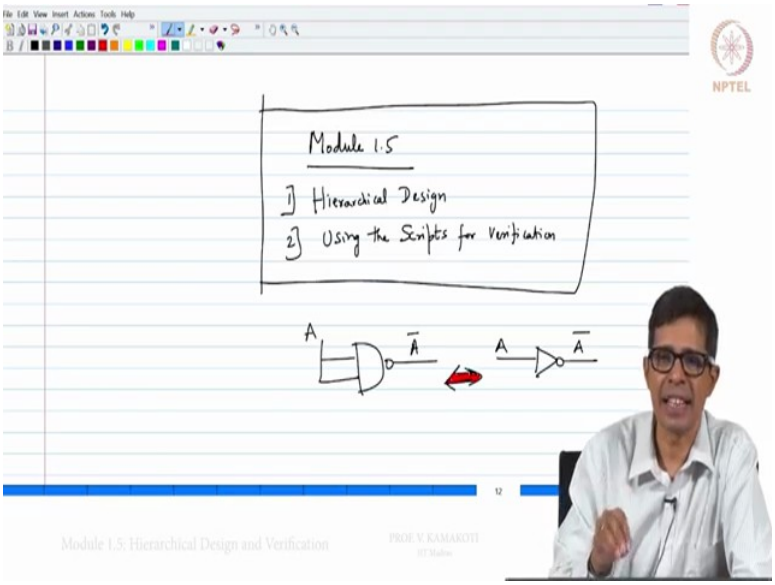
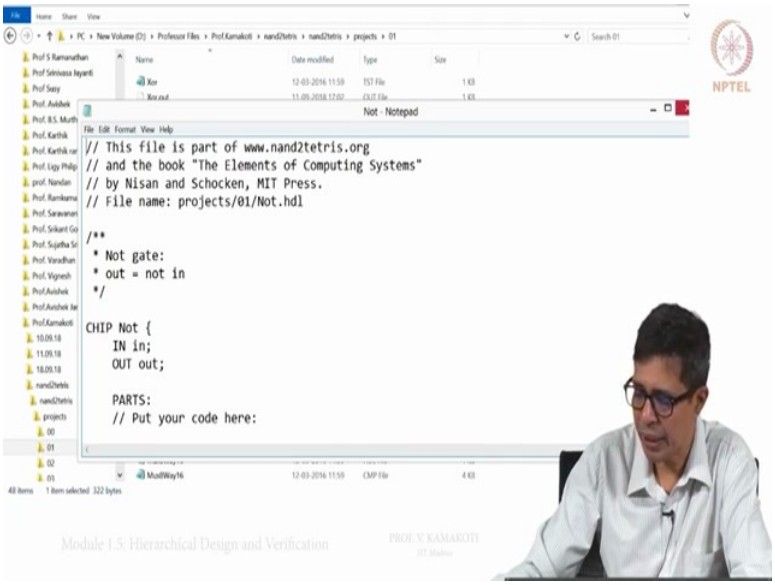
**Foundations to Computer System Design**
**Professor V. Kamakoti**
**Department of Computer Science and Engineering**
**Indian Institute of Technology Madras**
**Module 1.5**
**Hierarchical Design and Verification**

(Refer Slide Time: 0:24)



