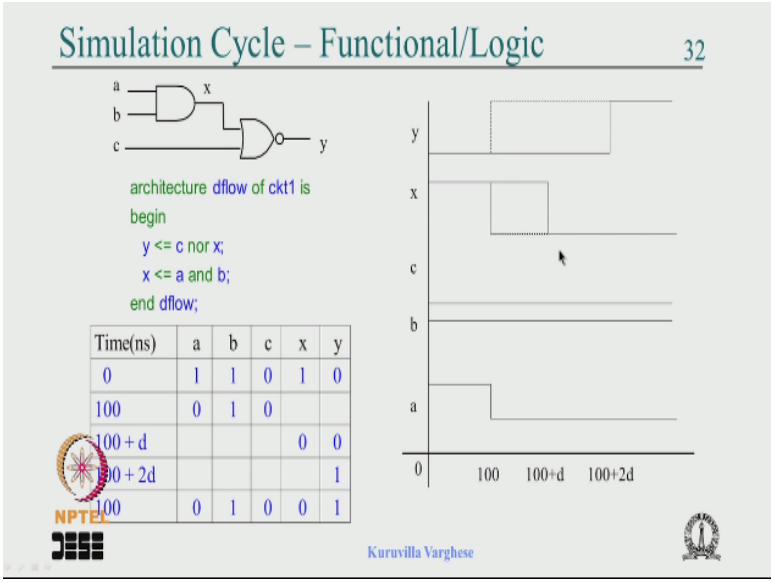
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science - Bangalore

Lecture-13
Classes and Data types

So, welcome to this lecture on VHDL in the course digital system design with PLDs and FPGAs. The last class we have looked at the simulation and the various classes and data types. So, we will have a quick run and we will complete the data types and classes and will go on to the basically the concurrent statement, which will enable you to write start writing the codes for combinable circuit is start with. So, let us look at this slides last lectures portions.

(Refer Slide Time: 01:02)



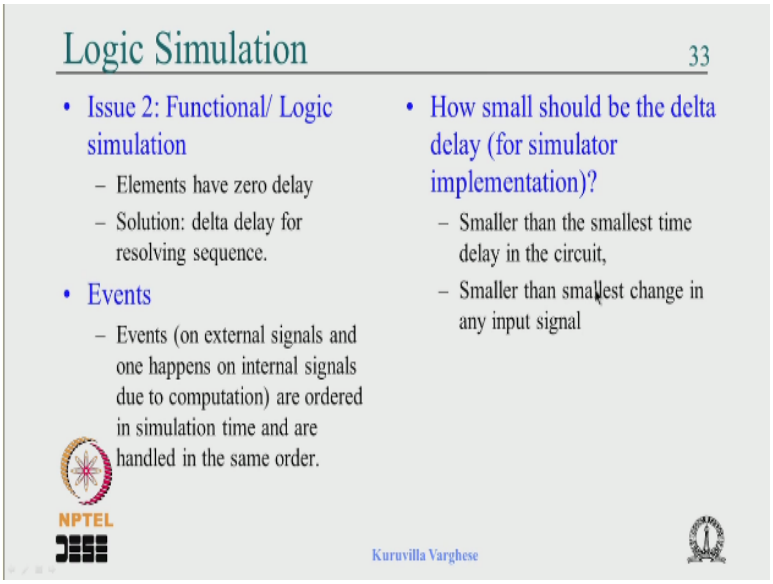
We have talked about how kind of event driven computation enables the simulator to resolve concurrency. We have looked at the case where there is time delay and we told that that time delay enables a simulator to keep track of the sequence. But in a logic simulation or functional simulation where there are NOR delays, this is problematic. Because if there is an event on a then x also the result will happen at the same time.

That will create an event and here the y everything changes at the same time. And you cannot keep track of the sequence or the one after the other it cannot be ordered and the problem came because there was nor delay. So, the solution is add a small delay called delta delay. That is what

we talked and do the event driven computation. So, if there is an event on a at 100 nanosecond look at the right hand side then evaluate x.

