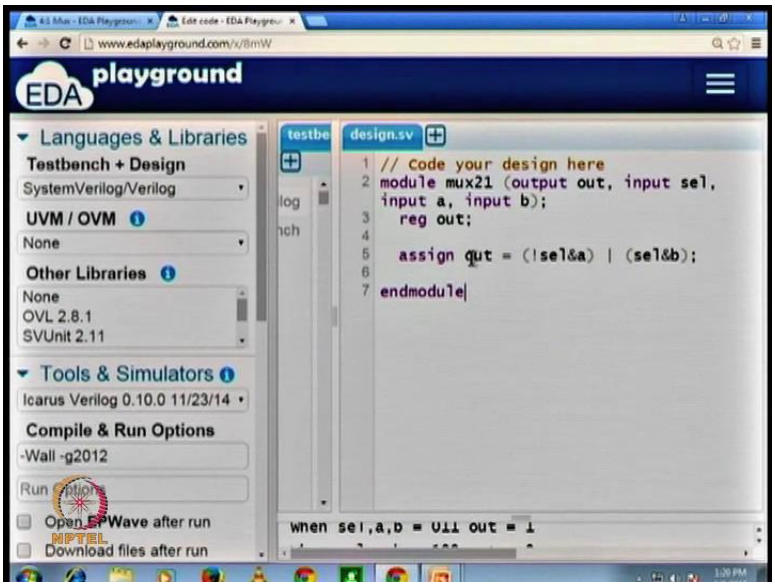
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 28
Always Statement plus Mixing Styles in Verilog

Welcome to the last week for this module, we are at module 28. So, in this module, we are going to learn a bit of verilog again, and what we are going learn is something called an Always Statement. This is a very important category of statements in verilog and I am also going to show you, how to mix different styles. So far, what we have seen is, we have seen how to instantiate basic primitives, how to use basic primitives to imply logic gates.

We have seen, how to use assign statements and we have also had cases, where we actually instantiated sub blocks and made bigger blocks out of it. For instance, when we did multiplexer, we designed 4 to 1 multiplexer using 2 to 1 multiplexer. So, this is the fourth style of statement called the always statement. So, I am going to talk about always statement for a while and I am going to talk about, how to mix different styles in verilog.

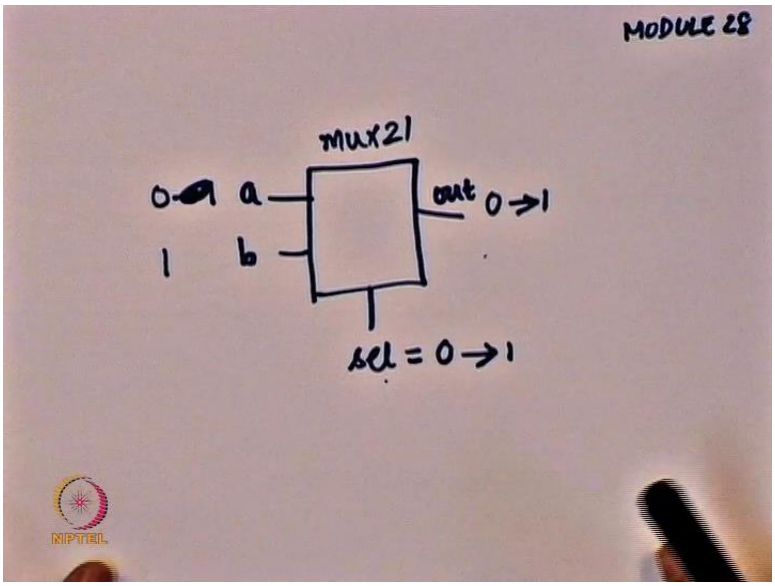
(Refer Slide Time: 01:14)



So, let me switch to the design. So, this is a design that you have probably familiar with. So, I have a module called mux 2 1. There is an output a single bit output and there are 3

inputs, select a and b and we have already done this. Assign, how to equals NOT of select AND a OR select AND b, this is something that we have already done. So, if we think about this, what happens is, any change in the right side of a, right side here on a, b are select will in some sense, trigger this statement to get executed and the value that you get from this evaluation is assigned to the out signal.

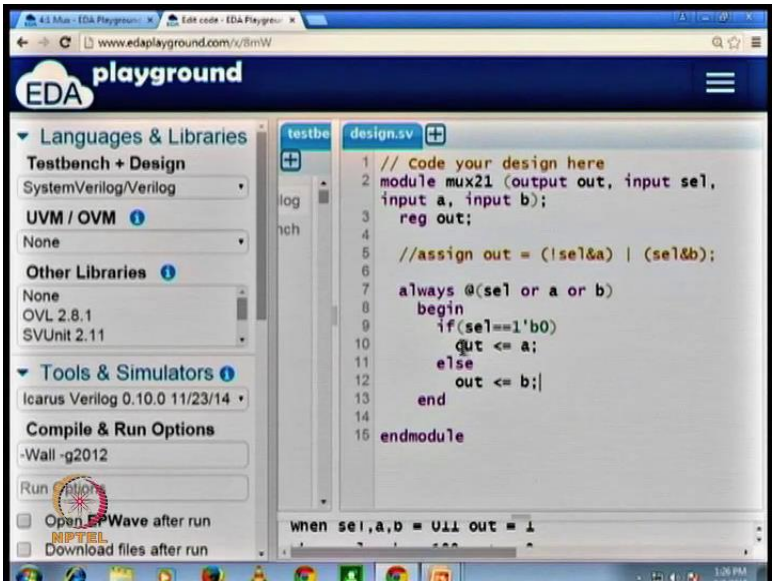
(Refer Slide Time: 02:05)



