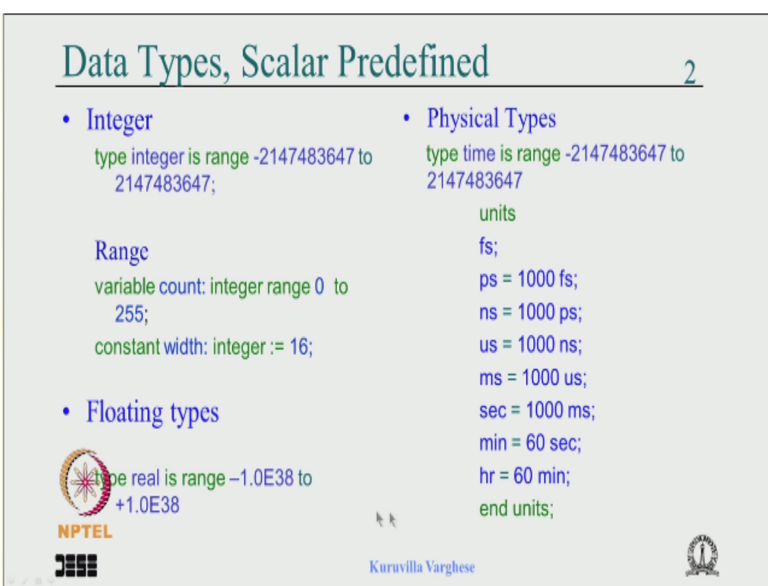
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science - Bangalore

Lecture-14
Concurrent statements and Sequential statements

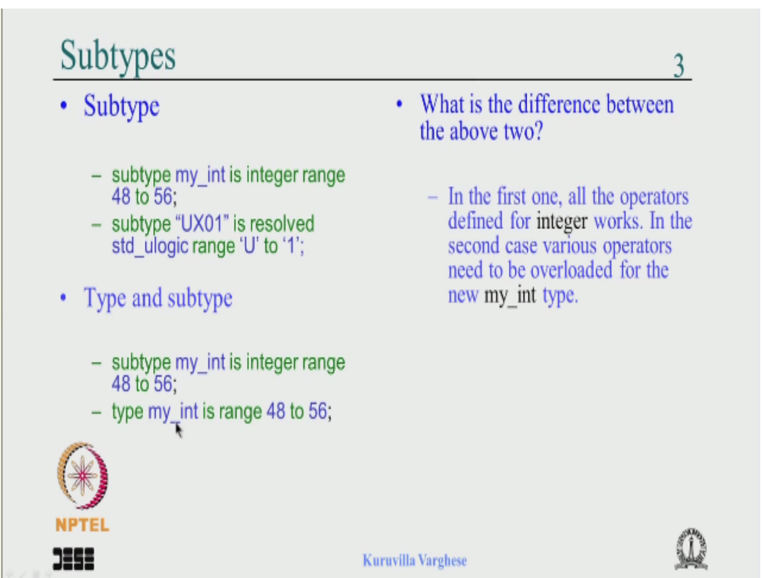
So, welcome to this lecture on VHDL as part of the course, digital system design with PLDs and FPGA. Before going continuing with the lecture we will have a quick run through the last lectures portions. The last lecture we have completed the data types basically the composite and the integer real scalar type and then we looked at the concurrent statement and with select statement. So, quick run through the slides. So, let us go to the slide.

(Refer Slide Time: 01:04)



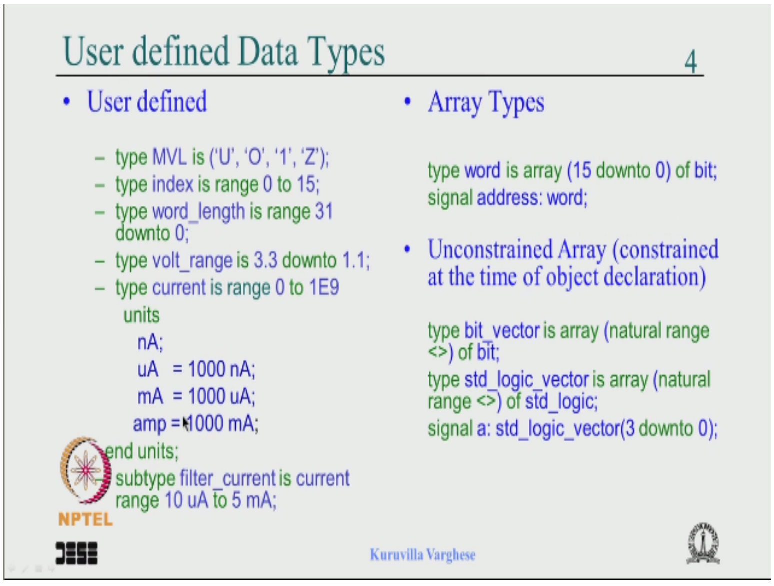
So, we talked about the data type integer. So, what is predefined in the standard library and how to use it, and the real type which is probably less use and a physical type you know the time with the unit as fento second, you can check what is the latest the VHDL standard kind of basic unit must be fento second only. So, that is the various units specified.

(Refer Slide Time: 01:38)



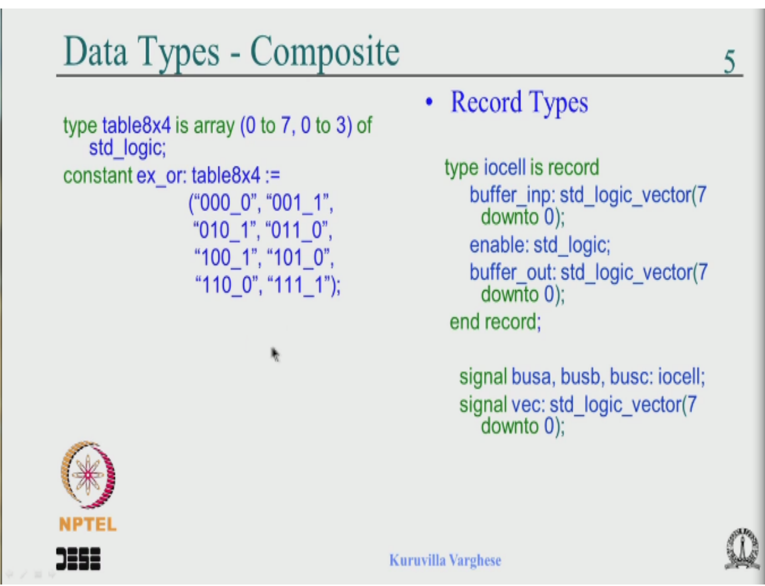
And we have it is possible to define subtype on a type. The advantage of defining subtype is that you can use all the operators. Unlike you if you define your own data type. Then you need to overload all the operators functions which you need to use for this particular data type.

(Refer Slide Time: 02:00)



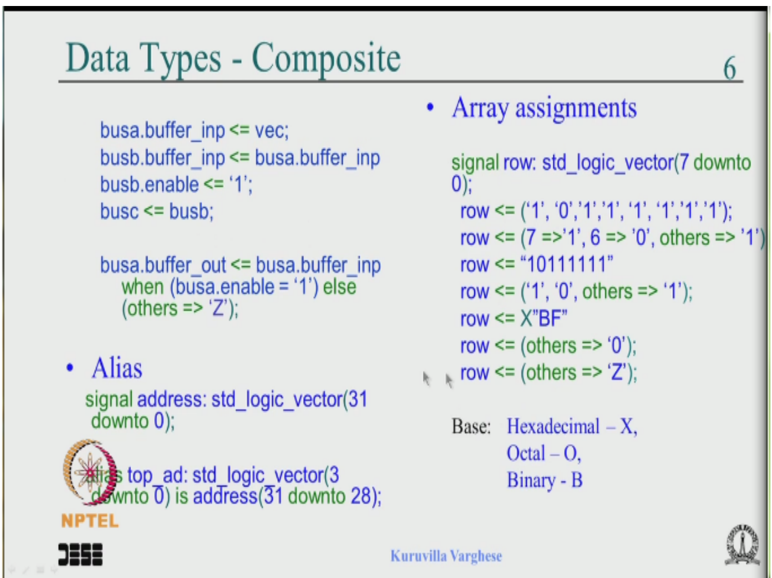
And, this shows various user define data type, an enumerated these are integer, these are this is real and this one is a physical data y and this the subtype. We look at the unconstrained array and we said that standard logic array is defined as unconstrained. That is why constraint it when we use it.