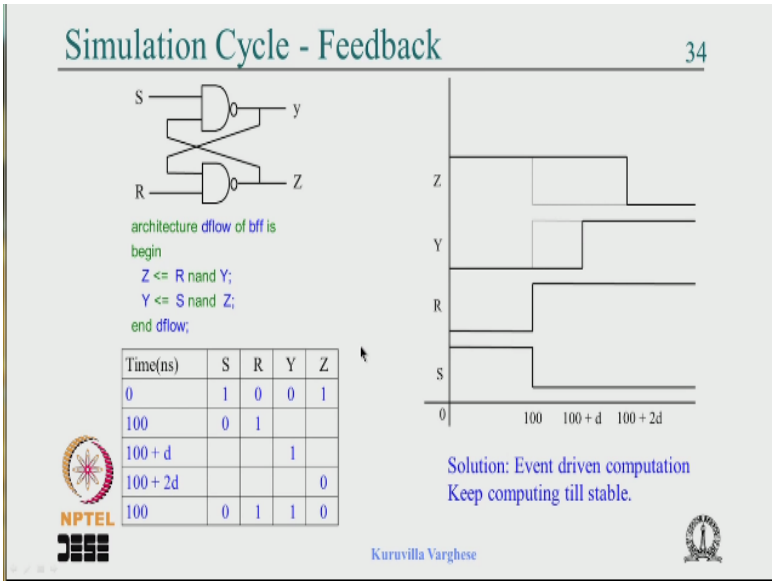
And which is assigned after delta delay, so that creates a event on x at 100 + delta. So, the simulation time move from 100 to 100 + delta and this is evaluated and y is assigned at 100+2 delta. And then there is nothing more to compute because there is nothing y does not compound a right hand side. So, the everything gets stabilise, and that is shown pictorially initially we show the delta delay. But everything is a delta is made 0 and everything is stabilise. Then you get the correct output. A is going from 1to 0 and you can see y is going from 0 to.

(Refer Slide Time: 02:55)



And we also said like when you implement you can say that the delta delay how small it should be, it should be smaller than any circuit delay or any changes in the input, you know that these 2 are related in way. Like you specify like delay is you know change in the input kind of in relation to the delay is you implement anyway.

(Refer Slide Time: 03:22)



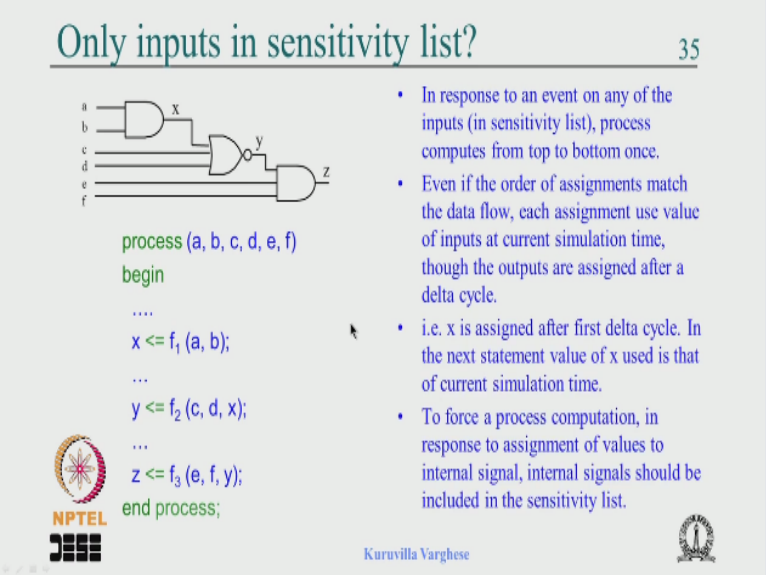
So, for to kind of satisfy our curiosity we have looked at a case of feedback, it taken as cross couple NAN latch and it was in reset mode and then we change it to set mode. And it was s and r both, change that 100 nanosecond and which kind of force evaluation of both z and y. In the first place then nothing z does not change there is nothing to be done, y changes from kind of 0 to 1.