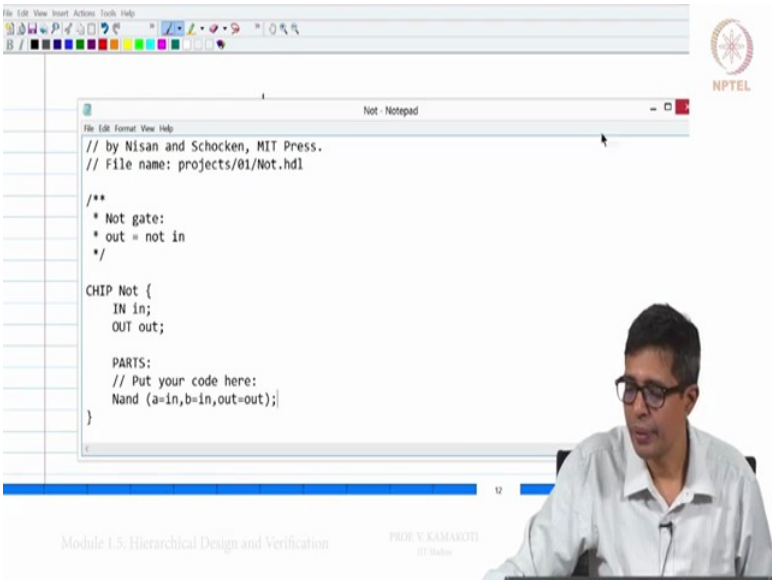

Welcome to module 1.5 and in this module we will basically look at hierarchical design also using the scripts for verification. What we mean by hierarchical design is that, note that as I mentioned in the earlier part of this module that we will be building the entire computer using NAND gates, so NAND you can assume it is available as a library function, so NAND is available to you and you build everything on top of NAND, so using NAND you built a NOT gate, using NOT you built something else you can go on, so what we will demonstrate very quickly is how this hierarchical design can be carried out.

(Refer Slide Time: 1:05)



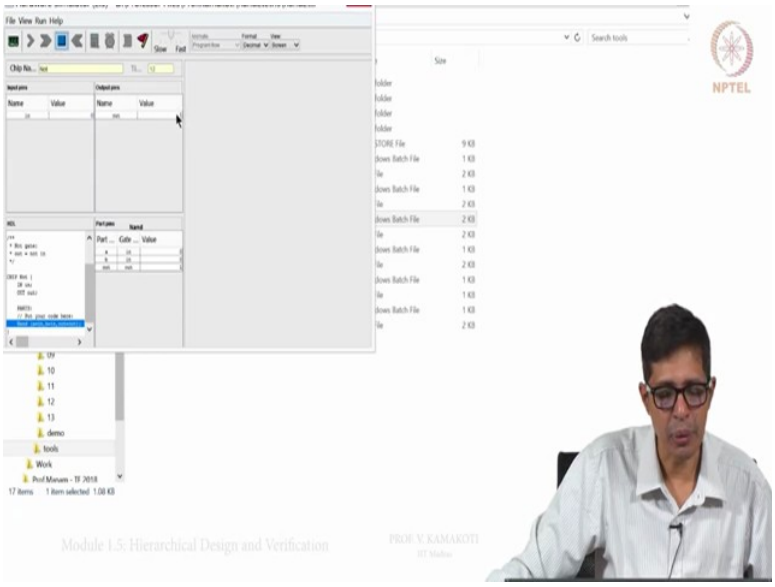

So let us go back to the project, project 1 so for every gate you see here or every design there is an HDL file, there is a test file and there is a compare file, so we will just take the HDL file and this is exactly what you get here and this is what you have to put here, so we have to basically build a NOT gate using NAND gate and we have already seen here that NAND a NOT is nothing but NOT be realised by a NAND by giving the same input to both the inputs lines of the NAND gate, so this is how you realize an inverter using a NAND gate, so now let us go and do this here.

(Refer Slide Time: 2:14)



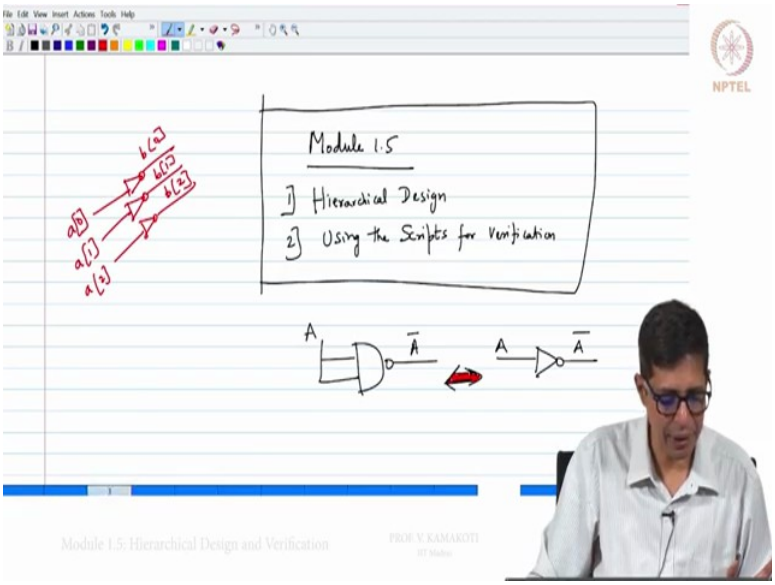
So this is basically NAND as a told you NAND is already available it is given to you and you have to build everything using that, so NAND so a is equal to in, b is also equal to in and out is equal to out, so let us go and save this.