So, if you look at the picture here, what we are having is, we have a, we have b and we have the select line and we have output. So, this is our mux 2 1 and what we want from any combinational circuit is that, any change in any of the inputs should potentially change the output. Remember, combinational circuits do not have memory, so the current input must be able to drive the current value of the output, which means any change in any of these things should change the output.

For example, let us say select was 0 and a was initially 0 and b, let us say was a 1 and if a went from 0 to 1, you would expect the output to go from 0 to 1. Because, select is 0 and a is going from 0 to 1 without changing anything, then the output should still get excited and move from 0 to 1. Or let us say a and b, where 0 and 1 and select goes from 0 to 1 in which case, you want b to be selected; again the output should go from 0 to 1. So, there is one memory here, these inputs are staying as they are and one of the inputs is changing or one or more inputs can change. You would want the output to react to that and that is the definition of a combinational circuit.

(Refer Slide Time: 03:33)



So, combimbational circuit is essentially any change in input should reflect as a change in the output and that is what we have in the statement here. So, assign out equals NOT select AND a OR select AND b does exactly that. So, there is another alternative way to do that using the always statement and this always statement is lot more useful, when you design sequential logic.

But, let me talk about combinational logic and I am going to do the same circuit, I am going to design the mux 2 1 instead using a always statement. So, always statement starts with the keyword called always. So, we will look at this word always in a little while and what we have is, I am going to write down the syntax, we will come back and look at what each of these things are. What do we want the always statement to get, what do we want the design to do. If there is any change in a, b or select, then at the combinational circuit in some sense should have a revaluation of the output.

So, what I am going to do is, all these three signals namely select or a or b, if there is any change in any of them, then we want the output to react to it. So, I am going to put them in a list called the sensitivity list. So, we have a sensitivity list here and the sensitivity list consists of 3 inputs here, sel, a b, these are the 3 inputs. And if there is any change in any of the inputs, then we want the output to react to that; that is the meaning of this always at.

So, this at is, if you press shift and you press 2; that is the symbol here. So, in most keyboards, you will see it as shift and 2. So, always at sel or a or b and as usual, I always

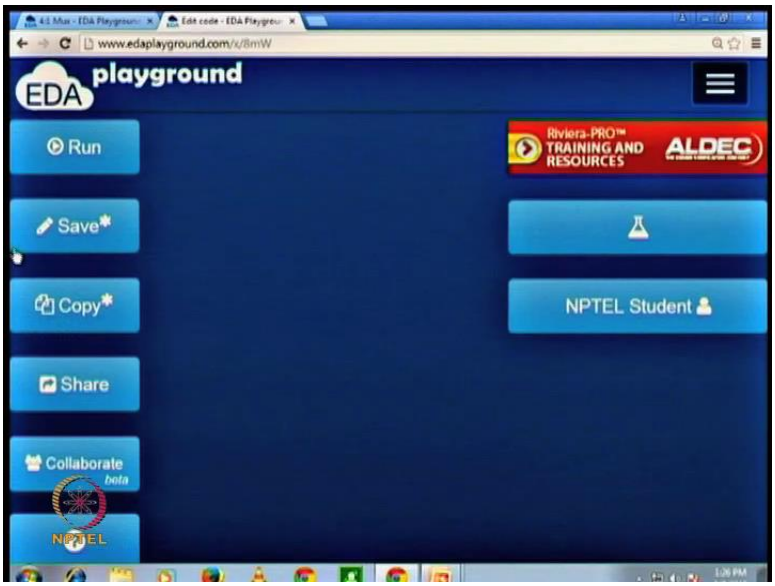
put the begin and end, before I do anything else. So, now what we have is, we have a list and I know that, this always block, so we will call this always block is taking 3 inputs select a or b and it supposed to process the inputs.

So, now I am going to introduce you to a new statement called the if statement, this if statement is very similar to what you have in C or C plus plus or any other language. So, I write it down and you will see that, it is a quite easy to understand, if select equals 0, then we want out to be equal to a, else out to be equal to b. So, let us see, what the statement is. So, now, I have the complete module, if you believe me or not, this is the complete module, I have the module description done.

So, always at select or a or b, there is a begin and end and I have written something inside that, there is if and then an else. So, let us see, what this is what the meaning of each of these things are. So, the first line that you see has a keyword always at what is called a sensitivity list and it ends with an end. So, there is a begin and end and there is a if statement inside. This if statement itself has, it is like any other programming language, if some condition, then some statement else some other statement.