Which is assign a 100+delta the computation at 100 nanosecond is over. So, it is move to 100 + delta and that force is a computation on the Z. And Z changes from 1 to 0 at 100+2 delta. That in turn force this to be evaluated at 100+2 delta. But there is no change y remains the same. Everything is stabilise, delta is made 0 you get 100. And that is how which is shown here.

It is going from kind of reset to set, and you can see that this is the delta delay which is shown ultimately we can see y is going from 0 to 1. So, simulation in the case of feedback a function of simulation with feedback. Two things can happen either it go on oscillating or it can kind create a stable output after few delta delays, depending on normally you can imagine it is related to number of levels of the logic you know.

As if there are multiple 3 levels of logic and we have 3 internal signal then it will take 3 sequence, 3 steps for to stabilise. Even in the case of feedback, if it is going to stabilise in a cycle it depends on how many kind of levels of logic is there with the internal signal, then it is going to be stable. Otherwise it will oscillate which is not an issue at all, as for as simulation is concerned.

(Refer Slide Time: 05:28)

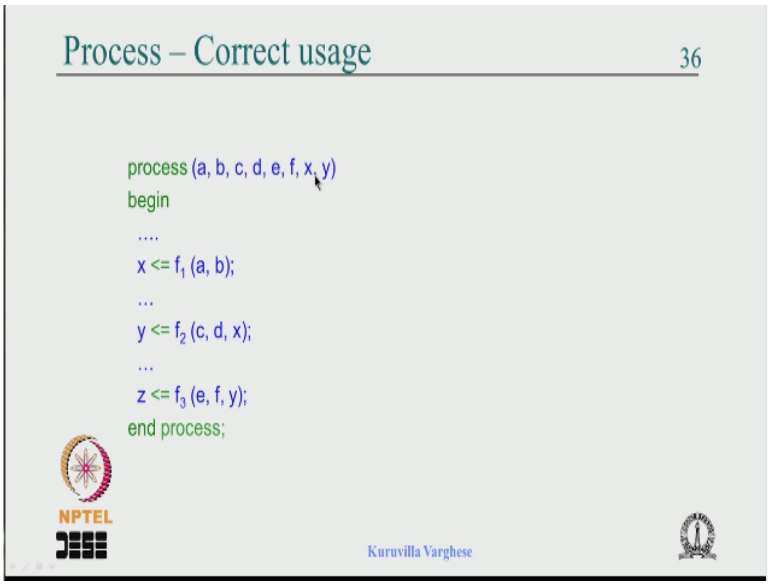


And, then we looked at this scenario where we had some 5 inputs maybe 6 inputs and we have some internal signal, and we as we described earlier, we implemented a process with this input signal in the sensitivity less. But now having non this delta cycle business we know that. Even though x y z is written in the proper order assignment with some mean for case whatever.

Then but the x gets assigned only after a delta cycle and when it comes to y it gives as the current value of x, and the change in x does not get reflected in y and change in I mean so, y and z remains kind of old. X is going to be assigned correct value after the delta cycle. So, to be able to you know made this process compute correctly we need to put this x and y these internal signal in the sensitivity less.

So, x get assigned after delta cycle which creates a simulation time or I mean event on x. So, that comes back and runs through it y gets assigned again and even on y will make it to compute again and ultimately after 3 equation it gets computed. So, the rule is that basically you have to put the internal signals in the sensitivity less.

(Refer Slide Time: 07:01)