(Refer Slide Time: 02:26)



And you can a multi dimensional array. Any number of dimensions and we have seen an example of that .and you can have record whereby, you can kind of a you know related signals of variables put together and this show some input output enable of Tri-state gates are put together, and you can assign like record in structure in c.

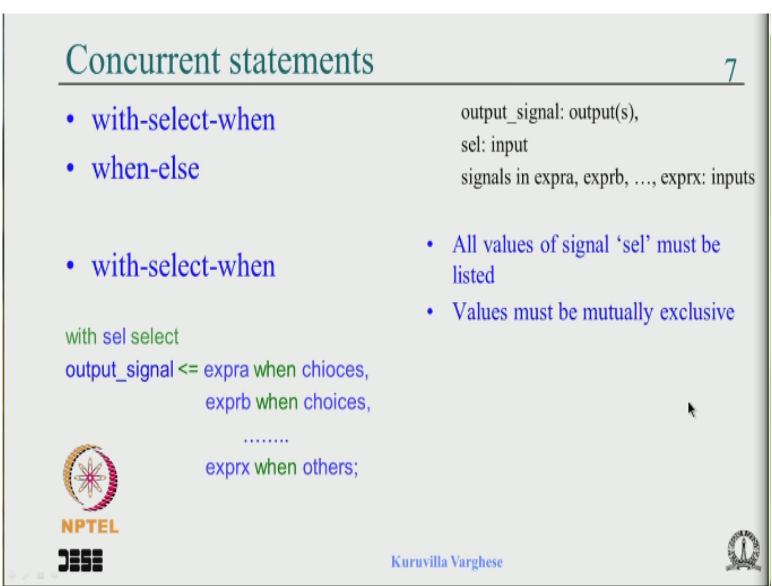
(Refer Slide Time: 02:57)



The similar syntax is use and you can see that the component individual component within is access with the dot, and if you have some signal, a part of it can be alias use the top address line of an 32 bit address is alias with the top ad, we have also seen various ways of assigning the rows, this is a positional you know kind of association. This named association, this is like a string and this one is again positional association with this others hexadecimal base specification.

And if you say others means that everything is you know one value. That in this case it is 0. The other case it is tri-state. So, this is very useful one because if you have say 32bit bus which you want to kind of output you know something is connected to 32 bit bus and you want to initialise it you want to tri-state you do not have to write you know keep on writing z at counting at 32 times and all that. Just say others that that the most useful form of array assignment.

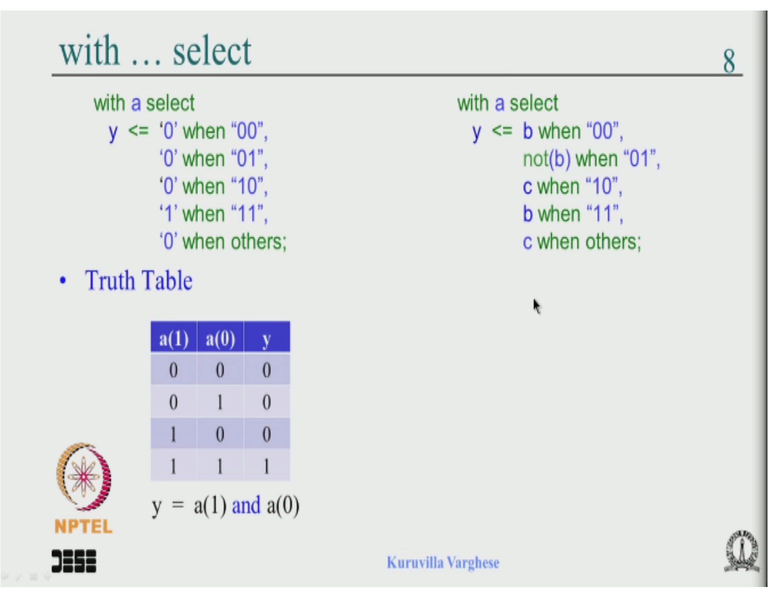
(Refer Slide Time: 04:16)



And, then we have looked at the important part concurrent statement. As a name suggest this is used in the concurrent body. That is in the architecture statement region. This cannot be used with in a function or a procedure or a process it has to be straight away used in the concurrent body or the architecture statement region. So, the syntax of with select when is with some you specify some input, use a select.

For all mutually exclusive values of that signal you specify the output the numerical values or as function of some other inputs okay. So, we said that is nothing but a truth table.

(Refer Slide Time: 05:01)

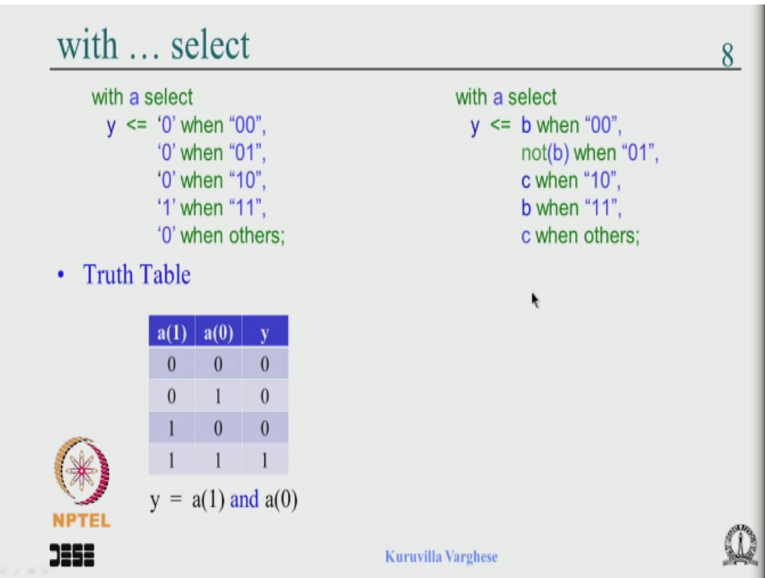


In the simplest case we have seen an AND gate, say a1 and a0 and for all the 00, 01, 10, 11. The value the output is specify just single truth table. The equation is wherever there is 1, so that mind-term as a the equation or if you have multiple one, then you say O or if you a two 1. Then you say or and that so, you wherever 1 is there, you pickup that midterm or the next one and so on.

And this we write when others to do for the simulator to completely specify all input combination. Because standard logic can true bit standard logic can take 9 into 9, 81 values. And we are only specifying 4 of them. So, rush seventy seven is you know covered here. So, you could even say 1 when others. But, as I said you know specifying some known values will help you debugging.

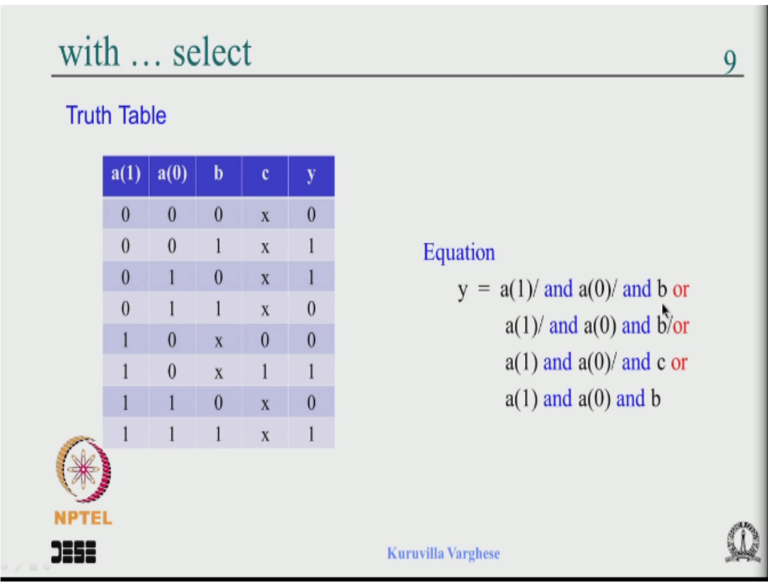
Like you know that in some of those kind of combination occur. Then the 0 can happen. Even if you can say tri- state does not matter. Because it is useful for debugging you know. and next we have the case where the output is specified for all the values of a input signal a. As a function of b and c okay and that is little more complex truth table.

