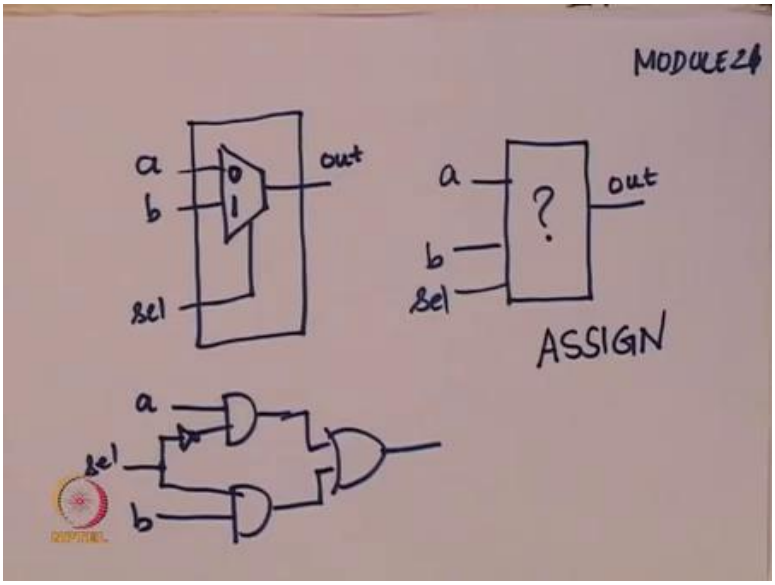
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 21
Verilog Demonstration: Assign Statement and Instantiation

Welcome to module 21 of week 4. In this module, we are going to look at Demonstration of Verilog once more. So, and when I do that, I want to show some of the principles in writing verilog code. Particularly, I am going to introduce you to a new kind of statement called assignment statement and I am also going to show you, how to do, what is called hierarchical design. So, let us start with this basic problem.

(Refer Slide Time: 00:45)



So, let us say I want to design 2 to 1 multiplexers and for this 2 to 1 multiplexer, I want to write verilog code. So, let us look at the picture here. So, what I want is, I want circuit which will behave as a 2 to 1 mux. So, it suppose to take 3 inputs a, b and select and it has to give 1 output called out and internally I want a 2 to 1 multiplexer and to be more specific, I want this. So, when select is 0, I want a to be in the output and when select is 1, I want to be b to passed on to the output. So, this is the hardware that I have in mind and I want to be able to describe that.

So, one thing you can do for this particular multiplexer is, you know that the multiplexer

can be implemented using AND, OR and inverters. So, you can go and do that. So, we already discussed this in the last week said verilog demonstration. So, we saw, how to use the primitive gates or primitive data types, so we looked at AND, OR and so on. We can use those primitive structures and write, what is called structural description for this.

So, a structural description would essentially say, I want 1 AND gate which takes a, another AND gate which will take b as input and you have a select line, you would imply some structure like this. So, once you have a structure like this, you will have four verilog lines, let us say 1 for OR gates, 2 for this AND gates and one for the inverters. So, that is completely possible, but I am going to do something different for this class.

So, we go to see, how to do this verilog description in a slightly different way and the first thing we are going to look at is, what is called an assign statement. So, the statement is called an assign statement and I am go to this website EDA play ground dot com and show you the demo of how to write verilog code for this and I will also write a test bench to go with it.

(Refer Slide Time: 02:57)

So, I am going to do this on EDA play ground dot com, I am already logged in. So, I am designing a circuit called 2 to 1 mux, I am going to call it mux to 1 and I know that, there is 1 output and there are 3 inputs. So, one thing that verilog allows you to do is to directly say, whether it is output or input, when you describe the module itself. So, I am choosing to do that now, so I need 1 output and 3 inputs, namely select input a and input b.

So, unlike the last class, where we wrote only the names and then we explicitly set inputs or outputs in a separate line, I have done this in the modules line description itself. And I generally have this habit of putting module and n module; so that, I know, I do not have to remember to do it later. As when I create a block of code, I also put the corresponding matching description at the end. So, now I am going to describe the output.

So, at this point, if we did what is called structural description of a multiplexer, what we would have done is, we would have instantiated the two AND gates, 1 OR gate and 1 inverter and so on and wired all of them up. But, as I mentioned earlier can work at different level of abstraction. So, I am going to write it another level of abstraction, where we can actually write the Boolean equations directly.

So, for example, we know that output is actually select bar into a plus select into b, this is what we wanted from the multiplexer. So, there is a way to do that directly in verilog. So, I am going to do that now. So, select bar into a, so I will come to the syntax in a little while or select into b. So, if you look at this statement, this is just a Boolean equation. So, I am following the syntax of verilog and verilog syntax is very similar to syntax that c and c plus, plus has.

So, this ampersand in verilog is a bit wise AND, this type symbol that I have here, which you can see right above the enter button, that is bit wise OR and this exclamation symbol which is along with the one key that is NOT. So, you have NOT, AND and OR. So, if you know how to write expression is C, this is very similar. The only key thing we have to remember is, this is a bit wise AND, and this is a bit wise OR. So, we have only one ampersand and one type symbol.