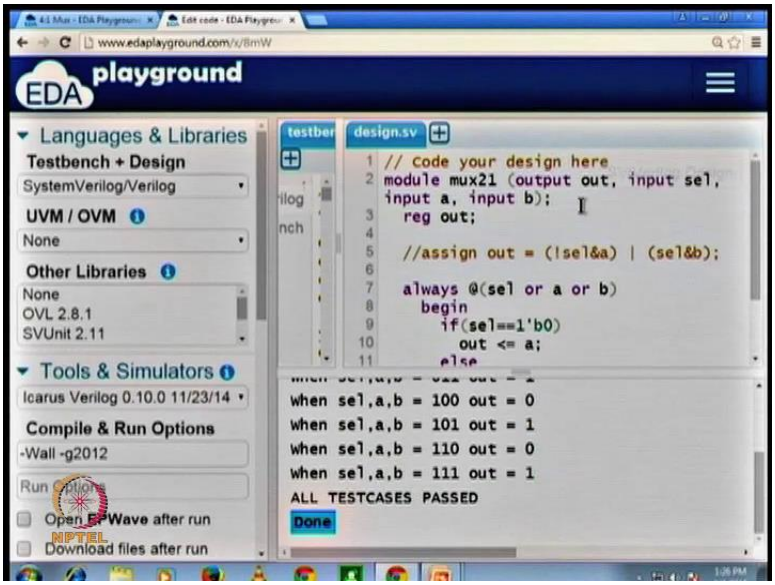
So, now let us see, how this whole thing is supposed to work. So, as a module this mux 2 1 has input select a and b and it has an output called out and if there is any change in select or a or b, then this statement is executed. If there is any change, if there is no change, then this statement is not executed. So, stimulatory is a piece of software, when we simulate this, if there is any change in select or a or b, then this statement gets executed and the value that you get out of this is assigned to out.

(Refer Slide Time: 07:55)



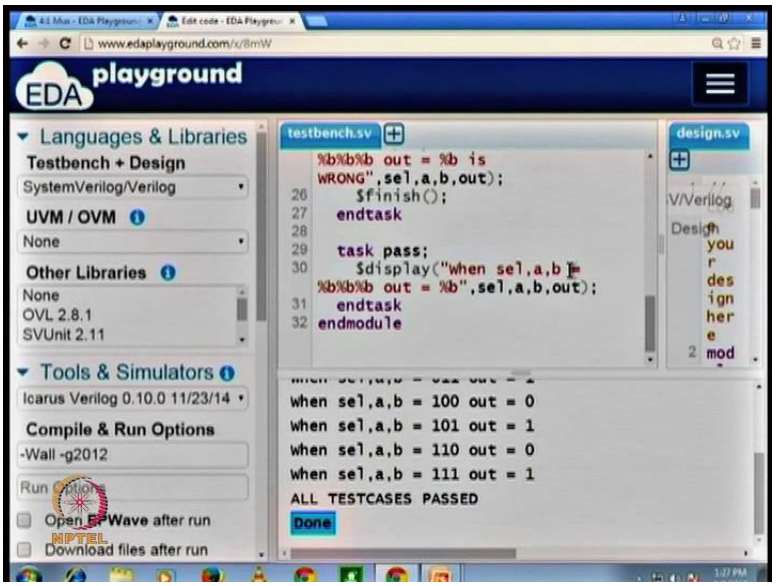
So, let us try and save it and compile it now. So, I will save this and I will run it.

(Refer Slide Time: 08:00)



So, I will run it once more just in a case.

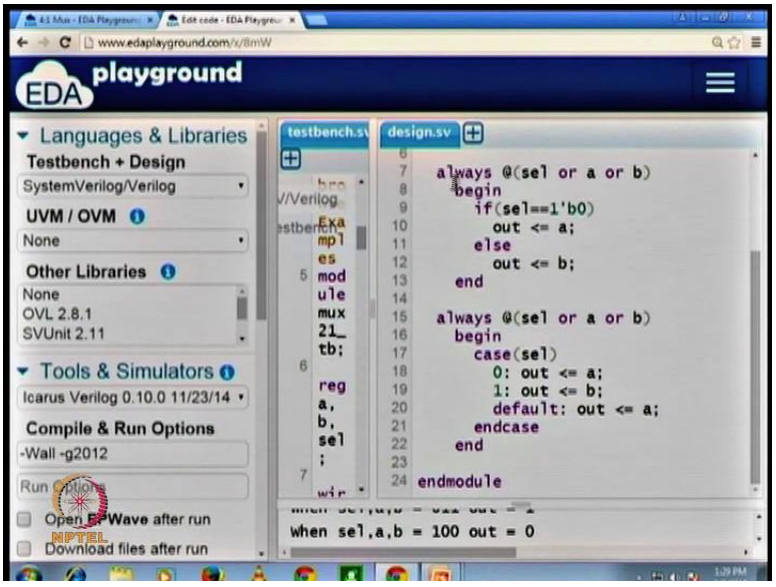
(Refer Slide Time: 08:12)



So, if we look at this, when select a comma b equals 0 0 0 out to 0, when select a comma b equal 0 0 1, out to 0 and so on. So, this essentially at the end, we also have this test case check. So, this is the same test bench that I used earlier. So, this is the same test bench that I demode in the class earlier, I think it was in week 4, module 21, I believe, I am not sure about the module number, but this is the same test bench then.