(Refer Slide Time: 06:30)



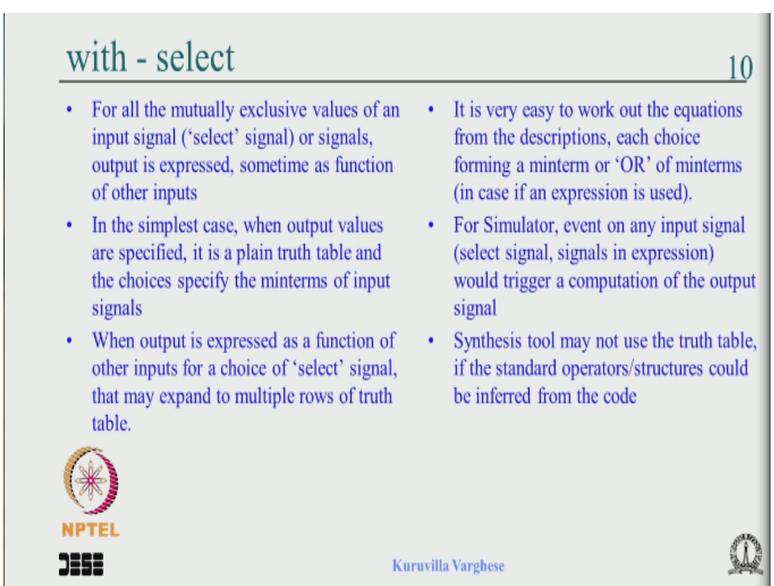
So, you see when a is 00, then y is b, so y is 01b, 0y is 1. When these 1 and so on. And the c is do not care. So, this expands in a multiple rows of the each one is expanding into kind of 2 rows of the truth table.

(Refer Slide Time: 06:49)



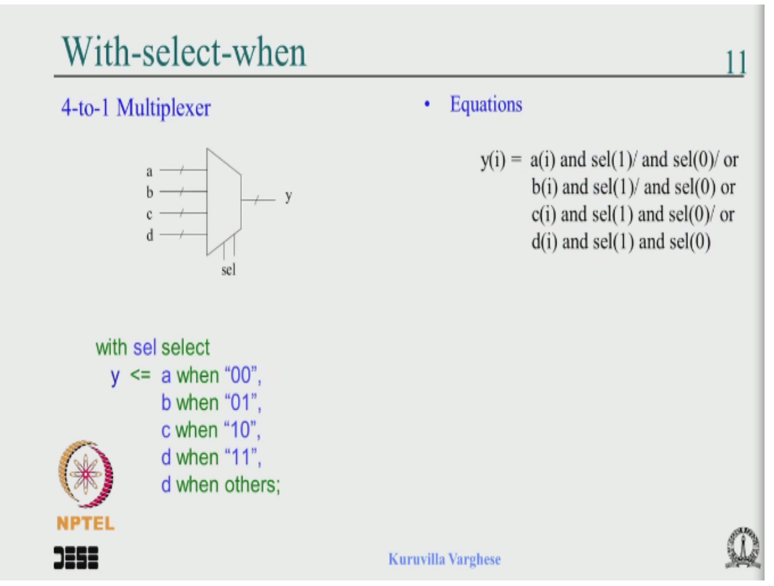
But, when the equation is written it is kind of a1 bar, and a0bar, and b or like. So, this is straight away this choice is a1 bar, a0 bar and b is y or a1 bar a0 and b bar and so, each one is a midterm or a product term or this and so on it goes the equation. So, that is what is shown here.

(Refer Slide Time: 07:20)



At we have that same thing, and one thing to remember is that whenever there is an event on a, b or c, the simulator computes this. And as for as synthesis tool is concern. It will look at the you know look at the description and form the equation that is the simple as it is.

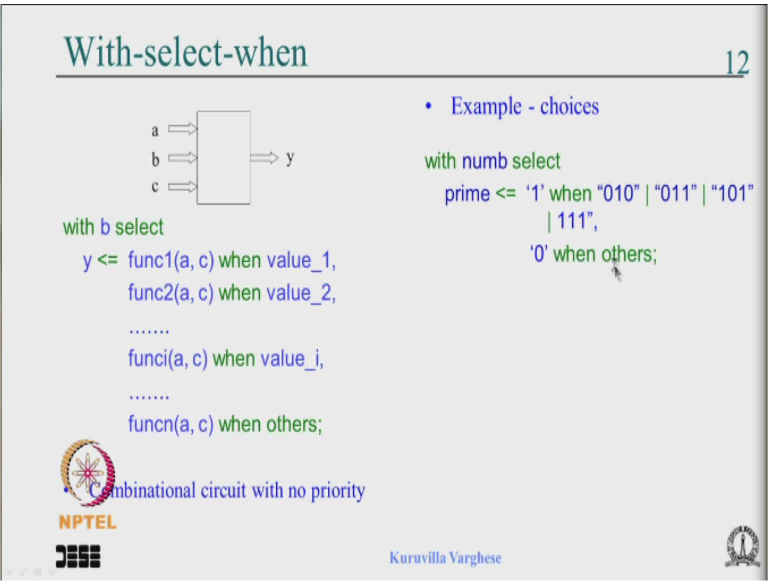
(Refer Slide Time: 07:45)



And this can be use for you know example is shown with the multiplexer. So natural choice for the select signal is this. In the select signal of multiplexer. So, 4 various combinations 00, 01, 10. Y is y is a b c d like 0, 1, 2, 3. And the equation is like this you know. The select 1 bar and select 0 bar and a or and so on okay. So, each one is a midterm or this one, this one, this one.

And I am showing only yi, so because for a bus you know, whether it is 4 bit or 8 bit. The equations are same you know exactly similar for each bit of it or because of it goes to the same type of AND or structure. And it is easy to confuse this kind of syntax with the multiplexes function, but and we have seen that.

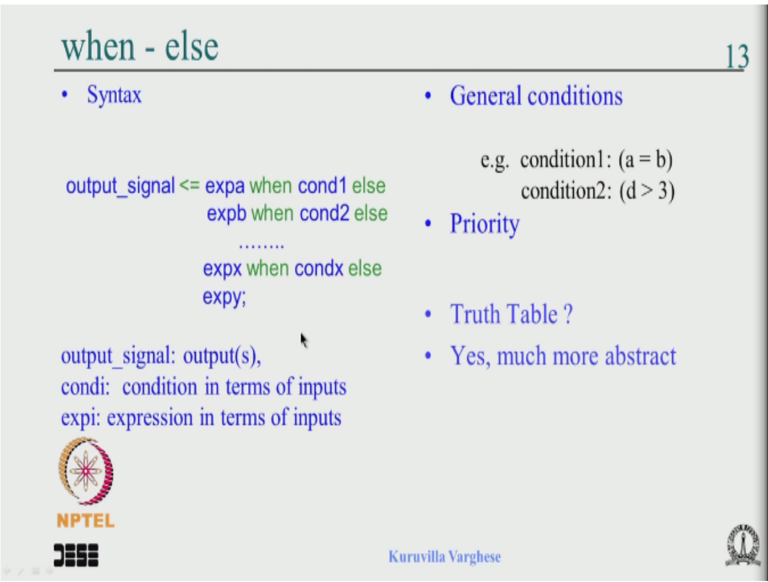
(Refer Slide Time: 08:45)



Any combination circuit without priority can be easily specified by with select. So, if you have multiple inputs one output. Then you pick up some input and for all the mutually exclusive values of b you specify y as some of a and c**.** Depending on your logic and that is what is with selects provide you. And this is there is no priority, actually this is for it is a plain truth table with little abstraction, maybe it will translate in you bringing a and c.

So, some more columns are added, some more rows come into picture. So, still a straight forward truth table and as I said if you have the output has a same function or same values for different choices of the values. Then you can use OR. And that is what is example say.