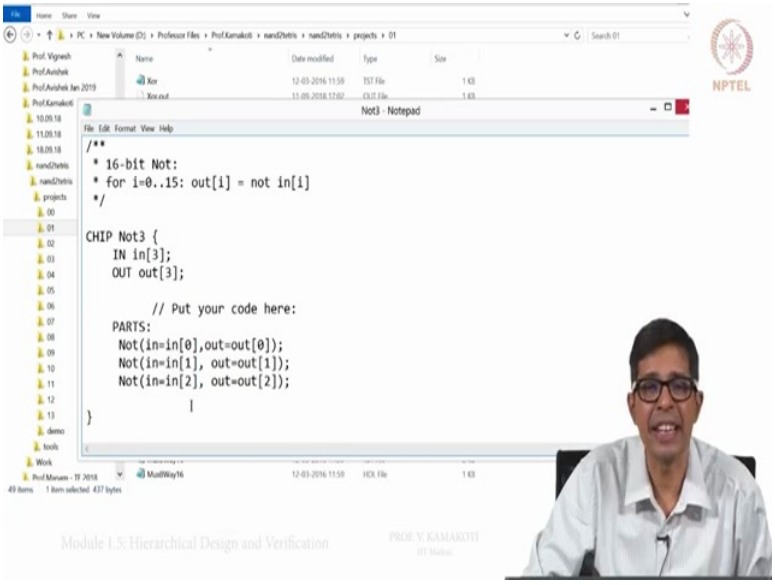
(Refer Slide Time: 2:48)




Now we will go back to tool and open the hardware simulator, since we are using Windows we use the Windows batch files here, now we will go and upload not, so not.hdl so this is the thing we have uploaded not, this is how we have coded it as you see here NAND is a equal to in, b equal to in. Now here basically I can give some input I give in as 1 and I run I get out as 0, I give in 0, I run and I get out as well, so we have built a NOT gate, right? Now using this NOT gate let us go and build say for example, so let us go back to the project so let us say I want to build some NOT of 3 for example, let us say I want to build not 3.

(Refer Slide Time: 4:09)