So, what we have is, we replace the assign statement with something called a always block and this always block is very fundamental to verilog. If you want to module sequential circuits, we can actually do it only with always block, you cannot use assign statements to deal with sequential logic, you need always block for that.

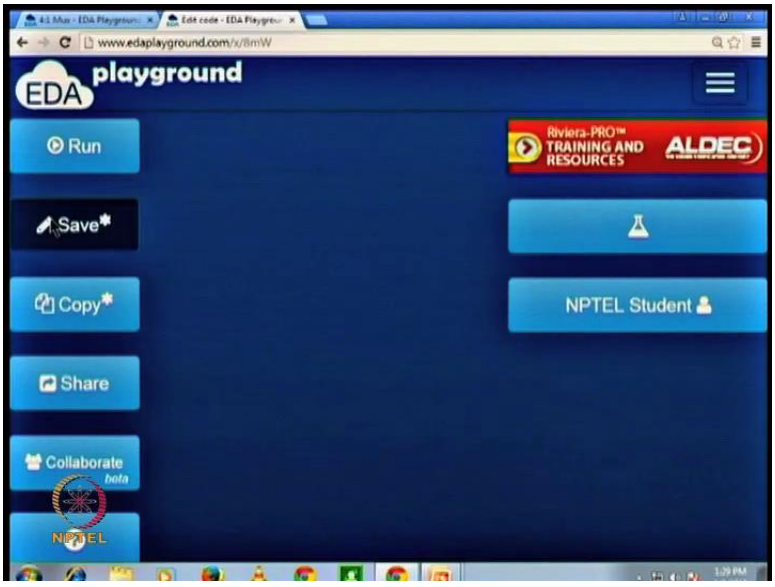
(Refer Slide Time: 09:00)



So, I am introducing you to always block in this module and I am going to do this same thing in a slightly different way. Again, using an always block, always at select or a or b all again with begin and end, there is also a case statement inside in verilog. So, case select, when 0 out equals a, let me try and speak this small, out equals a, when 1 out equals b and verilog, there is a clause for default and by default, I am going to a. I am going to assume that by default the input is 0 itself, I will assign that and case has something called an n case.

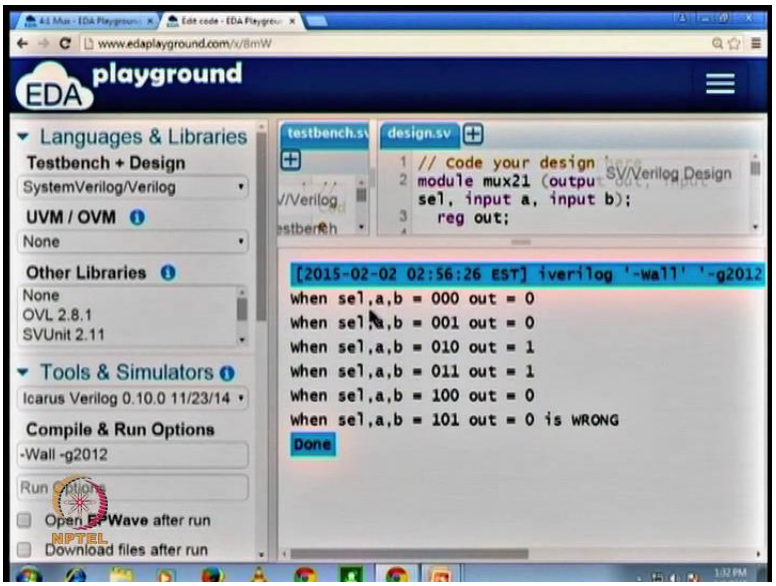
So, my claim is that, the always block that you see here and the always block that you see here are actually equivalent to each other. So, both of them take select a and b, these 3 inputs in the sensitivity list and this statement, the case statement here is equivalent to this, if then else statement here.

(Refer Slide Time: 10:31)



So, let me comment out this part and show you that, this part actually works and it is correct, let us check that. So, I am saved it and I am going to run it now, let us run it.

(Refer Slide Time: 10:41)



And again as before all the test cases passed and this always block seems to be correct. So, this always block that I wrote here seems to be correct. So, one thing that you want to notice is, the sensitivity list has sel, or a, or b. So, this or is not the same as bit wise or and this is not even the same as logical OR; that you will use in conditions. So, this or is used in the sensitivity list think, you can think of it as a list of signals, the list of inputs that this always block will get existed upon, that is all.