(Refer Slide Time: 09:49)



So, that is what is with the with select concurrent statement. So, let us look at the next one, which is when else which is little more complex than the with select. So, please have a look at this syntax. So, which says that you have some output signal it could be a single bit or multiple bit it does not matter you have an output signal which is assigned some expression and this can be a numerical value like 00, 01 depending on the bit or it can be some input condition, input expression.

You can say a and b or a bar or whatever okay. When condition 1, that means condition is again an input you know. You can say c is equal to 3 or p greater than q and so on. So, some condition based on the input. Then you say else which is not done the with select okay. Else some other expression when some other condition okay, else and so on okay and at the end we say an else which is that means everything else is expression y okay.

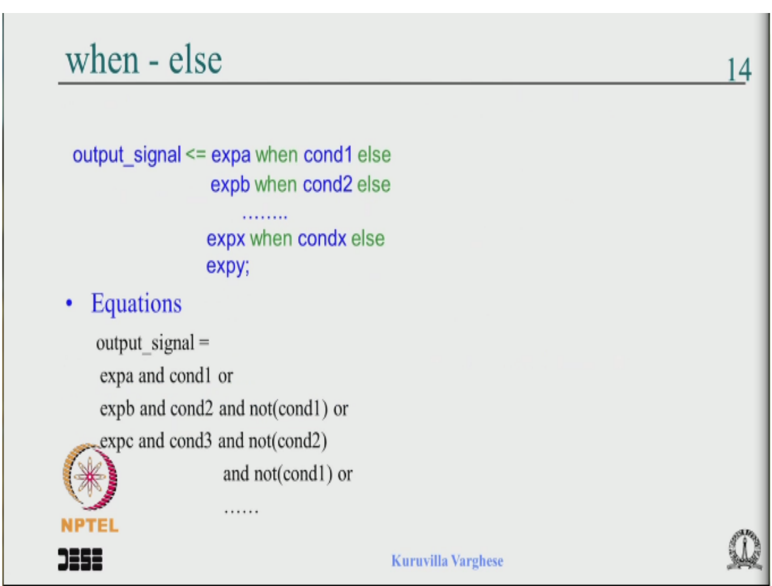
So, this naturally brings in the priority okay. That means we have saying output is some expression when some condition. Suppose we have saying a condition a equal to b and we say suppose the this is 2 bit this is we said 00. When a equal to b, then we say else it means a is not equal to b okay and we say another condition say d greater b. So, it means that the output is maybe this is 00, this is 01.output is 01, when a is not equal to b, and d is greater than 3 okay.

When you say else it means that it is not dis-condition and not dis-condition. That means you say a is not equal to b, and d is less than or equal to 3. Then maybe the output is expression 3 and some other condition okay. So, as you go down the not of all the previous condition is a coming into picture okay. So, it is it brings in priority first thing is note that is that there is a priority.

So it will little more powerful because you see this condition 1 can be in terms of 2 signals. And when it comes condition tool can be another signal, another group of signals and this expression itself can be some inputs. So, it is quite a powerful, quite an abstract statement. Much more powerful than with select, it brings in priority and most importantly the question is that e set are we completely specifying a truth table by this kind of structure or a syntax that is a question**.**

You think about it whether it is a truth table, the answer is yes it is a truth table, but it is in a much more abstract and I will show you soon, very quickly I will show you this is nothing but a truth table. The complete truth table is specify in terms of all the inputs involved that we will see in a moment, So, let us look at the how the equations are derive for the combination circuit from this syntax.

(Refer Slide Time: 13:50)



So, again I am not putting down some signal it is still little abstract. So, you have an output signal and expression a when condition 1 else, expression b when condition 2 else, expression c when condition 3 and so on okay. So, the equation comes like this, output signal is expression a

and condition 1. When I say I a condition 1 is equal to working out the product terms. The all the product terms of the condition 1.So, it can explain into multiple product terms okay.

So, it may be like you say d greater than c. There could be some you know product terms in terms of b and c or and so on okay. So, there could be multiple product terms. So, it still abstract but then the meaning is that the equation is expression a and condition 1 or and when you come here it is an nor of condition 1. So, expression b and nor of condition 1 and of condition 2 okay or when you come here it is expression c and nod of condition 1 and nod of condition 1, condition 2 and condition 3.

So, it brings you know it expands like that and if you remember your basic codes. You would have seen a priority encoder you will see there is an AND gate as it goes down to lower priority from the higher priority. There is an AND gate with lot of bubbles you know becoming bigger and bigger. So, that is similar thing it is happening there but there it is you know the single kind of normal is a single bits which is going to the AND gate. Nod of the previous inputs and so on. So, this is the same thing so it is quite powerful

(Refer Slide Time: 15:54)

p(1)	p(0)	q(1)	q(0)	r(1)	r(0)	y
0	0	0	0	0	0	c
0	0	0	0	0	1	c
0	0	0	0	1	0	b
0	1	0	0	x	x	a
0	1	1	0	0	0	c
0	1	1	0	0	1	c
0	1	1	0	1	0	b
1	1	1	1	1	1	c

So, let us see an example and I want to illustrate how it is specifying the complete truth table. So, let us take an example y is a single bit signal a, b, c are single bit like y is output single bit, a, b, c are inputs, which are also single bit. But these are inputs pqr are inputs. And there are 2 bits you

know. So, you have p1p0, q1q0, r1r0 and so on okay. So, you can imagine a truth table with p1p0, q1q0, r1r0 and abc and y okay.

So, imagine wherever the values of p is greater than q, irrespective of the conditions rabc, so b and c, y gets a. And when it comes other rules like p is less than or equal to q. So, wherever the roles in the truth table. Where p is less than or equal to q or conditions or rows. And r is equal to 2. We write y is b okay. Else for all other rows we write c. So, it completely captures a the truth table, but it is very powerful, I will show you the truth table so I have put it.

I have kind of compress abc, ideally I should have in the input section. This is a input section, this is output. I should have put abc also. But to safe phase I have included the output expression but it is easy to understand okay. So, in reality the truth table is much bigger than this. So, you look at this scenario where the p1p0, q1q0, r1r0 are the inputs, y is output.

So, look at this scenario where p1p0 is 01, q 00, r do not care. So, here p is greater than q, so the output is yj. As I said you should have a column, then if a0, yo a is 1, y is 1. So, it is simple you can write that. And when it is comes to this case you see p is 01,and q is 10. So, the p is less than q. Then r you look at the value r 10 this is 2, then as specify the y is b.

But in other cases where p is less than q and r is kind of not equal to o 2 the y is c. And you can populate this values all the way from 000, 001 all the way you know. That all the way comes to 11, and 11 you see is p is not greater than qr r is not equal to 2, so this c. So, this a at least this truth table as 6 column. So, it is kind of 64 rows. But if you bring in abc then it is a 7 sorry 6 + 3,9 kind of column.

So, you will have 5, 12 rows for this truth table. But you see the power of this statement. All that 5, 12 rows are compressed into 3 statement, and many times this is how we think you know. We do not think like though when we have a spec of a combination circuit you write the truth table. But we always think in terms abstract like this you know you have some inputs. Then the problem statement itself could be like this you know.

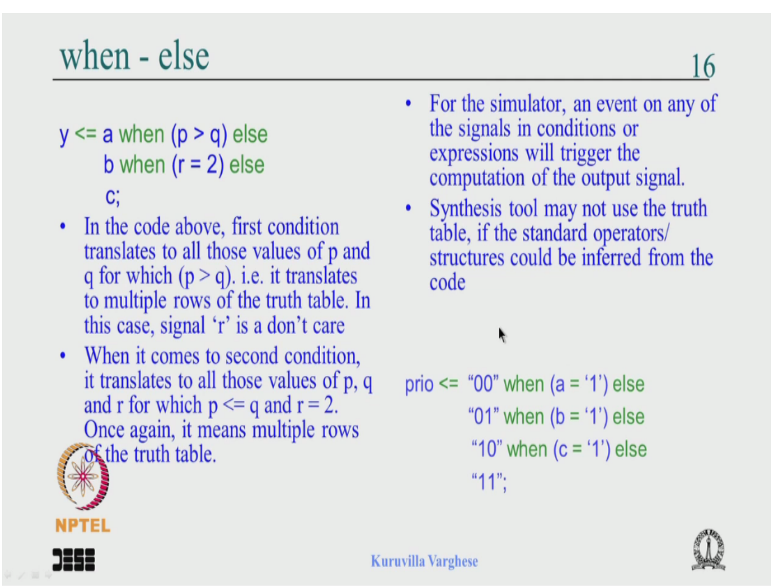
You have p is greater than q, then the output is this otherwise if some other input is equal to something. Then the output is this. If none of this is then the output is something else and so on. So, that also shows that the language allows you to think real life. But you should not loose sight, you should not think this is some magic. Ultimately this we has specifying the truth table and a word when a simulator as well as simulator concern.