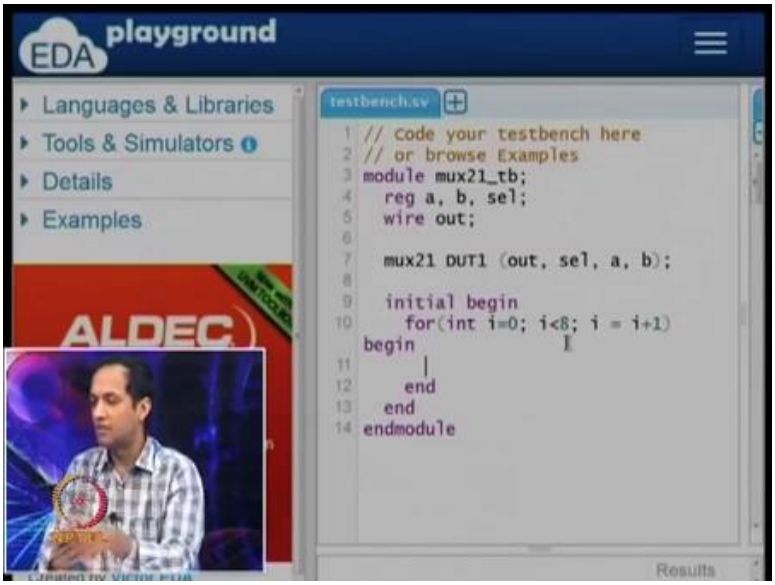
So, it will be nice, if you can actually write an equation like this. If you know an equation, it will be nice, verilog can actually letters do something like this directly. In fact verilog does that and the way to do that is, verilog allows us to do something called an assign statement. So, there is a key word called assign, you can notice that, the assign statement actually became, the keyword assign changed to color.

So, we can say assign output equals NOT of select AND a OR select AND b. So, this is select bar into a plus select into b. So, usually if I put parenthesis to explicitly say, this is one group of operation and this is one group of operation. So, you do not have to do that, once you know, what is the precedence of ampersand over OR and so on, but this is a very safe way to do it. So, let us say I have a design in hand now, it is as simple as that,

that is it, no instantiation nothing.

So, what we have done is, you taken the Boolean equation required for output in terms of select a and input and we have directly written that as a Boolean equation. So, it is a very simple designing and it is okay to do that.

(Refer Slide Time: 06:53)



Now, I am going to write a test bench for this, so for the test bench, I am going to call that mux 2 1 underscore test bench. So, I already said that, I use this convention called underscore t b to indicate that, it is a test bench for this particular module. If I am testing from an x OR gate, I would have put x OR underscore t b and so on. So, now, what I going to do is, I need descriptions, so I need a mechanism by which this whole set up is going to be inside a test bench.

So, just like what we did in the previous video, we need to be able to give a, b and select as inputs and we need to be able to observe out, which is the output signal. So, as I did in the previous lecture, I am going to declare three regs, we will see what a reg is later. But, we will use three regs, wire a and b and select and I am going to declare a wire called out. So, these are the things which are going to be in the test bench and we need a mechanism by which the test bench gets connected to the actual module, so the design under test.

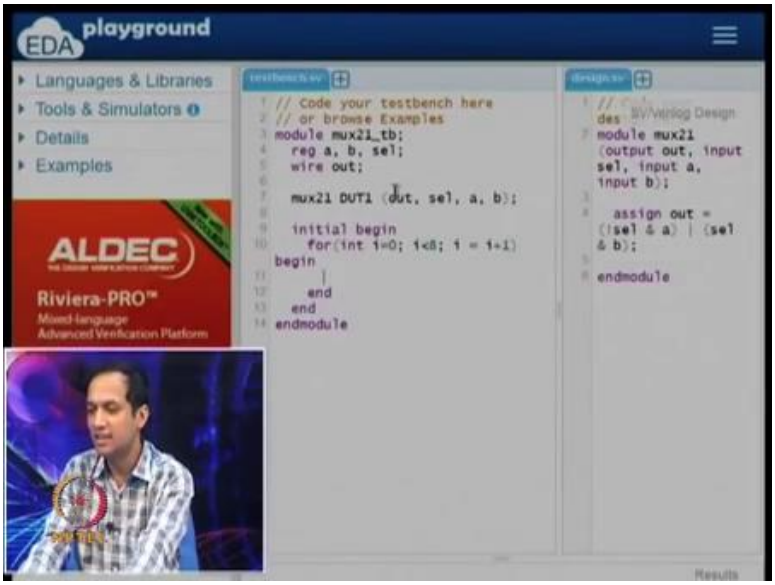
So, for the design under test, we know that, the design under test the type of the design is called mux 2 1 and I am going to give a name for that called DUT 1. The order of the pins where output select, a and b. So, at this point what we have is, you can imagine that,

we picked up a breadboard, we put a multiplexer in it and we have not connected any wires or anything yet, so that is what we have so far.