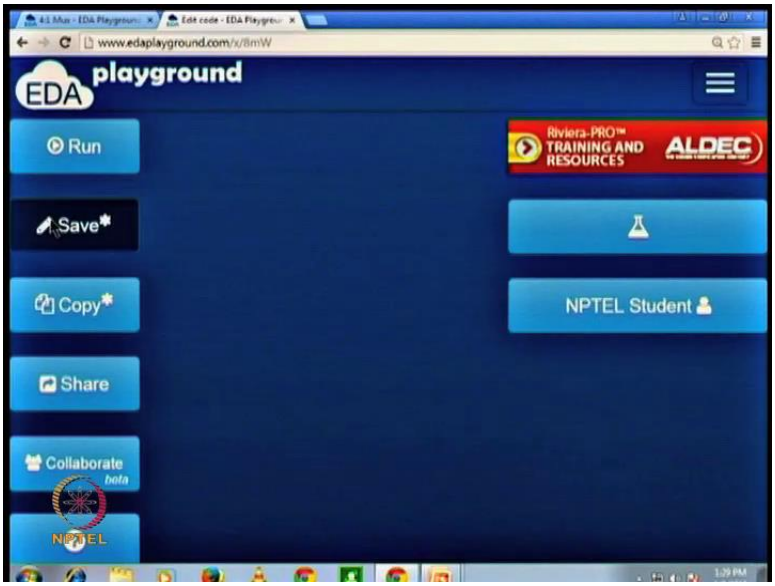
Do not attach any more meaning to this or here, it is not doing logical evaluation of select a, b with a logical OR, it is not doing that. So, keep that in mind, you can think of it is a separator between these 3 inputs, select a and b. So, if there is a change in select or if there is a change in a or if there is a change in b, execute this statement, whatever comes out should be given to out.

So, when you execute this statement out either gets a or b and there is a default clause, this default class is actually safety net and I will talk about this safety net in more detail, when we handle sequential circuits. So, it is always a good practice in a case statement, if we are going to talk about some combinational logic, it always a good idea to have a default clause and in the default clause, I have put out equals a.

So, now, I want to show you a little bit of an interesting thing. So, I talk about the sensitivity list. So, going back to combinational circuits, any change in select a or b should make the circuit revaluate and possibly change the output. So, now, I am going to show, something really nasty that can happen if you forget things in the sensitivity list.

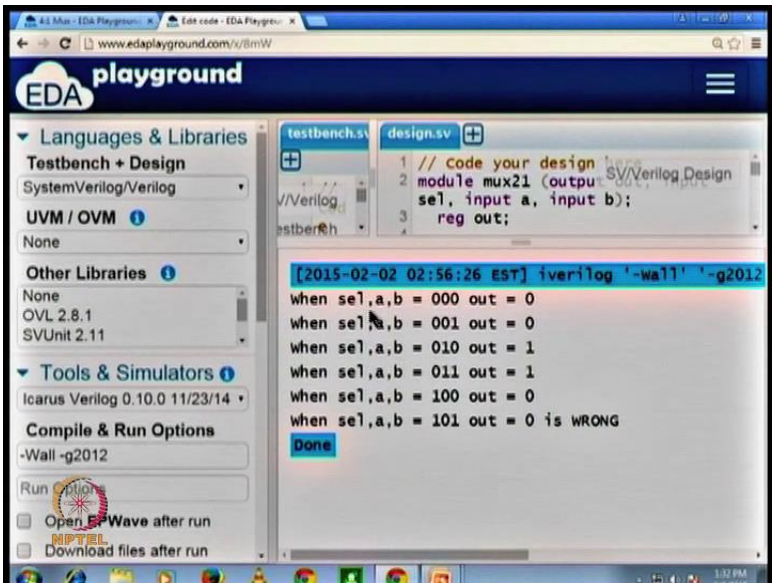
So, I am going to remove this b from the sensitivity list. So, the sensitivity list has only select or a. So, otherwise the code inside looks exactly the same, I did not change the code inside. So, it earlier past, but I have only change the sensitivity list now.

(Refer Slide Time: 13:03)



So, let me see, what happens now, I am going to save this and let me run it again.

(Refer Slide Time: 13:13)



So, something happen, so I will show you what is happen, let me pull this up. So, thing see imply they are working fine, when select a comma b is 0 0 0, out is 0, when select a comma b is 0 0 1, out is 0 and so on. So, we can check up to this line here that the output is actually fine. So, let us look at all of the select is 0 for the first four lines, so when the

select is 0 for the first four lines, output is suppose to follow a, so it is following 0 0 1 1.

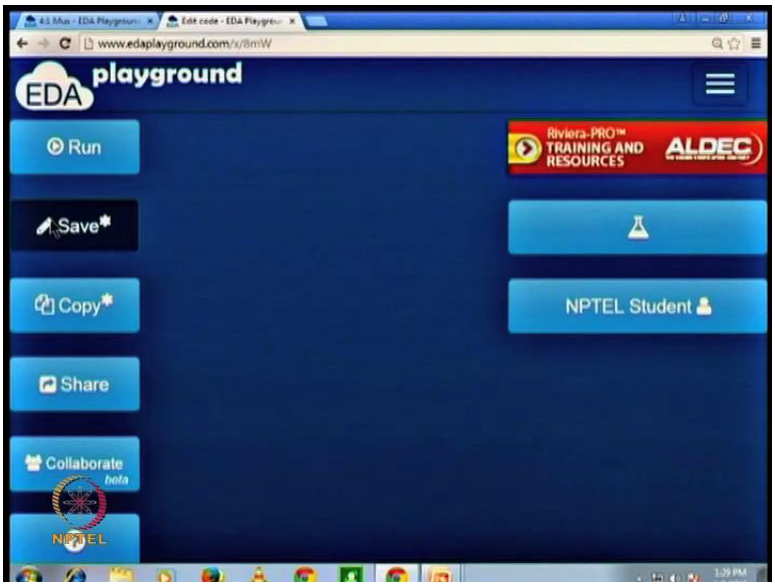
So, for it looks okay, let us look at the fifth line, when select comma a comma b is 1 0 0, out is 0. So, this looks like it is actually correct. So, when a comma b, so select is 1, you are suppose to choose b and it looks like, it is actually taking b and it is putting 0 there. Let us look at the next line, when select comma a comma b is 1 0 1, out is 0, it is wrong, it is wrong because, when select is 1, output is suppose to be b, which is actually 1, however it shows 0. So, there is something wrong here.