If there is any event on p, q, r, a, b or c. This y will be computed or this you can imagine like a process with this abcpqr, and the sensitivity less. Any event happen on any of this input, y will be computed. And y is if there is another statement which has concurrent than that also will be computed. But as for as synthesis tool is concern say it is going to look at say p greater than q.

And there is an operator greater so, it goes to library pick up. The greater operator and that would have been written as a already as a synthesisable code which shows the structure of greater than q like some input greater than some other input. So, that will be replace**.** The synthesis tool will be plug-in that circuit here. And if you think this could be a kind of you know when a kind of multiplexer**.**

When this condition happens this output is left, this input is let to the output, even not and this condition happen this. This input is let to the output. So, this will be some kind of various operators implementation of operator some kind of priority and some kind of mug sing happens. Ultimately, as for as synthesis tool is concern. So, we will see how the synthesis tool does this as we go along, I am giving you a kind of taste of how things happen now only at the beginning.

(Refer Slide Time: 22:09)

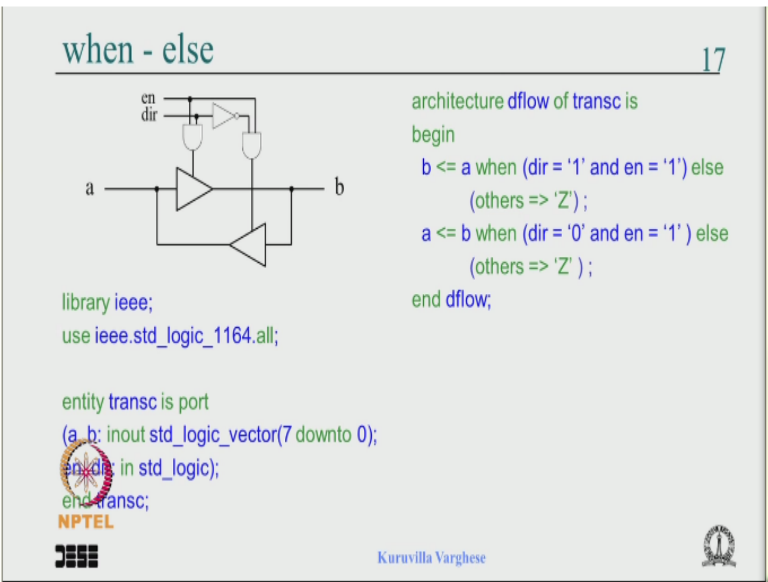


So, this just I have written description in this code. Wherever there is p greater than q. Where there the y is a, when it comes here whenever there is p is less than or equal to q and r is 2, that is b. And for all other condition c and any event happens on any of this inputs, the simulator computes, synthesis tool loops at the operator and in for the operator and replace it with the template structure from the library.

That is what is happening. So, let us take an example which is a priority encoder. So, you can imagine a I am not drawn the picture. But then you can imagine there is a block, where there are 3 inputs a, b, c single bit input, which is encoded into 2 bit because we have 3 bits, and so where the maximum priority is given to a, next priority is b, next priority is company

And none of that happens the output shows 1,1. So the coding is that the prio which is output a 0, 0, when a is equal to 1. Else that means a is not 1, a o. Then an b is 1, then the output is 01. And when it comes here a0, b0 and c is 1, then the output is 1, 0 and when it comes here none of this is 1 or a0, b0, c0. Then the priority output is 11. So, this shows a natural very simple example of using the when else**.** Maybe we will see the little more kind of little more complex example with when else.

(Refer Slide Time: 24:05)



So, let us look at this example this bring in kind of 1 or 2 elements, we have studied together, the array assignment the in out mode and when else. Everything is put together so that you get a taste of some order real life coding. So, this please look at this structure it is a bidirectional, buffer or a transceiver. So, please look at it, so a bidirectional gives you know the connects 2 lines in either direction.

So, you see here when the direction is 1, and enable is 1, a is driving the b. So, you can imagine this as a some output section driving a bus or something like that. This where everybody is tied together similar structures. So, when direction is 1, enable is 1. This is enable and since direction is inverted here. So, this is disable, this cut-off this output is cut-off though the input is coming here.

So, naturally a goes to b, and when a opposite is a case, when direction is 0, this is cut-off. Because this is get 0, and the enable is 1 then this is enable. So, this part is cut-off, then b, assume that b somebody is driving some output is driving this. And b goes to a , so mind you this is bi directional you can drive it. And, somebody else from outside from also can drive it. So, this has to be the mode of this b is in out, mode of a in out.

Because in principle, when this is cut-off somebody can drive it and so on. So, this is the library description you know of the VHDL. This is the standard logic 1164. And this is entity transceiver

trans is port. We have 2 signal and which are 8 bit okay. I am showing 8 bit. Because of control signals are common for all the tri-state gate. So, ab is in out. Because of this structure standard logic vector 7 down to 0.

Enable direction is in standard logic end transceiver. And we define architecture a some name of this entity is begin and end this architecture names ends this part, and this is where we write the concurrent statement. Now you see we are going to write this part first, and part. Because in the concurrent statement mind you. You need to have a statement for 1 output and another 1 output. So, we are going to write b output first, and then a output second.

So, that is what if you are doing .b is an output, b gets a when direction is 1. And enable is 1, else it is tri stated, so we say at this being a bus, we say others you know the assigned z that is it okay. Which says that b is gets a when both are 1. Else it is tri stated, similarly you be define the output a. So a gets b when direction is 0 and enable is 1, else it is tri stated know this is tri stated, else others is z, end d flow.

So, this shows how to use the when else in the more practical more real life case we have seen how the in out is used. We have seen this array assignment the others goes here. So that shows as an example of using when else, another example of using when else. So, let us look at the so, we have completed the concurrent statement, the concurrent statement essentially 2 types which select and when else.

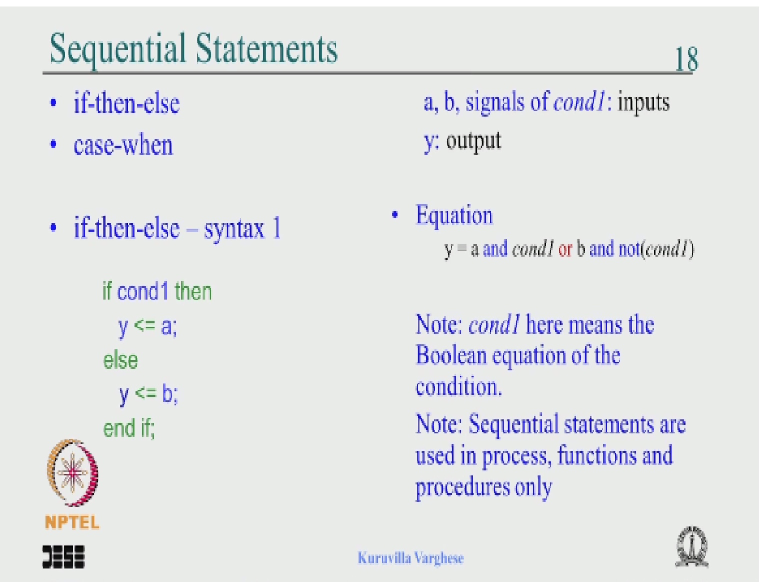
We have some loops we will see that you know together for both the concurrent and the sequential together, and the main concurrent statement are with select and when else. Which select is no priority, output is normally specified as for all the values of some input. The output can be specified as a simple truth table of numerical values or if you have other inputs you can write as a function of those inputs.