So, we have put the breadboard, we have put the multiplexer chip in it, we still have to say, this is the a pin and it has to connect to this power supply, I am going to iterate over all values, all these things will have to do. So, I am going to write an initial statement for that. So, I have one initial statement, so this initial statement, in this initial statement I am going to write a loop which will iterate over all the possible combinations of select a and b.

So, just like what we did in the previous example, select a and b will have 8 different combinations, so I going to write that. For int i equals to 0, i less than 8, i equals i plus 1. So, this is the for loop, begin, end. So, because of the size of the font that I am choosing and so on, I have done this, so that you can see in the video clearly. But, if you are sitting in the front of a computer, you probably have a much small of font size, this begin would come into the same line itself.

(Refer Slide Time: 09:39)



So, l can show you, how it will look. So, it would look like this, whereas, so if you have a properly sized windows, it would look nice, as platform is quite nice. But, I chosen a bigger font, so that you can see, what we are doing.

(Refer Slide Time: 09:55)

So, let me work with this font now. So, we have the loop running and already I mentioned this in the last class. If I want to give a vector of inputs, so select a and b are three different inputs and when I say, i is assigned to select a and b, you are select a and b on the left side. So, i gets value from 0 0 0 to 1 1 1 and these specific bits are assigned to the left side. So, this is something that you have already seen, I am going to do something slightly more this time.

So, I also said, I am going to give this every 5 time units, I will put that in. So, till this it is very similar to what we did in the previous test bench, on top of this I am going to do a few more things. So, one thing that we do for test bench is that, we can always go and visually see, what the output is, we can see, what the waveform is and so on. But, at sometimes it is also a good idea to actually know that, this is what I am expecting from the design and am I getting the correct output or not.

So, when we write the test bench modules, so this is not something that you will get from your breadboard comma your oscilloscope kind of set up So, you have to inspect as a human being, you have to see what the oscilloscope has and you have to check, whether the outputs are correct or not. You have to do those things or you have to inspect the LED that you are connected or not.

But, in verilog, since you are writing a program, you can also programmatically check whether the output that you are getting is correct or not. So, I am not going to do that right now. So, I know that if select is 1, so we will come to the syntax later. If select is 1

and if output is not equal to a, let us say I wrote that design now, the design is actually correct. But, it is possible that I make some mistake or if you go and write your own design, you may make a mistake.

So, what I am going to do is, if select is 0 and output is not equal to a, I am going to declare, so I am going to do something. So, I am going to say that, there is some failure; there is something that is happening. So, I am supposed to get a, whatever a is, whenever select is 0. However, if for whatever reason it fails, I am going to indicate that, there is a failure.

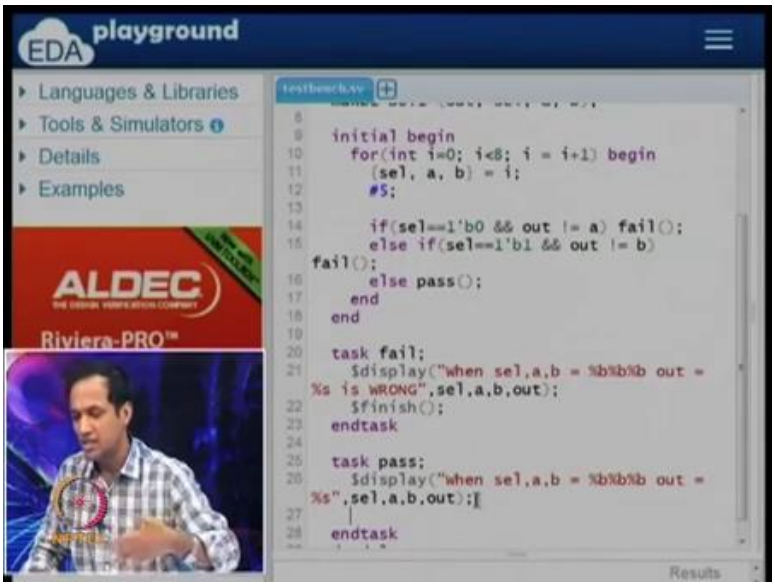
The same thing can happen, if select is 1 and output is not equal to b, this is also a problem. So, I want to say that, this is also a place, where I want to indicate that, it is a failure. If it passes, if all the things pass, I am going to say that, it is passing. So, I am going to do a few things in a little while, so again if you want to see the structure, this is how the structure looks like. There is an if statement, there is an else if statement and else statement.

If you look at what is happening in fail and pass and so on, it is looks like a function call that we are making, it is in fact actually a function call. So, on the right side, we have a design in the test bench, we are going to use the full power of writing software as though you do it in the programming language. So, in you write a design description, you generally do not write functions and so on.

But, when you are testing it, your test pad or the test bench is essentially a piece of software by itself, it is not implying any hardware directly. It is not that we want to synthesize an oscilloscope or we want to get a voltage supply out of this. If we give this voltage supply and if you have to see the waveforms, what is going to happen, that is what we are doing in this whole business of simulation.

So, you can use the power of software programming inside the test bench, anything that you do in test bench, which uses traditional software techniques, it is actually welcome and it is good to have. So, what I have done is, I have put a series of if else statements and there is a for loop, this is the overall structure.