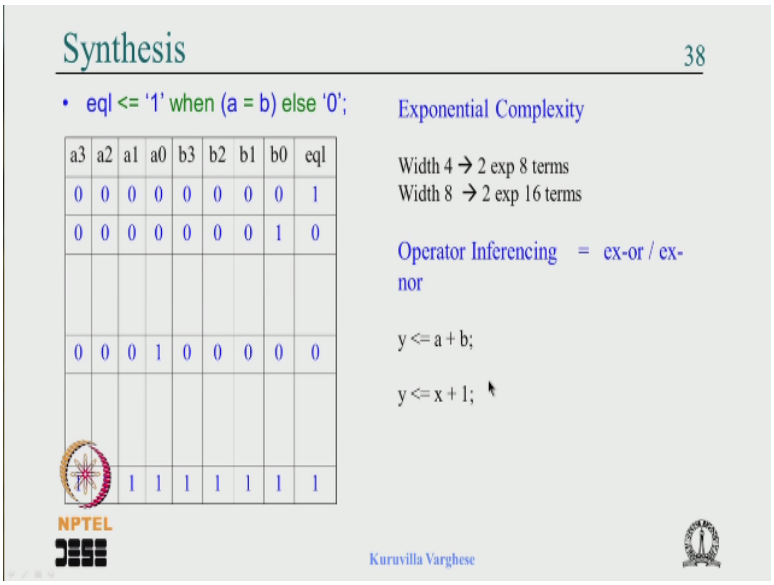


If there is 1 and as I said you could have written an a single expression for z then this is not required. If you have internal signal then that need to be put in the sensitivity less, we also said that the concurrent statements are equivalent to the process. That means you can think of concurrent statement as processor process with whatever is on the right hand side of concurrent statement in the process.

That means if you have multiple concurrent statement it is possible to write you know we related to saying kind of inputs. Then you can write a single process to kind of concisely represent all that concurrent statement. Similarly, if you have a process with multiple output it is possible to kind of right equivalent multiple concurrent statement. So, that should be understood and in any case if you have multiple process, multiple concurrent statement everything is kind of concurrent to each other.

If an event happens then wherever that signal appears on the right hand side of concurrent statement. Wherever the signal appear in the sensitivity list of a process, all will be computed by the simulator okay. Again when we talk about all the sensitivity less event definitely we have talking about simulation not synthesis. As I said synthesis looks at the written code to infer the structure. So, let us move on to the last part we have handled and okay.

(Refer Slide Time: 08:40)

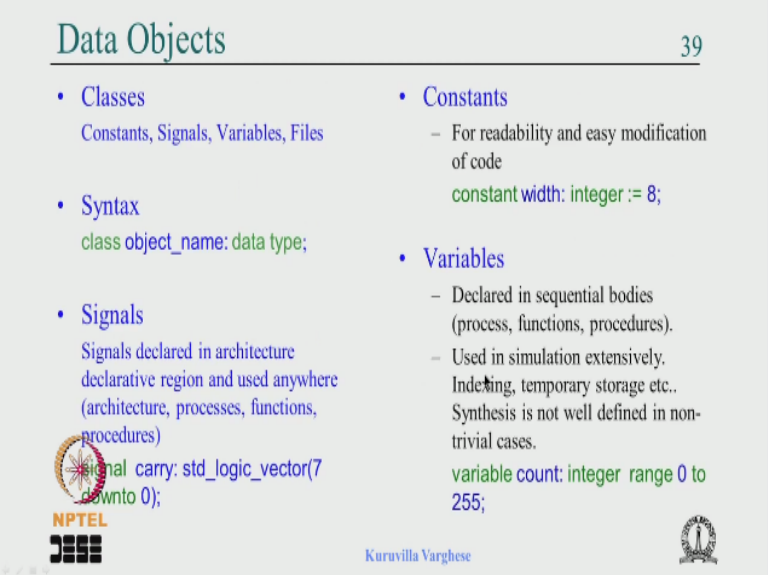


We have looked at the synthesis at we said, when you have kind of such a statement the sure way of synthesising is to write to truth table. And in this case we write the a, b as a as column and you write equal output in terms of those inputs. But problem is that as the data width input width increases. The complexity is exponential, because it is everything expands in power of 2.

So, this become prohibitively kind of costly in terms of memory and computation. So, most of the time the synthesis tool will infer, equal means ex-or + means an adder, +1 means an incremented and so on okay. And normally these are implemented in the library with the proper code to implement the required circuit. That means the + will be the moment the + is taken as a kind of function or an operator.

You go to the library then the adder is implemented there. Similarly when you get equal ex-nor with the and or ex-or with the with the subsequent circuit with the is implemented in the library code, that is how the synthesis is done not on truth table and because that can be prohibitively costly.