In that case it translates into multiple rows of truth table. And we have seen some examples of with select coding. The next little more abstract 1 is when else, which is little more powerful we can specify, we can specify condition in terms of multiple inputs. The expression can be some

other input, so it can capture a much more real life scenario from the specification given for a combinile circuit.

And we have seen how some simple 3 kind of choices can translate into a huge truth table, so that is a power of when else. And we have the coding for a priority encoder, and we have seen the VHDL coding for a simple bidirectional buffer or a transceiver with 2 controls enable and directs**.** So, let us now move on to the sequential statement which are more useful more complex than the concurrent statement like which select and when else.

(Refer Slide Time: 30:13)



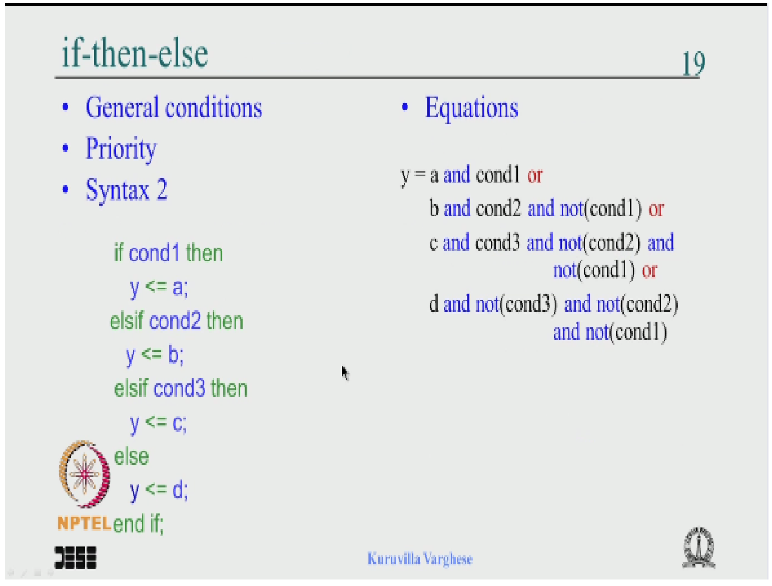
So, let us move on to the sequential statement there are 2 types, this is identical to what you have learned in a c language or sequential languages**,** so you have if then else and case when. Only thing is that it is now kind of related to the hard ware. We have describing the hard ware. So, all the equations are different you know it is not you should know what is hardware behind it.

When you use this statement you should know what hardware it means okay. It is not mear some kind of some variables you have working in a sequential language okay, that should be kept in mind. And I will tell you what is a kind of equation Boolean equations or the hardware you get when you use this syntax, so the simplest syntax is like this. If some condition, condition 1 then some output get some expression of the input okay or numerical value it does not matter.

So, if condition 1, condition in terms of the inputs, then y gets a else, so the priority is there. Else means if note of this condition then y gets b okay. So, end if so a and b are input signal the whatever signals in the conditions are inputs, so the equation is y is a and condition 1 or the moment you say else or b and nor condition 1. So, which is exactly similar to the when else okay.

So, this one is exactly similar to when else. But mind you the sequential statement can be used only in the sequential bodies. Like process functions and procedure, you cannot write e for k is directly in the architecture statement region of the VHDL code. That is not that that is not possible. So you should always use a sequential statement in process, functions and procedure. So, let us look at more complex kind of syntax.

(Refer Slide Time: 32:42)



So, basically the conditions like in when else, it is general conditions you can say p greater than q, c equal to 3 and things like that. And there is priority. So, the next kind of complex syntax is if condition 1 then y gets a. Instead of else you can say else if. There is not els eif it is elsif else if. Condition 2 then y get b, elseif condition 3 then y gets e. At then you say else which comprises of all not of all the conditions.

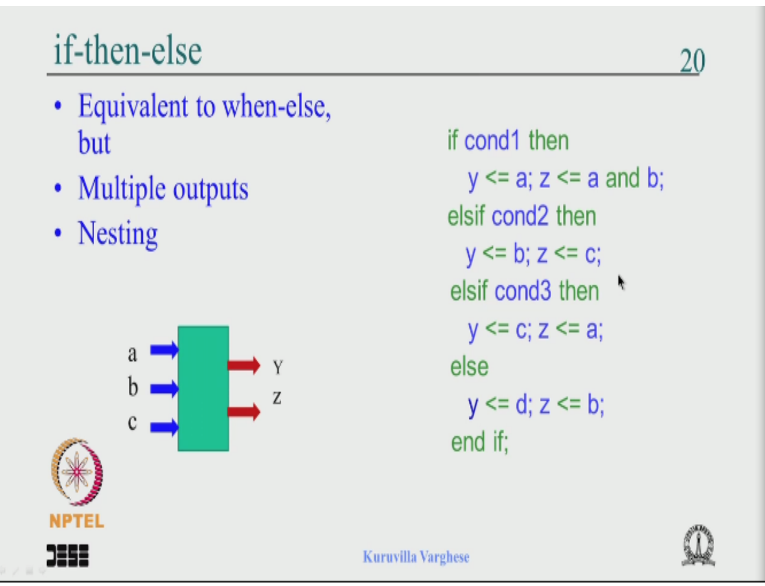
So, you need to have when you specify the proper combatant circuit you should have the last else. I will tell you what happens if not f, so it is very important which is comprises of everything else. Then only the truth table is complete okay. Because, we have putting the condition which

means some rows of the truth table. When you say last else all other rows in the truth table that what it means.

And you can see that the equation goes like this y gets a and condition 1 or b and not of condition 1 and condition 2 or c and condition 3 not of condition 2 not of condition1. So, exactly like when else it builds up you know. As I go down when you come to the last one which is d and not of condition 3, not of condition 2 and not of condition 1. So, like a priority encoder. It builds up and this itself can be very complex.

We are in a kind of abstract condition 1. Condition 1 can be translated to multiple rows of the truth table lot of mean terms of product terms it can come depending on the condition you put. So, that is if then else statement okay.

(Refer Slide Time: 34:41)



Now, I said it is equivalent to when else, but so the question is if it is just equivalent to when else, what is the big deal you know, why you have to write a process and put this inside a process, so can you kind of think of a reason how if then is different from when else. Also you look at the when else the when else statement was earlier. Stay here this is the when else, an output is specify as some condition.

Exactly like if then, so what could be the difference between when else and if then think for a while. So, you can think of the c language. That might gives a give you a clue. So, basically if you look at it what it allows is that when you say condition 1. You could write another input you know you could write z get something, z get something, z get something. z get something. So, in an if then else structure you can specify multiple outputs.

So, that is 1 difference when else and if then and that is very useful very powerful and so that is 1 you can specify multiple output and another thing again if you compare with the kind of sequential language like c. You can write in principle say here if condition 1. Then I can nest it I can say another if, you can say if condition say 5 under this condition.

Then y gets a else y gets something else and you say end if okay. So, you can nest if okay everywhere you know. You can have a if you are if you are if you are and so on okay. And it is not that you can go on you know nesting it. Many synthesis tool limit. The level of nesting because the equation can become messy and you are bound make mistakes and so on. So, if then is else is equivalent to the when else.

But it support multiple output, it support nesting you know. That is very powerful you can bring in lot of complexity in description by this 2 the multiple outputs and nesting. So, this is what I am going to show, so you have y and z output and 3 inputs abc all are let say multiplex, and now you can write say if condition 1, then y gets a, z gets a and b, else if condition 2 y gets b, z gets c, else if condition 3 then y gets c, z gets a, else y gets d, z gets b and so on.

So, this is shows you that you can specify multiple outputs using and equations are similar you know how to work out the equation. There is no you know great you know complexity as for as the equation is concern. You have the same inputs and condition as column a, b, c as column. Then you have a y output, z output you can write the work out equation say.