(Refer Slide Time: 14:20)



So, now let me show you, what some of these things mean. So, if select equals 1 apostrophe b 0, this apostrophe is the signal or is the key, next to the enter key. This is the close code, close single code, one tick is what verilog forks call it, one tick b 0 and out is not equal to a, then I know that there is something wrong in the design. So, if that was true; however, it is possible that when select is 1, somehow out was not equal to b.

So, one of the cases would have been wrong, if either of these thing happen, I know that, there is something wrong in my design. If this is correct, then everything is correct, I can pass this test case, this test case is correct. So, what I am going to do is, I am going to put this side an initial block and what this will do is, it will iterate, the for loop will iterate from 0 to 7, it will stop when it is 8.

So, 0 to 7, it assigns i to select a comma b, select comma a comma b and you can imagine this as signals that have been fed into the device and you will get output from the device, in this case, the device is mux to 1. So, let us assume that, there is 5 units of time that I have given for the mux to respond back; that is what this hash 5 means. I am giving some unit of time for the hardware to respond back.

In this case, it is all in software, but in reality, you have to wait for the output to change, you cannot give an input and instantaneously expect the output to be ready. So, we are mimicking by putting a hash 5 and these three lines, what it is doing is, we are checking whether the outputs are correct or not. So, this is a very quick and easy way to do this. For example, if you written a design, let us say you design the very complicated circuit,

take two numbers and multiply and so on. You have your design and that is on one side, on the other side if you want to check whether the result is correct or not, you could do a into b for instants.

So, a into b does not implying any hardware at all, you know that it is a multiplier, how to design a multiplier; that is not clear from it. Whereas, when we write design descriptions, it is usually very clear to us, what we really want from the verilog description. So, so far we are there and I am going to write two, so I am going to write, what are called tasks. So, I am going to write one task called fail and this task is supposed to take care of the failing condition.

So, when will we call this functions fail? If one of these if conditions were correct, so somehow the output was not even when the select input was 0 or the output was not b, when the select input is 1, then we have a problem. So, I am going to say that, there is a problem in these cases. So, I am going to print something to this screen saying, display when select a, b, I am going to write it like in English. When select a b is these three values, so this is similar to printf statement, you can see that, it is resembling the format of printf.

When select a b is this, output equal to this is wrong, I am going to write something like this and what are the different things it is just like a printf statement for C. So, the first part is a format specifier and the second part is the actual parameters. So, when select comma a comma b is select comma a comma b, output equal to whatever output we got, this is wrong. It is okay to say, it is wrong here, because you will call this function only when the output is wrong, you will not call this function to the output correct.

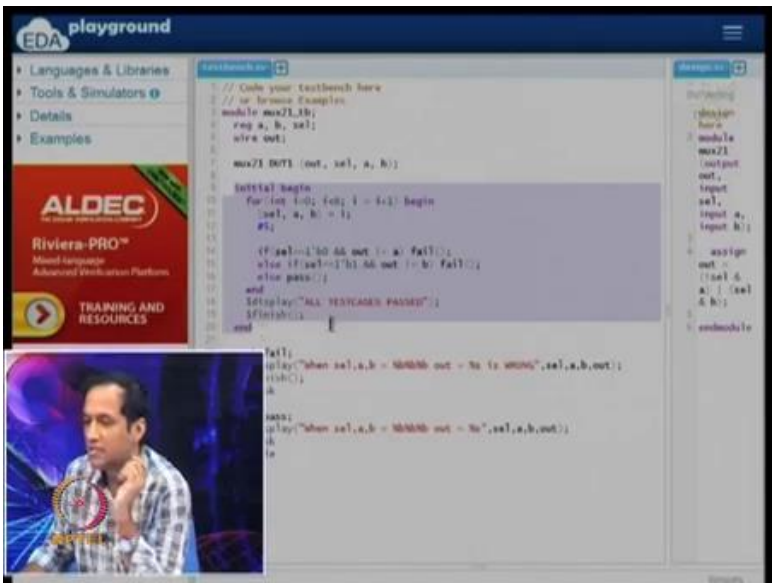
And this take care of the fate that, there is a problem and if there is any problem in your design, if there is anything wrong in the design at all. Ideally what I want is, I want to just say that, do not continue with the simulation anymore, there is the mistake in the design, you have to go and check it. So, I am going to finish simulation. So, what this will do is, abruptly it will finish the simulation will not even go to the next iteration required for running this program, so I will abruptly finish.

Similarly, for pass, so what I am going to do is, I am going to say that, if it is pass which means for each of the combination of inputs, if it is correct, I am going to put this statement. So, I will copy from the previous one, I will say that, out to equal to this and that is it. So, I will not say, it is wrong, the output is actually correct. So, whenever I say

the output equal to that, we will assume that, it is correct.

So, if it is wrong, I explicitly wrong, if it is correct, I will just print it out and in this place, I am not put dollar finish. Because, for in one into combination of the output is correct, I still have to check all the input combinations. So, I should not finish that, I should check all the input combinations.