The reason that it is wrong is because in the sensitivity list, we have actually remove b for this statement. So, in a combinational block, any change in any of the inputs is suppose to effect a change in the output. So, let see this line, in this line, when the simulator evaluated it, select was changing, select was changing from 0 to 1, a was changing from 1 to 0, these two are enough to trigger the case, evaluation of the case statement.

So, it triggers the evaluation of the case statement and in this case out equal's p. So, out is 0, which was okay, if you go look at this line here, where it was correct, where it was last correct and if you look at the line here, where it actually went wrong, the only thing that is changed is t. So, select is remaining the same, a is remaining the same, b change from 0 to 1, when b change from 0 to 1 as a combinational circuit this mux 2 1 is suppose to have reacted to that.

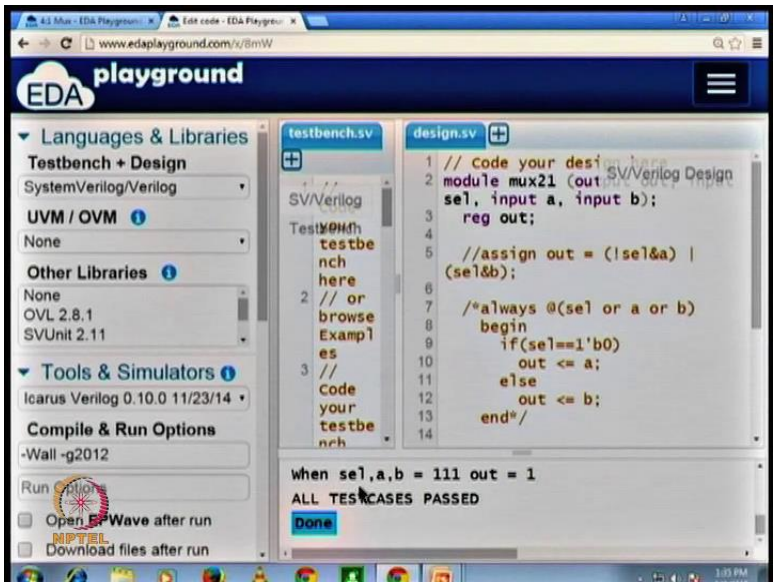
But, since b was not in the sensitivity list, it did not evaluate this case statement once more and the output was the previous output itself, which was 0 and this output equal to is actually not expected in the test bench. The test bench is actually expecting a 1, so the test bench is reporting that this case is incorrect. So, you have to be careful, you have to put all the signals for a combinational block, if we have implying a combinational circuit, you should have all the inputs in the sensitivity list.

(Refer Slide Time: 16:10)



So, I will go back and save it and write it once more, I ensure that it is actually okay. So, all the test cases are passed.

(Refer Slide Time: 16:15)



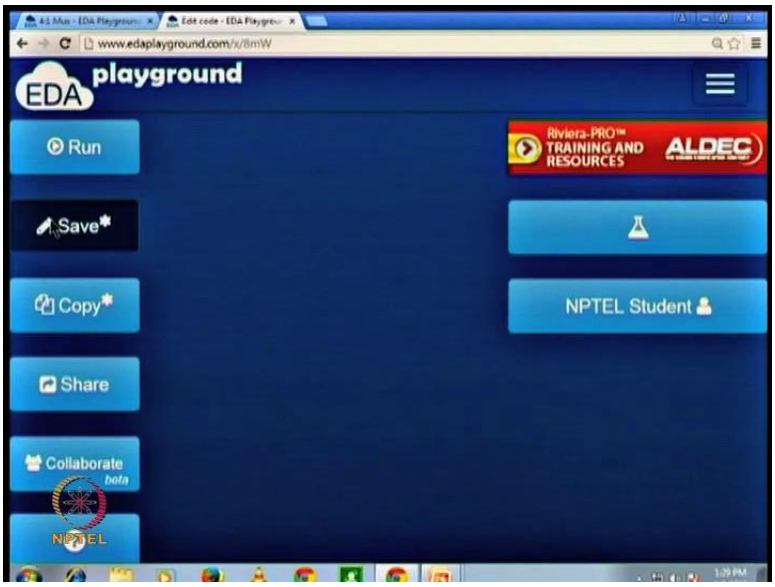
So, this is the basics of an always blocks, in always block can have statements which are actually called sequential statements inside and we will talk about the notion of concurrency and sequential statements in a later class. But, as of now, you know how to write an if then else statement and you know how to write a case statement. So, let us look at the next case. So, I said the other thing that I will talk about is mixing different styles.