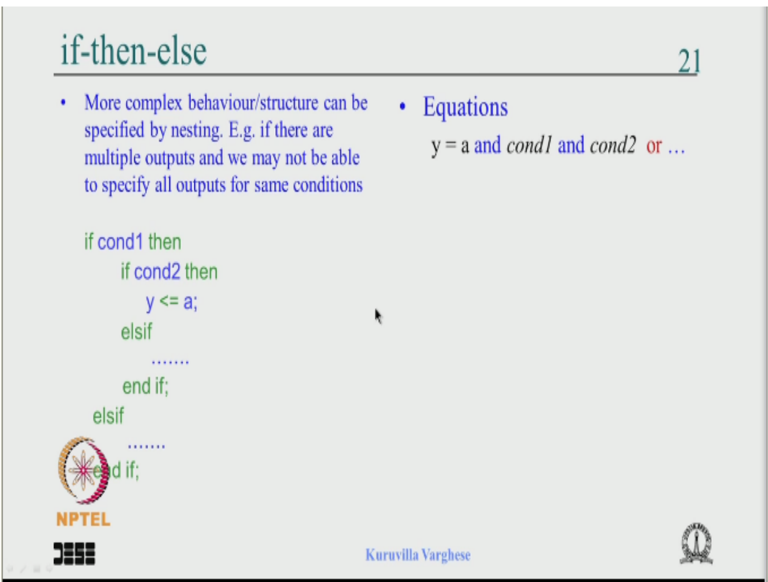
Y is a and condition 1 and b and nor condition 1, and f condition 2. When it is come to z. Z is a and b and condition 1. And or c and not condition1 and condition 2 and so on okay. So, that can

be worked out but if you are clever you should be asking a question say all fine you know it is great. But, this assume 110 that the condition for the outputs are all same okay.

Maybe that you cannot the relation between the input and output are such away that you have no way to specify like this you know. Z is a and b not on this particular condition, respective under this condition, some more conditions are required for a and b. Then you are in you know. You cannot this kind of structure, but that is where the nesting helps you.

You could write say if condition 1 y gets a, then you can write if some other condition is met then gets a and b else something else. So, you could for the nest if to specify very specific conditions you require for multiple outputs. so, that is where the nesting is important.

(Refer Slide Time: 39:42)

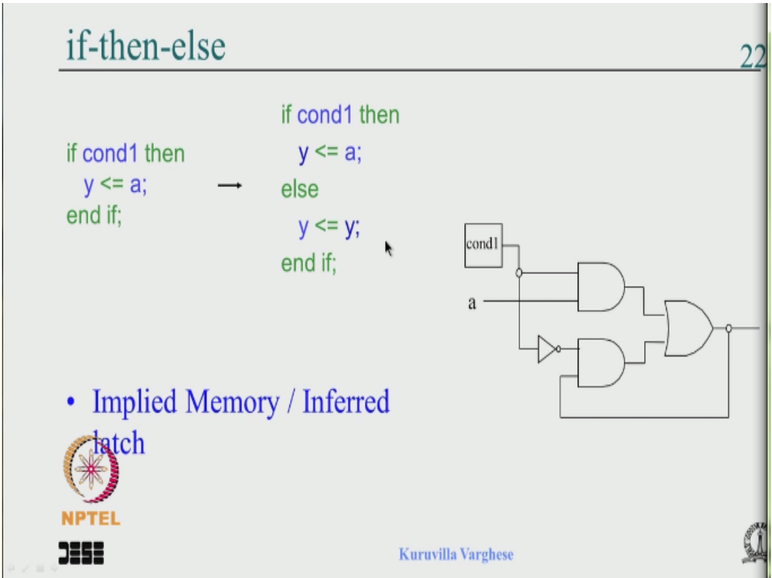


So, you can have a more behaviour or structures can be specified by nesting. Suppose we will not be in the case of multiple output, we will not have the same conditions you know, satisfying all the outputs. So you could have like this you know if condition 1 then maybe z written here something. Then you say if condition 2 then y gets a else if, t gets b end if.

Then else if you know condition 3 or 4 then so on. So, you could nest if and equation as for as y is concern it comes likes a, y is a and condition 1 and condition 2. Because it is under this condition 1 we are putting or then you say you know b and not of condition 2 and condition 1.

And when it comes to this else if condition 3 for say z or something like that. Then this condition 2 does not appear there and so you can work out, so this is where the nesting is useful.

(Refer Slide Time: 40:53)



And let us come to another point okay. What happens if you miss else in a if case like you write if condition 1, then y gets a. And we do not write else. We just say end if and mind you this is a kind of valid VHDL syntax VHDL support this. The simulator support this, the synthesis tool support this. So, what is the meaning of this, so you can attribute different meanings you can say.

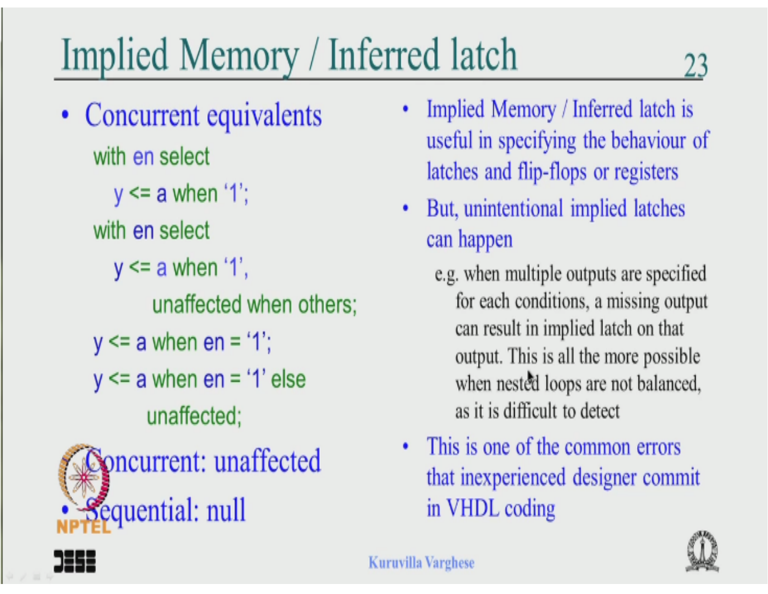
If condition 1is not met y can be 0, y can be 1, but these are less probable from the description, but what is the VHDL attributes or VHDL take this for is shown here okay. It is just by definition do not kind of argue on why this should be like that. But this could be the probable meaning and the VHDL take this way. Like means if you write if you miss else.

It is means that the condition is map then y gets a, else y is y itself okay. And that is funny that is dangerous okay. In the sense that it shows a fed back okay. This condition is met some y gets an input. If this condition is not met. The output is feedback into the input. That is the meaning of it and I am showing. So, there is a memory, it memorises for this condition is normal input goes to output.

If this condition is not met it memorises the previous one. So, it is called implied memory. Because this code implies a memory or inferred latch or you can say this code infer a latch from the return code. So, that is why it is called implied memory or inferred latch. So, the situation is like this if y and a are single bits, and there is a condition maybe greater than q whatever. So, that is a decode of this, p greater q.

When that is 1 a goes to the y and if y is if that condition is not match. Then this path is enable and y is fed back and it is latch okay. And , this is nothing but a 2 to 1 mups. So, this you can replace by a 2 to 1 mups with the select line. And the select line is the condition 1, and the select line is 1, a goes to be otherwise the b fed back. So, this you can replace with a 221 mups and with an invertor you get a latch you know. That is normal latch kind of RTL symbol. So, that is what you get okay. So, this is very valid and if you need a latch like this you can write code like that**.**

(Refer Slide Time: 44:16)



But, the question is that can be have a latch in the concurrent statement okay. Concurrent statement like which select and when else. The answer is yes, because we at least in the case of when else. Because it is equivalent to if then you can imagine you say you say output is something when some condition is met, and you say instead of saying else you keep quiet, you do not state that.

Then you get a implied memory or inferred latch in a concurrent statement. So, or in the with select case you specify a condition a decode, and you do not specify anything else you know. Then you get l, so let us look at the syntax. So, with suppose you take with say this is the enable of the latch with enable select. Y gets a when 1, and we are not saying something for when 0 or when others okay.