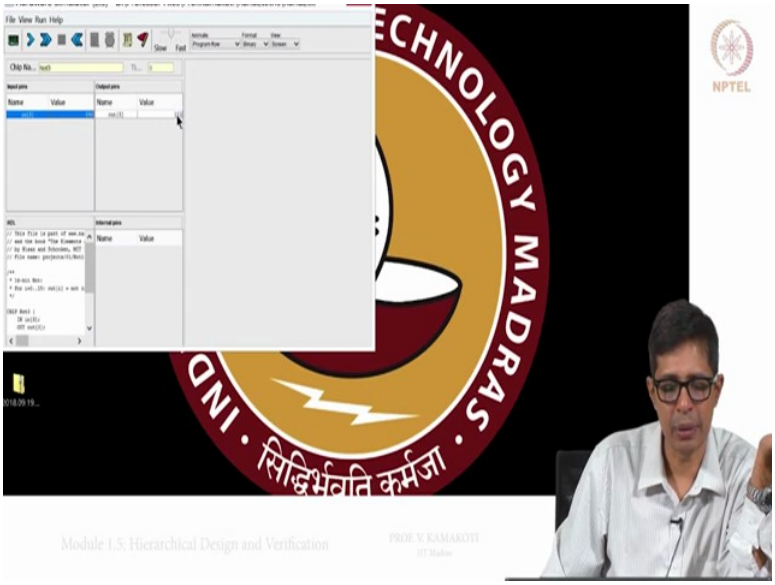



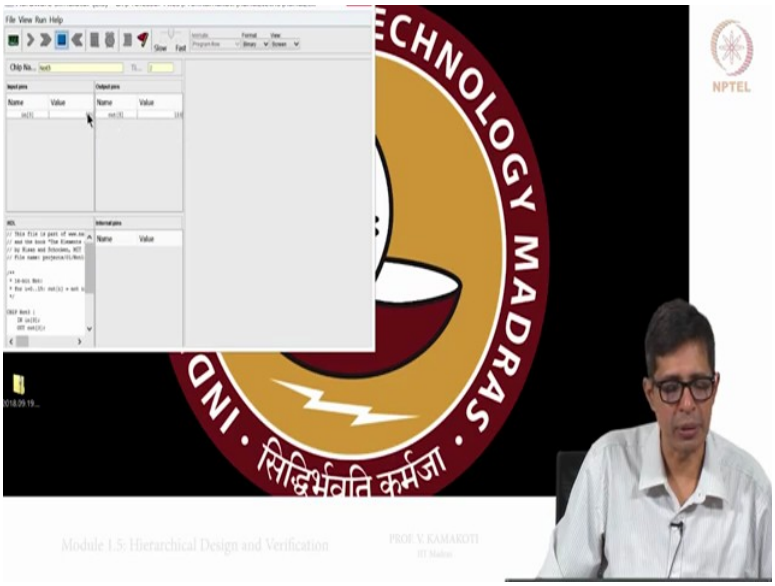

Not3 is nothing but I have 3 inputs a0, a1 and a2 and I need 3 outputs b0, b1 and b2, so I need 3 outputs a0, a1, a2, b0, b1, b2 and it basically has to invert a0 should invert to b0, a1 should invert to b1 and a2 should invert to b2, so this is what we need to achieve, so let us go and do this from scratch, so let us just take this not 16, so let us copy this and again. So let us open this not 16. Let me save this as not 3, right? So the name of the file should be the name of the chip otherwise this simulator will not work.

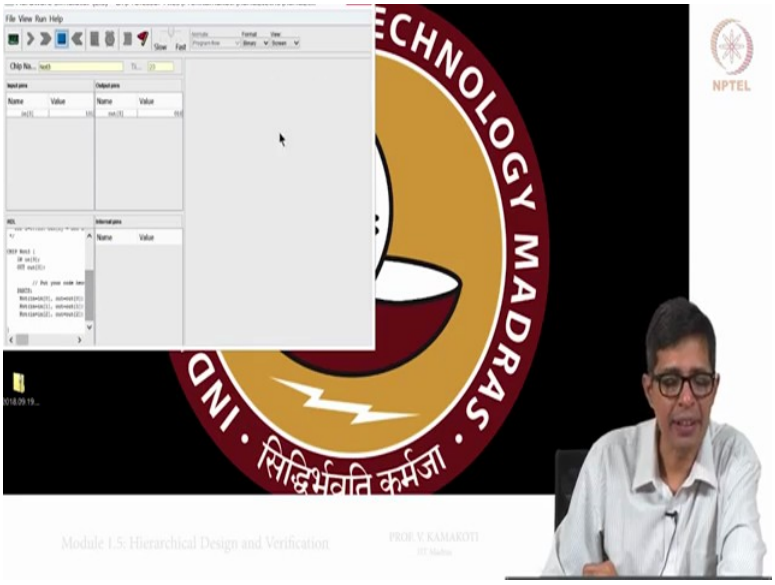
Please note that the name of the file is not 3 should be the name of the chip, so this is not 3, now the 3 wires that we are talking about in 3 and out 3, right? And what we need is now we want to basically invert… out i should be in i bar, so you we use not, not in is equal to in 0, out equal to out 0. Similarly we can do the remaining (())(5:57) okay so I can basically see

that not of in equal to in 1 and out equal to out 1, in equal to in 2 out equal to out 2. So I am using this not and how did I build this not? This not was built using NAND gate, so essentially now I have built 3 not gates which internally each not gate is realised using NAND gate, so that is the hierarchy, so I built one not and use this again and again and again to build another larger circuit, right? So this is the basic hierarchical design.

(Refer Slide Time: 7:04)