(Refer Slide Time: 20:06)



If it fails and go to get out immediately and say that, there is something wrong in this design. But, if it is passing, then I am go to let it pass, let it go to the variant and let us now go back to this initial block. So, what would happen, you would run for 0 to 7 and every time, you can thing about it as, there are 3 inputs in if I given, you waited for some time, the hardware get the output, you check the output.

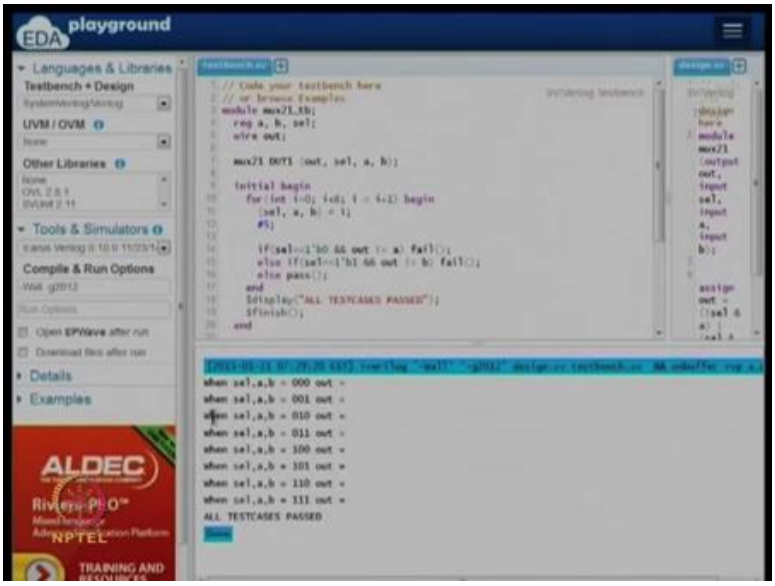
If the output is correct, you pass, so which means, you this will print something on the screen and it will go to the next iteration. If it is fails, you will call this task call fail, it displays what is wrong in your design and then finishes this simulation. So, at some point, for all the input combination if everything passed, you will get out this for loop, so the for loop would terminate.

At this point, I can actually say that, all the test cases. So, I can be happy about this, all test cases passed and now, I am ready to finish, so this is the overall setup. So, what we are going to do is, so I will just reduce the font size again. So, that you can see the structure, over all structure, even if you do not see the actual contents, you can see that, there is an initial statement here, which take care of giving the variant input

combinations, it also checks whether the output produce is correct or not.

And if the output produce is incorrect, we are going to call this function, when the output produce is correct; we are going to call this function. So, far we are okay, I am going to now run this. So, I am going to save and run it. So, what happen when you run is, it comes in the bottom, let see what is happening. So, I did not pick the correct simulative and so on, let me do that.

(Refer Slide Time: 22:10)



So, we will pick system verilog and I am going pick correct simulator and so on, let me do that. So, will pick system verilog and I am going to pick for tools and simulator, I am going to pick i verilog 10.0. So, right now I do not want the way forms. So, I will go and save it and run it. So, sometimes it can have issues, if you see that there is nothing which changed in the output, just go and run it once more, you will see the little become a blank screen and eventually, you see the output again.

Now, let see this, so this design is actually correct, it say when select comma a comma b is 0 0 0, output is blank, then there are several things which are all blanks. So, it is looks like I did something which is incorrect. So, let we go and see, what it is. So, I said when select a comma b is these three values, output equal to percentage S. So, percent S is a not a valid specifier, so should have percentage b. So, percentage b print it as a bit. So, we have seen d and other things in C programming.

So, percentage b means print it as a bit, so that is a mistake that I have done. So, I am going to as corrected that mistake and I am going to run it again. Let me run it once

more, now let see the output, so let us check pick a line in check it, when select a comma b equal to 0 1 1. So, when select equals to 0, a equals to 1 and b equals to 1, output is 1, so which is in fact correct.

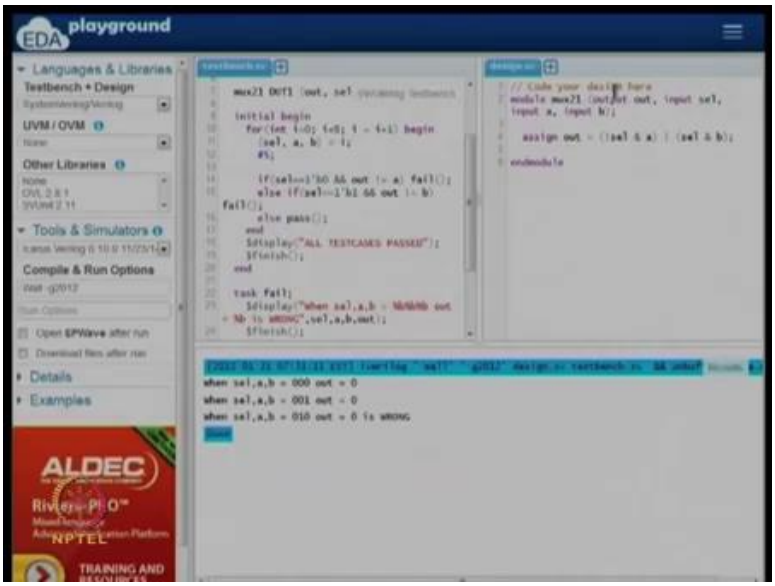
Let us check this line, when select is 1, a is 1 and b is 0, so we know that select bar into a plus select into b, so we should be selecting b. So, b is 0, so output is 0, this seems to be correct. So, let me show, what would happen I would make a small mistake. So, let say by mistake, I forgot this not condition. So, I put select AND a or select AND b. So, let say I make that mistake, let say you wrote a design like that.