So, what I have done is, I have written a mux 2 1 which was the original thing, which had an assign statement and I have mux 2 1 version 1. So, I have another module called mux 2 1 version 1, which seems to have if then else kind of writing and there is mux 2 1 version 2, which has case statement kind of writing. So, there are 3 mux 2 1's here, the first mux 2 1 is an assign statement is using an assign statement, the second mux 2 1 is using the if then else and the third one is using the case statement.

So, one thing you have to remember is this statements inside do not have the keyword assign. So, when we look at always block, always block will not have an assign keyword inside. So, this equality here, this equal symbol here versus this is assignment here, they are actually different things, we will get in to the semantics in a later class. But, you cannot put a keyword assign here inside the always block.

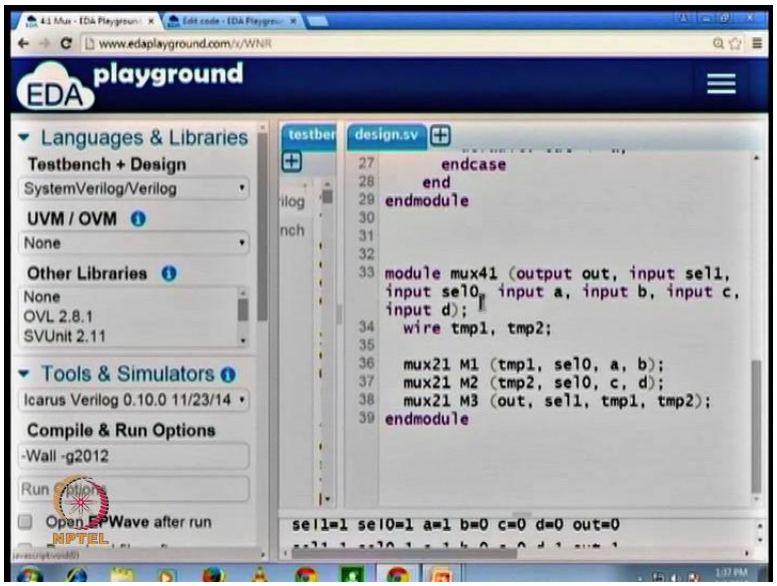
So, now, what we have is, we have three different ways of designing the multiplexer. We have a multiplexer which is design using a logical equation, we have another multiplexer design. So, it is a same mux 2 1, but design using if then else statement and again, another multiplexer design using a case statement. And there is a mux 4 1 here, this is something that I again showed in an earlier module and this mux 4 1 is a 4 2 1 mux. It seems to have three different muxes; M 1, M 2 and M 3 and they are all now at the type mux 2 1. So, if we go back mux 2 1 is use the assign statement.

(Refer Slide Time: 18:31)



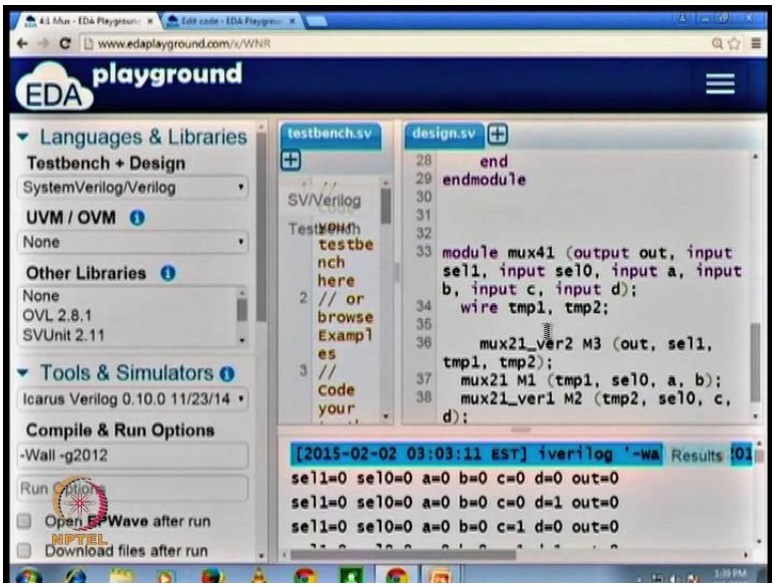
Now, let us first run it and check whether it works.

(Refer Slide Time: 18:34)



So, I wrote the test bench for it. So, I did not show it you in the last class, but I have a test bench which I can put online, you can check it later. So, this test bench is not checking for whether things are correct or not, it is just printing the output and believe me, it is correct.

(Refer Slide Time: 18:53)



So, what I want to show is that, these three multiplexers, the first thing I want to show is that, it looks like the there is some order in which we have declared all these declarations, we have done all the declarations. So, this mux 2 1 M 1, mux 2 1 M 2 and mux 2 1 M 3, it looks like the designed M 1 mux first, and then instantiate the M 2 mux, and then we have instantiate the M 3 mux.

So, if you come from a sequential programming language like C or C plus plus, you have to assign values before you use them. So, it may look like temp 1 and temp 2 has to be assign values before giving it to mux 2 1 M 3. What I am going to do now is, I am going to take that, I am going to put it on the top. So, I want to show that, first of all, the order of statements inside a module does not matter in verilog.

