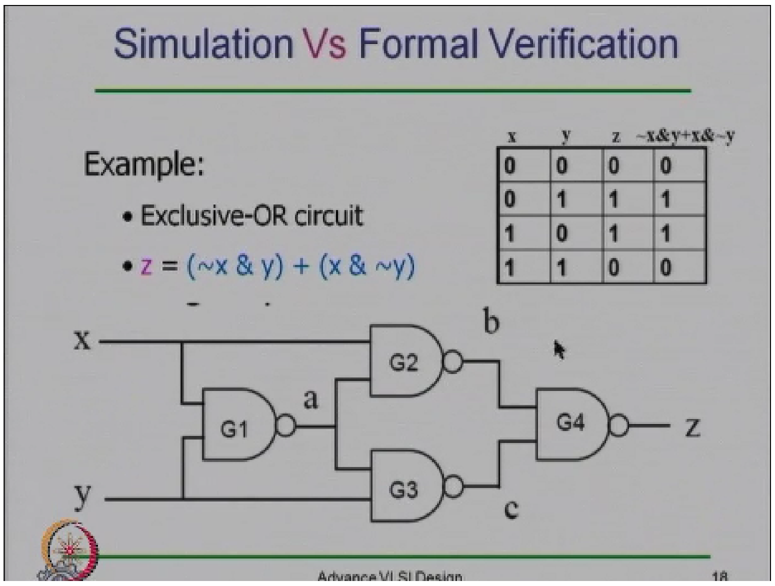
- Program testing can be used to show the presence of the bugs, but never to show the absence!

(E.W. Dijkstra)

Advance VLSI Design 17

There is a famous quote by E.W. Dijkstra and what he says is that here program testing that is equivalent to your simulation based verification can be used to show the presence of bug, but it can never show the absence of bug. It is only the formal verification which can say the absence of bug with response to given property.

(Refer Slide Time: 49:15)



Simulation Vs Formal Verification

Example:

- Exclusive-OR circuit
- z = (~x & y) + (x & ~y)

x	y	z	~x&y+x&~y
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

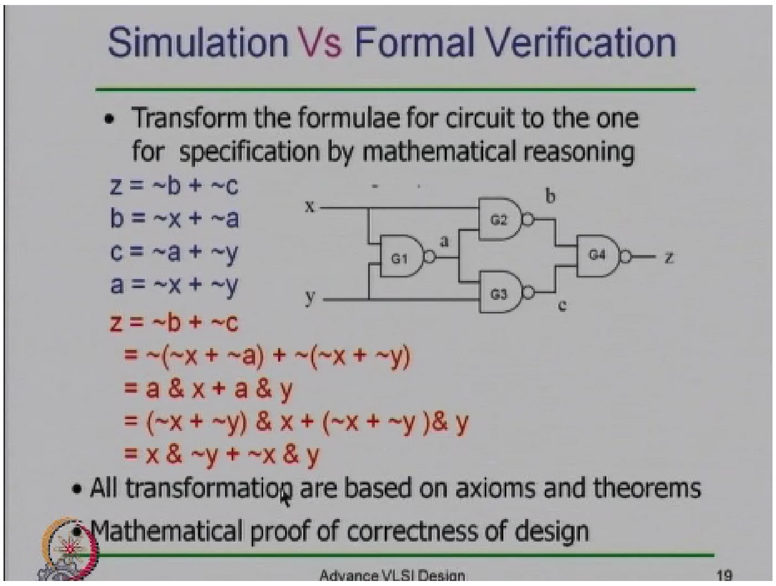
x
y
G1
a
G2
b
G3
c
G4
z

Advance VLSI Design 18

Now here if you look at the formal technique like say this is simple circuit. All of you know this NAND realization of XOR gate. If it is XOR gate in that case formally mathematically I can specify the output Z is equal to X bar Y + X Y bar right. Now here one of the ways is I can use the simulation based verification. So now for XY there can be four possible combinations and then from that I can simulate that here what would be the Z and from this one this mathematical expression I can also simulate what would be the Z.

And if there is matching in that case I say that this design is correct otherwise it is incorrect. So this is mathematical. This is simulation based verification. On the other hand, I can mathematically verify that.