If I save it and run it, I am expecting something wrong correct. So, this in run, let run it once more, let see the output now. So, when select comma a comma b is 0 0 0, output is 0 and select comma a comma b is 0 0 1. So, what we are expecting is, when select is 0, and we are a is 0 and b is 1. Since, select is 0, you are still supposed to get 0 out of it, so that is okay. When, select is 0, you are printing a, let us look at this line, when select is 0 and a is 1 and b is 0, we should be outputting 1.

However, the output is coming out of 0, because we made a mistake here, as soon as in the sequence of input combinations, we find out in error. So, what would happen is, in this for loop, you have i equal to 0, for i equal to 0, everything passes, for i equal to 1, again it passes, for i equal to 2 however, the design into give a incorrect result as one as incorrect.

As soon as this something incorrect, this is a very conservative way of programming as soon as you see something incorrect, save that incorrect, unless you go and fix the design you should not allow to continue. So, this is the setup. So, I will go back and put select there and that is it. So, that is basically the design.

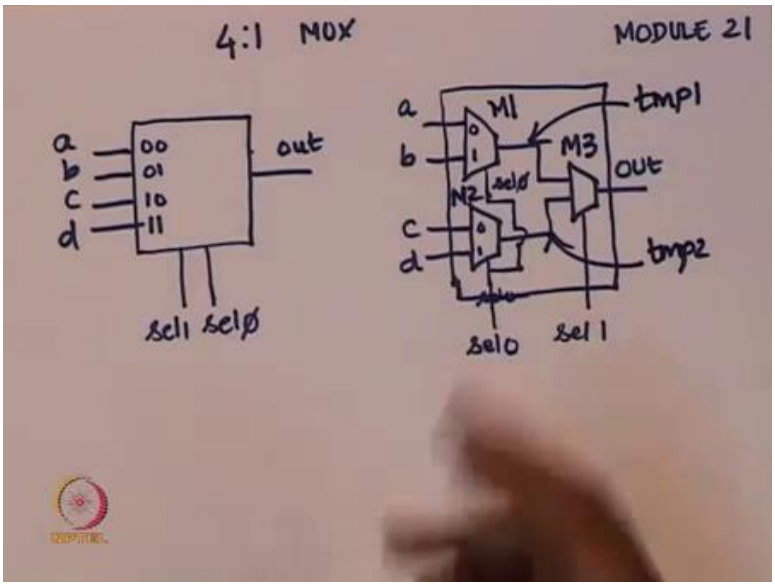
(Refer Slide Time: 26:06)



So, now, I know that, this design is correct and if you want you can go and open the way form in one. So, this is a very simple example in this case, I did not open, I did not put any statements for way form viewing and so on. I just want to illustrate that, you can actually something called automatic checking right inside the design. So, I have done that now, instead of looking at the print out and manually checking one after the other. I know that, when select is 0, the output must be a, when select is 1, output must be b, I am explicitly checking it in the program.

So, this is something you must have also observe the test bench that I provided you for the XOR gate. So, in this three at the end of it, I specified practice problem the test bench there is also something that actually uses the same technique. So, now, let we do one more design. So, in this design, I want to show you something, which is equal and to what is called instantiation. So, now, let say my design problem, let us look at the picture.

(Refer Slide Time: 27:00)



Let us say the design problem is design of a 4 to 1 mux. So, when we design 4 to 1 mux, the test cases are like this. So, this is the setup. So, will have two inputs select 1 and select 0, these are the 2 inputs and these are the select lines and I have 4 inputs proper inputs a, b, c and d, there is a single output. And the select 1 and select NOT, if they are 0 0 then output should be equal to a, then if these two are 0 1, the output must be equal to b and so on. This is what I want from the verilog design.

And I could go and write equation for this, I can go and write out equals combination of select 1 and select NOT a, b, c and d; that is one way to do it. I already demonstrated that, I suggest that you actually go and write yourself. Now, I am going to something else which is slightly different. So, I am trying to show different ways of writing verilog for a design problem.

Let us I want to do this in verilog, but I want to use 2 to 1 muxes, so I have 2 to 1 muxes this is valid for 4 to 1 mux, designed using a 2 to 1 mux. So, you give a and b is inputs in 2 to 1 mux, c and d inputs to another 2 to 1 mux and give select 0 as the select line. This is not select 1, this is select 1, this is select 0, so this 1 is correct, so this is not a mistake and select 0, you give select 0 as the inputs as the select lines to these two muxes.