((Refer Time: 19:51)) So, I will moved it up and I am going to run it once more and did not really change anything. Again, you can verify this locally yourself, there will be several lines here, there are 64 lines in which various inputs and outputs are shown. The first thing is he did not complaint about this temp 1 and temp 2 being different. The temp 1 and temp 2 are suppose to be driving M 3, but I put declaration of M 3 before the declaration of M 3 before the declaration of M 1 and M 2; that is actually perfectly fine.

So, what we have is all these three statements are actually instantiating gates and what we have done is, we said we want these three gates and we want the connection between these gates. The order in which we have to actually put them is not specified and it is perfectly fine. So, that is the first one that I want to talk about, the second thing I want to talk about is, you can actually mix and max tiles.

So, what I am going to do is, I am going to use version 2 for M 3 and I am going to use version 1 for M 2 and the assign statement version for M 1. So, what I have done is, I am actually instantiated three different multiplexers of different kinds. So, M 3 is going to use version 2, which has this case statement; M 2 is using version 1, which has the if then else version and M 1 is using the assign statement.

((Refer Time: 21:20)) So, verilog allows you to mix and match different things and what we have is, we have what is called different architectures for the same design. So, if you know vhtl the mux 2 1 is an entity and what we have a three different architectures. So, verilog does not give you the same facility like entity and architecture like vhtl does. We have three different versions mux 2 1, mux 2 1 ver 1 and mux 2 1 ver 2. We have three different versions and in a single design mux 4 1; we have instantiated three different kinds that is perfectly okay.

So, this is another mechanism, we can mix and match styles that are appropriate for the design that you are looking at. So, I want to show that this is possible. So, it is a very silly example here, it is a very simple example. However, I want you to know that such a facility is available in verilog, verilog does not restrict you to only one form on the other.

All the forms if we have seen so far, namely using primitives, using always blocks and so on, can all we mixed.

In fact, I do not have to instantiate mux here, I can write an always block to do only one of these that would also still work. So, probably this is a little too much to digest right now. As of now, just trust me that there are three different instantiations here and these three instantiations are actually used three different styles; that is perfectly okay. Even, within a single module, you can actually have different kinds of block mix together.

You can have an assign statement, you can have some primitives and you can have an always block and that is perfectly fine, verilog will actually allow you to do that. So, in this module, what we have done is, we have seen two things. One is we have seen something called the always block and the always block is something that actually goes inside the module and you can actually have what are called sequential statements inside.

I will talk about what this notion of sequential is, this is not the same as sequential design. So, there is something to do with the order of execution of statements inside a always block and we also saw, how we can actually write different things and mix them together, when I talked about multiplex of 4 1. So, what we have done in from week 1 to week 5 in terms of verilog is that, we have seen how to write basic modules all the way up to the four fundamental ways in which you can write design. You can instantiate, you can use primitives, you can use always block and you can use assign statement.

In fact, when instantiate the underlying model, should use one of the three, should use either primitives or should use always block or should use assign statements. So, the instantiation is just a Meta thing. But, the underlying things are you can use assign statements, you can use always blocks or you can use primitives directly. So, what we are going to do is, we are going to see larger and larger designs done with these styles, different kinds of styles.

And from next week onwards, what are will talk about is, how to pick an appropriate style for the design that you have at time. If you have multiple choices like these, how do you pick the right way to do things? When, should I use if statements versus a case statements, when should I use equations like in an assign statement versus an always block. This is something that usually comes with practice, but what I want to do in this course is give it to you straight. So, that you start in the correct way, instead of experimenting with various things.

Like with any programming language, there are 100 of ways in which you can do things, but designers who work and make large designs do not fool around with verilog and try to play with different ways of doing things. Usually, I as a designer what I do is, I pick one way and I learn it well and I use it over and over, instead of figuring out all the 100 ways in which things can be written.

So, this brings me to the end of week 5 also by the way and in week 5, we saw lot of heavy duty stuff. We saw basics of electrical circuits, what are the fundamental electrical parameters, more importantly; I introduce you to the notion of delay's as well as state machines. So, this week is quite loaded and on top of that, I have also introduce some new topics in verilog, I suggest that you spend a lot of time this week in reviewing the material, because the quiz is also going one level up.

The kind of questions that are there in the quiz is also going to go one level up. So, till now, we are had quiz questions that you have fairly simple and straight forward. From this week onwards, you will see that the quiz is also drifting it is level up. I recommend that you actually go back in review the videos, use the forum very strongly, ask questions on the forum, I will be there on the forum along with my Ta's and there will be others in the class room also.

So, I suggest that you take a series look at the forums before you ask your own question as find out whatever other people are asking, what I am replying and what the Ta's are replying, before you shoot of your own thread. So, I am hoping that you will be enjoying course so far and I did look at the feedback that you given and I am waiting for many others to give the feedback. I will talk about the feedback itself in a later class and I am already beginning to fix many of the things that you have mentioned in the feedback so far. If you have not given feedback, I suggest that you go back to the Google spread sheet link that I gave you and give your feedback, because it is very valuable for us.

So, thank you very much and see you next week.