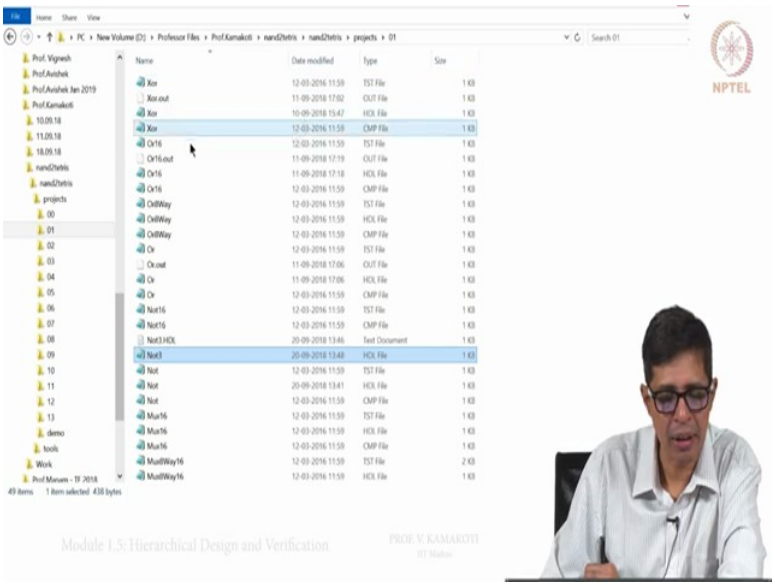


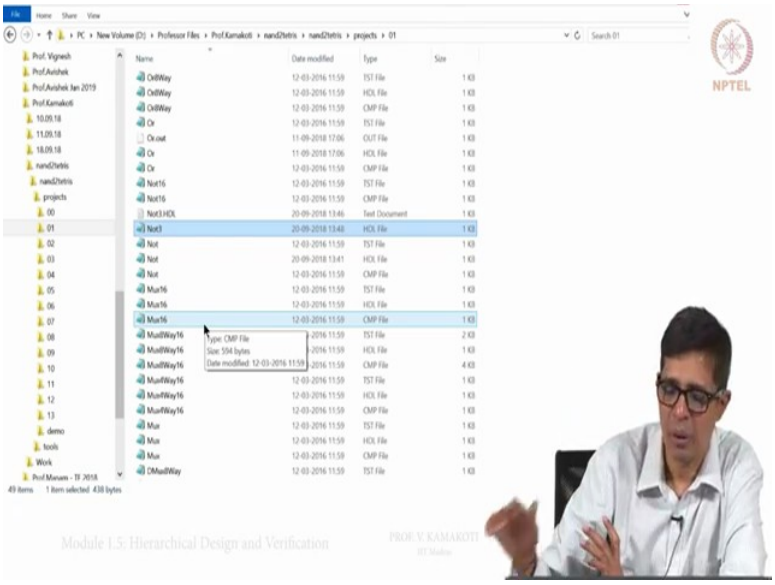






Now we will just go and execute this design, so I will load the chip so let us now go and execute this and so let us load this chip now the not 3 that we have seen, so this is the not 3 you load that chip and note that it got loaded. So we can give inputs and see the outputs, so the format we make it binary because we are giving multiple inputs, so this format also is binary, so the input now 000 and I get 111. Now if I want to change the input, so I can go and change any of this bit I make it 001 and I say okay I get, I run, I get 110. Similarly I can make it as 101 for example 101 and I run it I get 010, so this is how basically the hierarchical design can work wherein I have design not gate using NAND gate and now I have instantiated multiple not gates to basically build up the design.

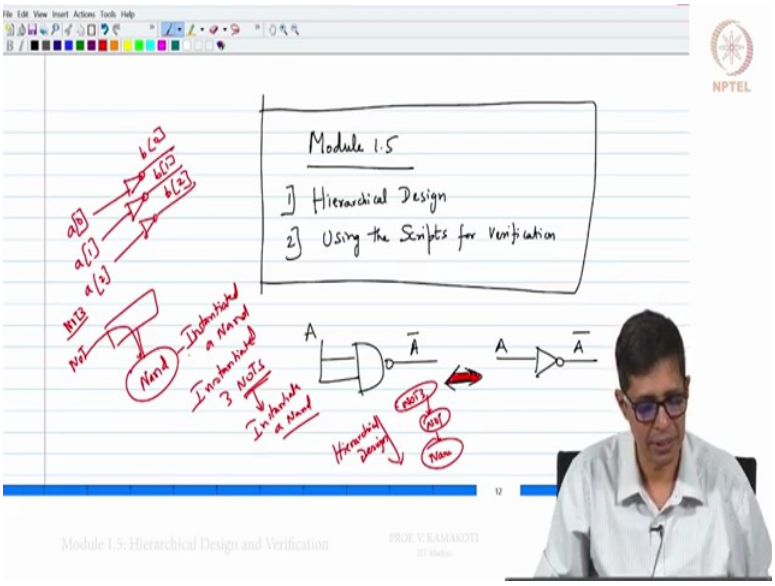
(Refer Slide Time: 8:32)





So in your project as we see here as we go in your project, so you have all these gates like XOR, so first you have to fill up you have NAND gate, using NAND gate you can make not and you can make XOR, you can make OR and from there, so we have other things like and, so all these basic gates and use this NAND gates to make n6 which will be 16 NAND gates not 16 to be 16 not gates etcetera, so you built one over top of the other and the basic thing that is available to you is only NAND and using NAND we have to do all this, so this is one thing that we learn what we call as the hierarchical design.