So, now, you know that, if select if 0 is 0, it will pick a from here and it will c from here, which is what we want. You look at this, if select 0 is 0, then either a or c should come, if select 0 is 1, which means the last 2 bit are 1, either b or d should go to the output; that is what we are doing with this part of the circuit. So, we give the same select lines here, if

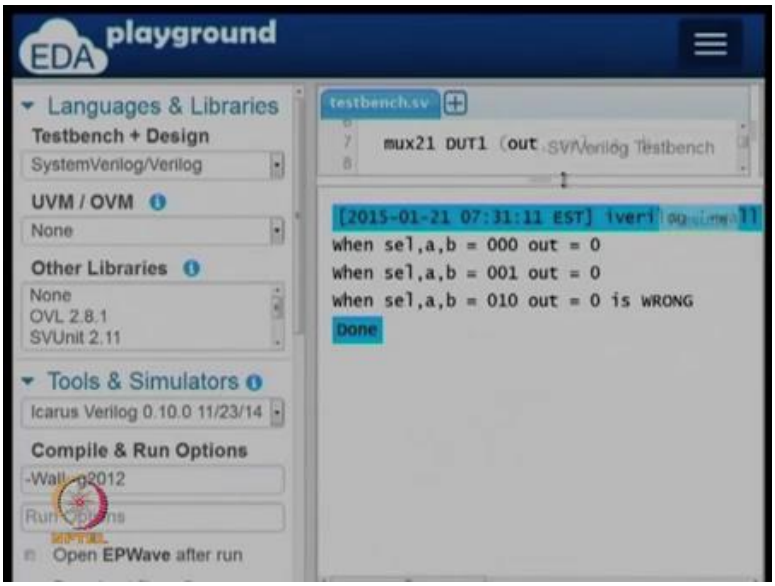
select 0 is 0, a and c will appear here, if select 0 is 1, b and d will appear here in these places.

Then, we have another 2 to 1 mux, this 2 to 1 mux is taking select 1 as the select line and if is is a here, it would b here, it should be c here, if it is b here, it will pick d here and it will pick one of those and pass it to the output. So, now, I already show this in earlier class, that this actually a valid implementation for a 4 to 1 mux. So, if I want to design a circuit like this which actually uses 3, 2 to 1 muxes to design a 4 to 1 mux. So, I need to be able to do that.

So, imagine, I ask you to build a circuit in a lab, where you have 3, 2 to 1 multiplexers; I want it put it to the breadboard and so on, then that is what, we are trying to simulate. So, when you do this, usually you go and give names to the multiplexers here. So, for example I am going to call this multiplexer M 1, this multiplexer M 2 and this multiplexer M 3.

So, the reason why we want names for that is, we want to able very refer to them by the names and say that, multiplexer M 1 is inputs are a and b, multiplexer M 3 is inputs are two intermediate wires and so on. So far, if I look at is a block box it is taking at a, b, c, d select 0 and select 1 as a input, it is giving out as an output. Internally, I have some structure. So, again I am going to something more, I will need a wire for this. So, I will call these temp 2 and temp 2. So, let say this is a circuit that you will draw, if I ask you to do it on a piece of paper, you will do this in your circuit. So, what I am going to do now is, I am going to write a verilog module for this.

(Refer Slide Time: 30:43)



So, let me open a new window, in this window I am going to write the verilog code. So, what I am going to design, let me just make the font size is much bigger. So, as I mentioned earlier, I want 2 to 1 mux which has to be instantiated and we want 4 to 1 mux out of that. So, the first thing I should be able to that is, I should first of all description for 2 to 1 mux.

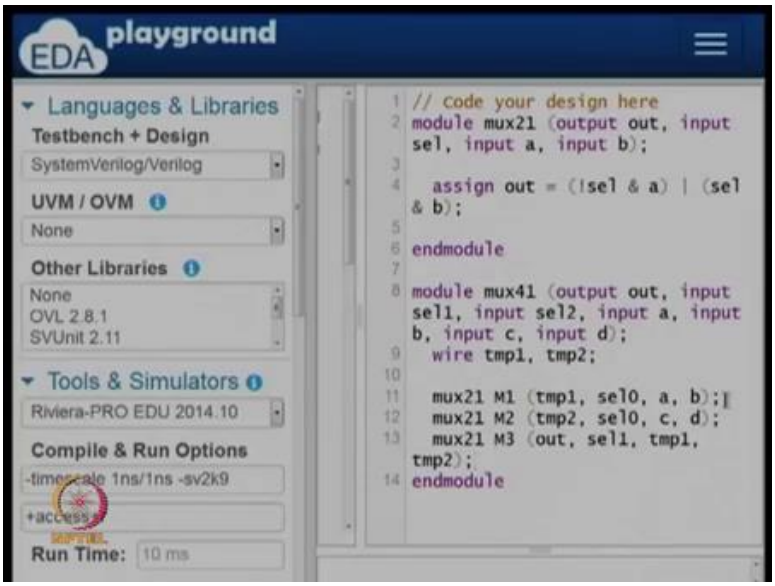
So, l can always go back and pick it from the other design there which I will do now. So, I already had a description for the 2 to 1 mux, I am going to copy that. So, the 2 to 1 mux is still using a Boolean equation, it is as it was a before. However, for the designing 4 to 1 mux, I am going to do something else. So, I am going to design a 4 to 1 mux, this 4 to 1 mux will give one output call out, as it was before. And it is suppose to be other few other things, it suppose to take two inputs select 1 and select 2 and it is also suppose to take 4 other inputs a, b, c and d; so this takes care of the black box description.