(Refer Slide Time: 50:06)



Simulation Vs Formal Verification

- Transform the formulae for circuit to the one for specification by mathematical reasoning

z = ~b + ~c
b = ~x + ~a
c = ~a + ~y
a = ~x + ~y
z = ~b + ~c
= ~(~x + ~a) + ~(~x + ~y)
= a & x + a & y
= (~x + ~y) & x + (~x + ~y)& y
= x & ~y + ~x & y

- All transformation are based on axioms and theorems
- Mathematical proof of correctness of design

x
y
G1
a
G2
b
G3
c
G4
z

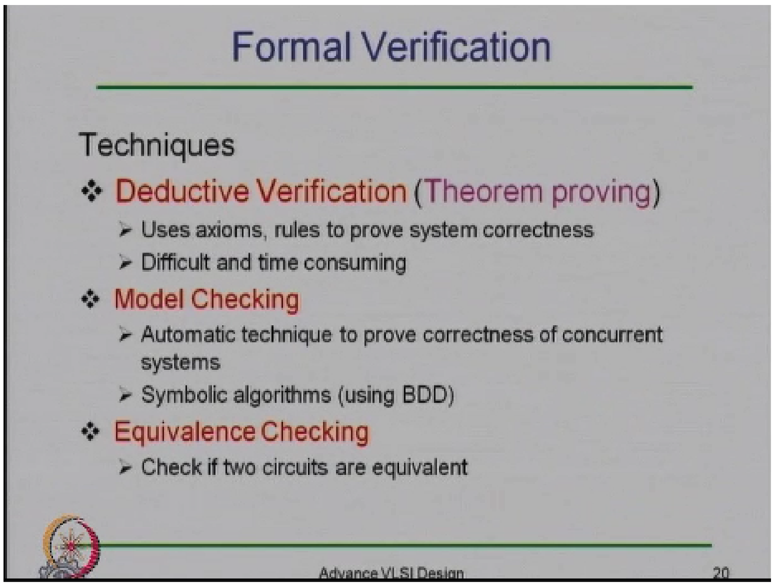
Advance VLSI Design 19

So now here the output Z I can specify in terms of B and C. I can say that Z is equal to B bar

+ C bar. Now here what is B. B is A bar + X bar and C is A bar + Y bar and then here what is A? A is your X bar + Y bar. If I put this Z this B bar and C bar and then here I put the expression for A in that case, I will get this as XY bar and X bar Y. I can rewrite this as X bar Y + XY bar that is same as your mathematical expression.

So that means here this tells you that this implementation always respects the specification that is given as X bar Y + XY bar right. So now here this is so what this says so this is based on the transformation that we use at various level which are based on the axioms and theorems so that means here we can use the mathematical proof for correctness.

(Refer Slide Time: 51:20)



Formal Verification

Techniques

- ❖ Deductive Verification (Theorem proving)
  - ➢ Uses axioms, rules to prove system correctness
  - ➢ Difficult and time consuming
- ❖ Model Checking
  - ➢ Automatic technique to prove correctness of concurrent systems
  - ➢ Symbolic algorithms (using BDD)
- ❖ Equivalence Checking
  - ➢ Check if two circuits are equivalent

Advance VLSI Design 20

So if you look at the formal verification there are three different ways to formally verified which are being practiced in the industry. One is deductive verification that means we have to prove the mathematical theorems. Deductive verification is semi automatic verification techniques because here this is based on the mathematical axioms and theorems in order to prove some theorem you have to use the axioms in some particular order.

And so here the tools are not very intelligent so you have to intervene the execution of those tools and guide them how it should progress to quickly verify that property. So this is semi automatic tool. So it is based on the axioms rules to prove the correctness. It is difficult and time consuming. There are other techniques here like your equivalence checking and model checking.

Equivalence checking is the check of equivalence of two designs. Like here for example if you verify the design of an adder vis-à-vis the specification now that may be your ripple carry adder. Now you optimize that and for timing you design carry-look ahead adder and you want to make sure that both of the adders are equivalent. Now here the equivalence checking can be used and this equivalence checking is fairly automatic technique.

And it can handle very large design and equivalence checking can be used at various level of abstraction. The other technique that we use is the model checking. In model checking here we specify the design behavior using some mathematical formula and we have to model the implementation using finite state machine or finite automata and then we prove the correctness of some of the property which are specified in using mathematical formula.

Again here this is based on the symbolic algorithm like BDD based technique or set based technique and this is fairly automatic technique. So model checking and equivalence checking are fairly automatic technique whereas the deductive verification is semi formal technique. In next couple of lectures, I will briefly discuss about these equivalence checking, model checking and deductive verification techniques.

Thank you very much for your patience for listening. We will continue with this in the next lecture.