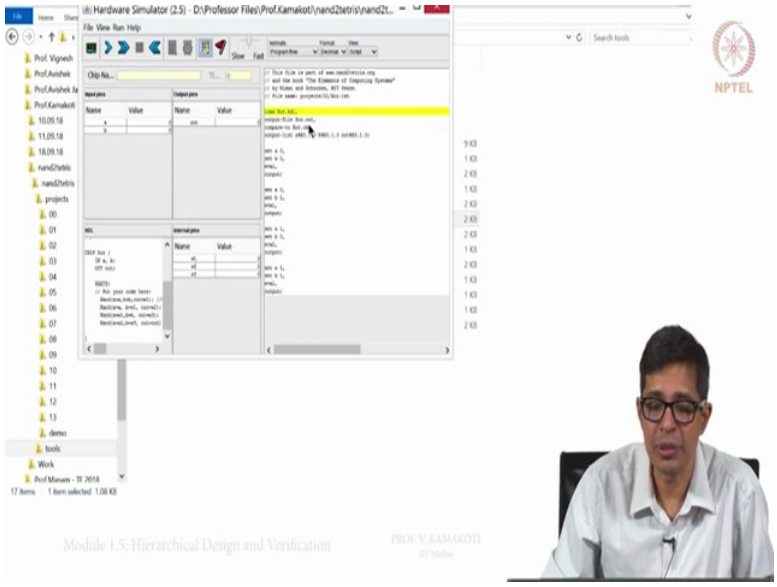
(Refer Slide Time: 9:18)