(Refer Slide Time: 32:00)

So, if you look at is a black box, it has four regular inputs, two select inputs and 1 output, we have this. What we need now is, we need way to describe the wires that we have and the description that we have. So, I said, we already need two wires temp 1 and temp 2, I will declare them now. So, what we have is, we have a 2 to 1 description, mux description for 2 to 1, I am going to design 4 to 1 mux using 2 to 1 muxes.

So, let see, how to do that, to do that, so I need to what is called instantiation. So, in the primitives, I said this is equivalent to pulling up gates and so on. But, mux to 1 is not a primitive, we designed it right now. We need a mechanism to say that, I want a mux to 1 called M 1, which will take select 0, a and b as inputs and give temp 1 as the input. So, will this statement do, so the meaning of this statement is, I want a device called M 1, this devices of the type mux to 1, it is a 2 to 1 multiplexer. And in the 2 to 1 multiplexer, I am going to take my inputs a and b and select 0, pass it as inputs to the 2 to 1 multiplexer. Whatever, it is giving as output I am going to call the temp 1 and this is a wire that is currently dangling.

(Refer Slide Time: 33:51)



So, if you go back to the picture ((Refer Time: 33:51)). What we have is, we have taken a of 4 to 1 mux, given it as input to 1 to 2 to 1 mux and we have taken b of the 4 to 1 mux, given it as another input to the 2 to 1 mux. And the select 0 is given as the select line and the output of that is call temp 1, we call this gate M 1; that is what we have so far. Now, I need another 2 to 1 mux. So, verilog does not stop you from doing this.

So, what we have done is, we have done what is called instantiation, an instantiation means I have a particular hardware object in this case is a 2 to 1 mux and I have picked up one copy of it. So, I bought 1 mux, 2 1 let say. So, the first one I have is called M 1, the second one I have is called M 2 and I am going to declare. So, to that mux, these are the inputs, it will take same select line it takes the c and d is the inputs and it gives you temp 2. So, temp 2 is the other wire.

Now, we have wires and this has to be connected to the third wire, the third multiplexer. So, third multiplexer gives the circuits output as a black box out is the output of the 4 to 1 mux and what is the final 2 to 1 mux suppose to do. So, let us look at the picture again ((Refer Time: 35:20)). So, M 3 takes select 1 as the input and temp 1 and temp 2 as the 2 inputs, select 1 as the control input and the select input and the output is the driven out directly which is what I am going to do now.

So, we have out which is the output, we can see that here. The select line is suppose to be SEL 1 now and we are passing on temp 1 and temp 2 as the other two input lines. So, the order in which we have given things are I have to be very careful. So, a and b are given

in that order, c and d given that order and temp 1 and temp 2 given in this order. There is the reason for that, because unless you do this, if you mix up a and b, then your 4 to 1 description would be incorrect.

So, we want when select 1 is 0 and select 0 is 0, we want a and when select 1 is 0 and select 0 is 1, we want b and so on. So, now, this is a design description for a 4 to 1 mux. So, what I am going to do is, I am going to provide you the code for the 2 to 1 mux and I will give you the test bench for it, for this also I will give test bench. I suggest that, you go and take this and write the program yourself, just like what you did for homework in quiz 3 at the end I gave a practice problem.

Just like that four week four also you do a few practice problems, I will give these verilog codes as well as I will pose some new problems and you can go and write it yourself. So, for this the test bench that I am going to give is very similar to the test bench that I growed for mux 2 1. So, I am going to instantiate this mux 4 1 into another test bench and in that test bench, I will pass various inputs and so on. I will get the outputs, I will compare the outputs and I will see whether it is expected or not, which is what I am going to do for the…

I will put that test bench code on the platform and you can see these four verilog files in the platform, you first get familiar with running it and so on. So, again even if you do not understand what the test bench is doing at least the design description must be clear to you. So, we design we describe what is called mux to 1 and we used three copies of this mux to 1 to design a mux 4 1 or a 4 to 1 multiplexer. That is what I want to you to capture.

So, far we have seen three different kinds of statements, we have seen something called primitive gates. So, we said AND, OR, and so on, those are primitive things. Then, we saw something called an assign statement, an assign statement usually appears in the module directly and you have something on the left side equal to some equation on the right side. And finally, we also saw how to pick various that are already present which you have design and how to wire them up to gather.

So, these are actually three different levels of abstraction for verilog design. So, I will leave the video at this point. So, at this point, you have the basic infrastructure to run verilog and I suggest that you go and take the verilog files and run it yourself and check whether the outputs as expected or not. So, if you want to try it go and write your own

mux 4 1. So, you can delete the mux 2 1 and you can write a Boolean equation for mux 4 1 and check whether the Boolean expression are correct or not. So, this brings me to the end of the video.

Thank you very much.