You say just say y is a when 1, and we do not say what happens when others, that means y gets y when others okay that is the meaning of it or you can even say like this with enables select y gets a when 1 unaffected by when others that means the output is unaffected when for the other cases both are same Similarly for when else y gets a when enable 1, we do not say else, then you get the same thing or you say y gets a when enable is 1, else unaffected okay.

So, for concurrent there is a you know syntax called unaffected. So, it means that the it we takes the same output, that is the meaning of it. Even for sequential statement there is a thing called null, which will give the same effect. That means that here you can say instead of this if condition 1, then y gets a else y gets null means y will you know remember the previous output.

That is the previous value that is the meaning of it. So, you could specify principle null, so that is the that is how you write inferred latch or implied memory using the concurrent statements okay. And unaffected null can be used and now mind you wherever you do not assign some output do not write unaffected and null. You will get a latch okay.

So, be very careful null does not mean you know initialising into 0 or something like that. So, do not write null wherever you feel that something should be initialised to 0. Null will give you a latch and do not write it, unless you require it okay. So, let us see the use this and you are not normally we use flip flops as memory in the serious design you do not use a combinational kind of latch in real life, so we do not use it okay.

So, what is the use of this kind of implied latch, so the first thing is that the implied latch or implied memory or inferred latch is useful in specifying the behaviour of latches and flip flops or registers. So, we are discussing the combinational circuit now. So, we are discussing how the

combinile circuit can be describe using the concurrent statement and sequential statement. We have not yet going to the sequential statement.

We are still discussing the combinile circuit, but when we go there we will see how this description helps in specifying the memory for memory part of the latches and flip-flops. But, in real life when you write combinile circuit and intentional implied latches can happen okay. So, that does not mean that you will write code like this but when you have complex code.

When you have a lot multiple say 1 scenario is that we take this example and where multiple outputs are specified, so here suppose you have y, z and maybe u is specified everywhere. And you cut and paste, you copy paste you know. That is the usual the scenario now a days a lot of copy paste happens. And suppose by mistake you forgot to mention z here okay, you copy paste it and you forgot to mention z here.

That mean it essentially means z is z as for as condition 3 is concern. So, when it comes to this choice that means condition 3 not of condition 2 and not of condition 1. Z will be fed back to itself and you get a latch. And all the more not only in multiple output. When you have multiple nesting very complex nesting which is unbalanced like you have an if an under that in various choices of if.

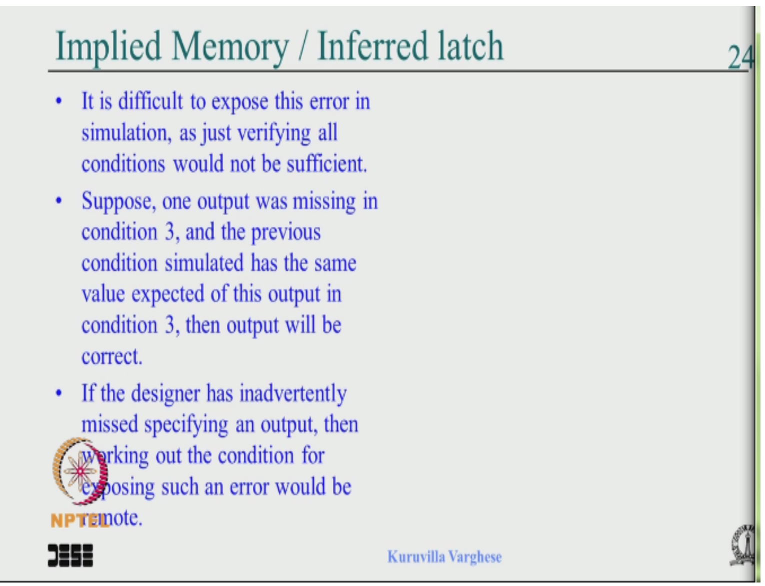
Some as another if, some does not have if and there are multiple output it can be very complex and you can really miss some output to specify because it is really difficult to work out all the outputs properly. In such cases you can miss some output and you will get an implied latch and it is extremely dangerous, I tell you and this is one of the as for as I am concern what I have seen is this is very common error.

An inexperienced designer commit in VHDL coding. That is this implied latch when not in simple case, because in simple case it is very evident when your multiple outputs and when you have nesting very kind of complex nesting in the sense that is unbalance. Then you will get complex nesting in the sense that is unbalance. Then you will get you bound make mistake, some output is not going to be specified properly for some condition.

And you will get a latch and mind you it is very difficult to debug that you know. And because when you make a mistake and you look at the code very less likely that you will kind of unearth that bug from the code. Because you are very sure, everybody is confident now a days and you lock at the code 10 times, 100 times you will not discover that error.

And if you simulate mind you, you will ever 99.99% you will not be able to unearth such an error by debugging in simulation. I will tell you in a moment what is a reason why it will not happen, and if you are giving it to in a real life if you are in a design team may times a verification is done by some other. And he would have worked out lot of test benches test factors for verifying the functionality. Even there it cannot be honour, so in a moment we will see why this is difficult to kind of debug.

(Refer Slide Time: 51:59)



The first thing is that it is not enough if you verify all the condition like you have some input and you like suppose you have 4 inputs, 4 single bit inputs. So, you have 16 condition or you have say you have 10, 24 input test factors you run it through or even you have a 1 million test factors you run it very systematically 1 million test factors. Sill this arrow on be brought up.

Why it is show it is because, suppose in our coding you mis 1 output in condition 3 okay. Now you have as I said you have simulated the million condition all the possible condition. But before

the condition 3 you have suppose simulated the condition 2 for which this missing output had the same value say you are expecting in condition 3 some value. Suppose the test factor you have simulated for condition 3.

The one before you simulated is condition 2 and suppose a condition 2 and condition 3 has a same output as for as this particular one is concern. Then the output will be correct absolutely correct. So, to honour this error you have to work out a condition where the output is different than this particular condition. And if as a designer you inadvertently made a mistake.

You are very because that is by mistake you miss that and if you are not realise this and you will not be able to that condition to honour this error, so the first thing is not to make some mistake you know make this mistake. So, it is in real life also it is bitter some mistakes are netter you do not make such a mistake. Like you drive on the wrong side of the road**.**

Then you are bound to cross into somebody else in to a road, so like it is bitter you do not make such a mistake you be very careful**.** That you do not miss output and be very careful when you nest the if. So, that the implied latch, implied memory or inferred latch does not occur. So, that is where the next statement we are going to look at is the case when**.**

And what we have left is the loops, but maybe that we have coming to the end of the lecture we will look at it in the next lecture. So, quickly we can run through the if then, if then is identical to you know the case when else in the concurrent statement. So, simple condition is if condition 1 then output gets something. Else output gets something else, the equation is similar to when else.

The complex condition is you know you keep on giving various condition. At the end you say else for not of all the conditions, equations are similar to when else the comes from multiple outputs and nesting. And you could specify multiple outputs when the conditions are not identical, you can start nesting. You can think of nesting in various phase. And when you say if you write if under an if it translate to the condition 1 and condition 2.

And when you come here condition 1 and not of condition 2 and so on. So, you can kind of work it out, and when you do not specify the else you get an implied latch and which is which is a latch which is useful and you can use concurrent statement to get the same effect. And you can it is useful in specifying memory in the case of flip-flops and latches. But in combinational case when you have multiple output the nested if then un intentional latches can happen and as I explained.

It is very difficult to one or thing simulation, because you made it by mistake to work out the condition is quite tough. And there is no point in you know there are people will make a mistake which can be corrected in 5 minutes. And next one week you will simulate, simulate, simulate to unearth that error. It is not worth-while you know. You plan properly you code properly, you go through the code.

You take any number of it you know, spend as much as time on paper thinking about it and design then less verification will be there you know. An experience designer or experience engineer should plan properly, go systematically. So, that with minimum iterations things work properly that is how you should you know work out things than you know rush to the things arbitrarily quickly writing cooking up something.

And forever debugging, forever sorting out the problem what 1 mistake you made in 2 minutes can kill your 2 weeks of time many people's time. So do not do that, plan properly. So, that I stop here today with this when else if then, if then else which are kind of similar but if then is complex more useful. So next class we will take the case when and the loops. So, please revise it write some examples of your own using this statements. So, thank you, I wish you all the best.