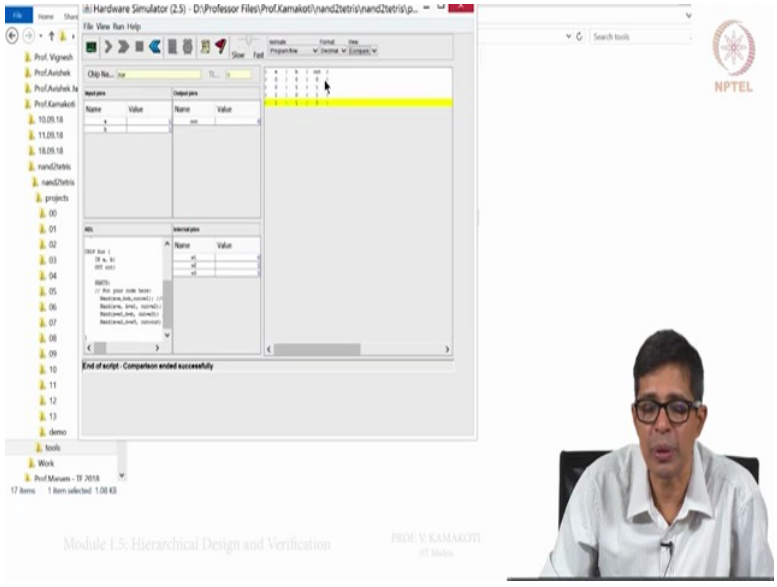


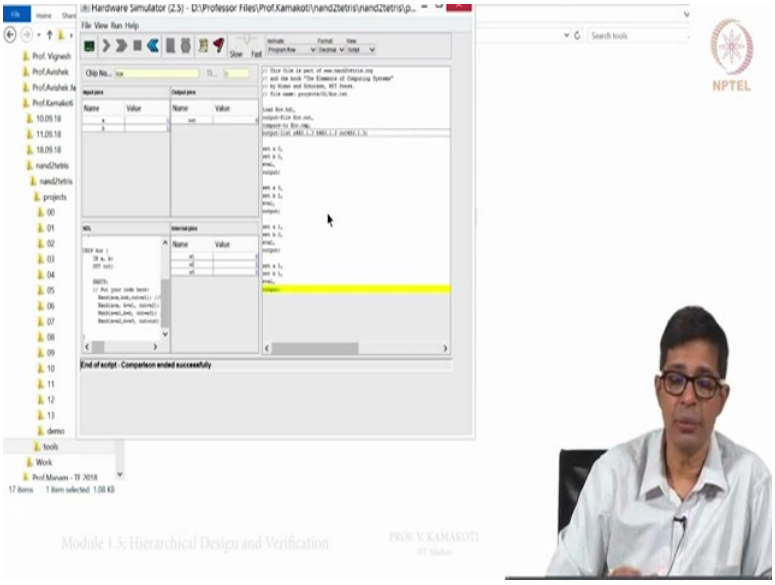

So this maps on to what we call in software engineering subject oriented design, so where we have lot of small objects the NAND is an object and using this NAND we built a NOT, in that NOT what have we done? We have used a NAND rather we are instantiated a NAND. NAND

is to get NOT, in NOT3 what did we do, we instantiated 3 NOTs which in turn each of these NOTs which in turn will instantiate the NAND. So every instantiation is called an object, so essentially by doing this we have got 3 objects, not 3 has 3 objects NOT NOT NOT each of this NOT as one object which is NAND, so we have built using NAND gate, we built NOT and using NOT we built NOT3, so this is the hierarchical design. The next thing that we will learn here is how to use the scripts for verification?

(Refer Slide Time: 10:56)






Let us go and very quickly see that for example let us say XOR. so we will take this and now let us take the XOR gate that we build long back, we have this XOR gate, so how was the XOR gate built? We did it in the previous time? It was built using 4 NAND gates. Now we want to verify whether this is correct or not, so essentially we load something call a script and when we click on this there is a script, we click on this button I will again do that, I will click on this button this is load script, I will load this XOR scripts, so what will this XOR scripts do? Load XOR.hdl it will load this file for you even if you have not loaded the chip it will automatically load it for you. It now says that the output file is xor.out.

So whatever output I am going to generate will be xor.out which go to this xor.out and then it has manually created xor.compare to which this out and compare will be compared and if there is any success or failure it will be noted and then there is an output list. We will just mention about this output list very shortly and what we do here? The input to this xor are a and b so we give a and b 00, 01, 10, 11 and after every combination and we say eval means it will evaluate that chip and output and how it will output? It will output according to this output list and to which while will it output? To the output file xor.out, so this is how this whole thing and the only thing that is pending is, what is this output list?

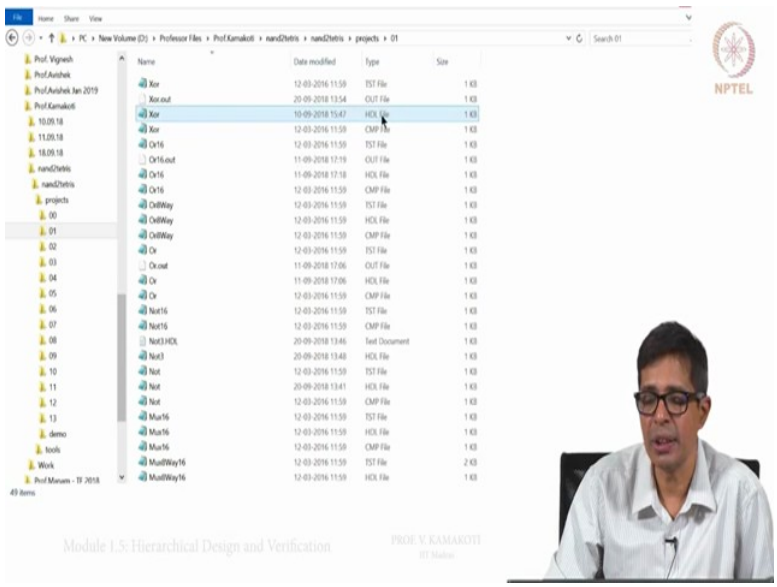
We will just tell within a fraction of time, so let us now run the script. When I run the script it says end of script comparison ended successfully that means whatever was there in your output file and the compare file both of them matched. You can also go and see here in this view you can see what was your output? So we had 4 outputs one after another after another, so what are the output generated? It generated 00, 01, 10, 11 and these are the things and this compare file is already given to you and that also looks exactly 00, 01, 10, 11 and what you

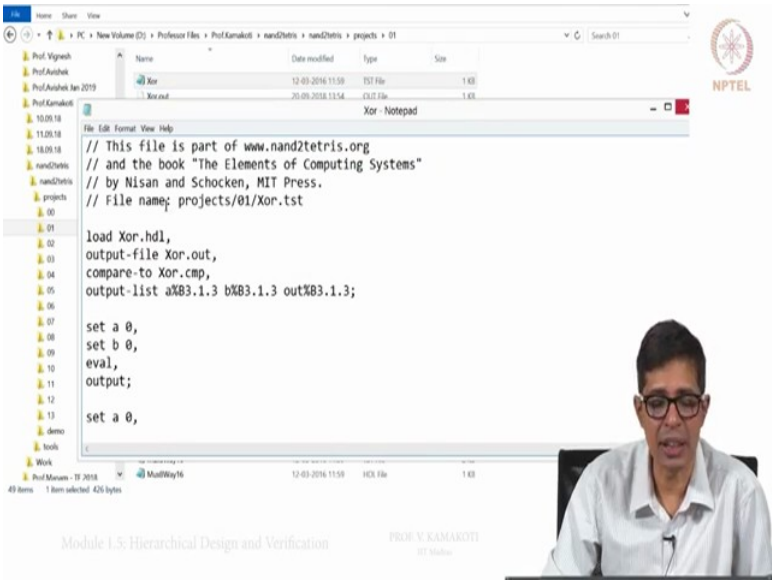
see is here there is a pipe symbol after that there are 3 spaces then a is printed and then 3 spaces.

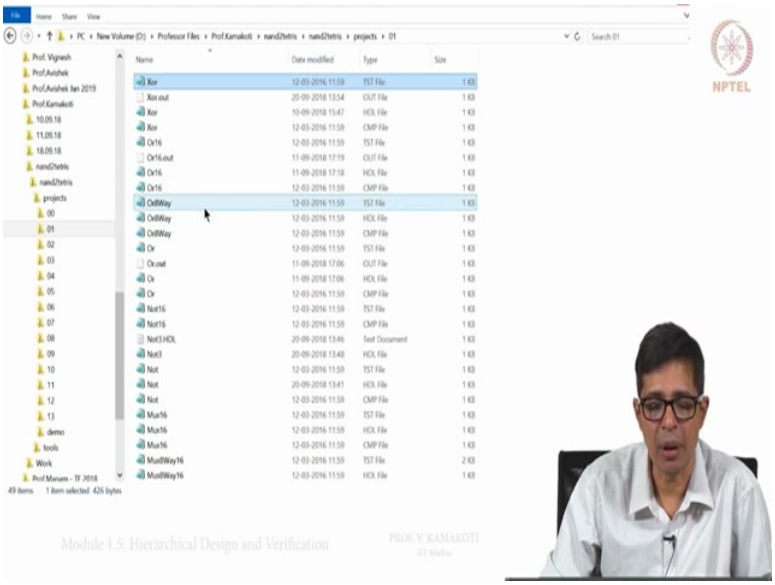
Similarly then pipe then 3 spaces b is printed 3 spaces then pipe 3 spaces then 0 is printed that is the value of out and then 3 spaces, so this out goes across those things, so essentially the format of outputting is 3 space then the value then 3 space, 3 space then value 3 space, 3 space then value 3 space and this is exactly what you can see on your script, so I go view script that is what you see.

So the output list is print a which is a binary such that you give 3 space before it print the value then 3 space after that, then print b which is also a binary this capital B stands for binary, if you want decimal you can put capital D and then print 3 spaces 1 then 3 spaces out again binary again 3 spaces then value then 3 spaces. So this is how this will work and this is how you compare these flows, fair enough. So this is overall way by which the hardware simulator works and this is how you keep comparing the things. So essentially what we have done here is that we have written…

(Refer Slide Time: 14:55)






So when you go to this project. All the projects that you will be doing as a part of this course you will have hdl file, you will have a compared file and you will have a tst file, so the hdl file is where you have coded the compare file is what already is generated and the tst file is also a script file were already generated which will test your design to certify whether it is working correctly or not, so these 3 are the 3 files that you will have for everything like for example r16 you have test file, you have hdl file, you have compare file.

Similarly r8 you have test file, hdl file, compare file, Or you have test file, hdl file, compare file and so on. So once you execute the design the out file will be generated for example after you finish OR now we have finish XOr that xor.out was generated. So now this xor.out is same as xor.compare that is what your script has shown. So this is how you can run a script and verify whether your design is working correctly or not. Thank you.