(Refer Slide Time: 10:10)

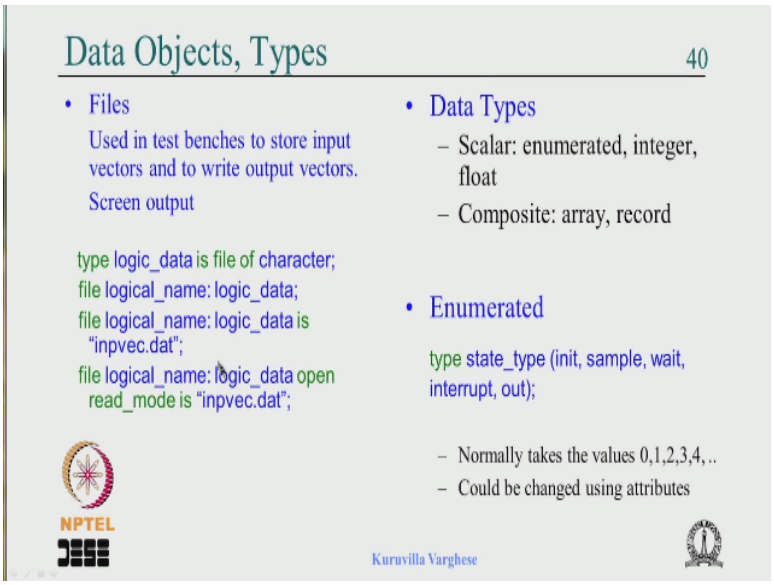


And we have looked at the data objects, classes, we have 4 classes constant signals, variables and files. The syntax is class a name, a data type, signals are used everywhere, we signal, name and the data type. Constant is used for kind of some kind of generic implementation you have a say counter then you can specify the width of the counter or the size of the counter in a constant, so that you do not have to go and change.

Suppose you want a 16 bit counter in 1 place you change it and you write everything in terms of width you know. Wherever you say the counter output, you say width -1, down to 0. Then that is done. Variables are used for indexing temporary storage in simulation it can be used for synthesis but in a non trivial case like a normal algorithm you take with variable.

Then sometime it does not make sense, we will see what is a use of variables and this is the kind of syntax variable is a key word a name in this case integer is specified, and so you have to specify the range. So, that the it can be inferred that is an 8 bit signal.

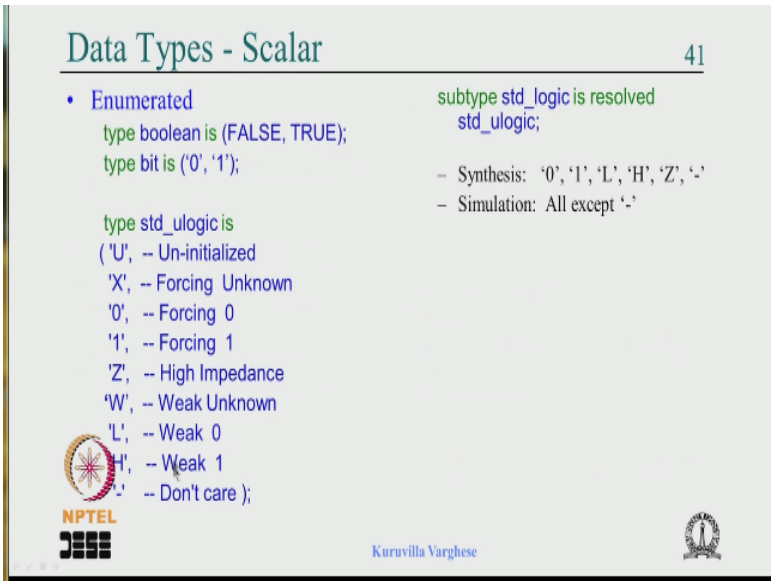
(Refer Slide Time: 11:33)



And files are used for test benches we will see when we go to the test benches then that is the 4 classes and everything is expressedily kind of declare. And the data types you have 2 types 1 is scalar the second one is a composite. In the scalar you have enumerated integer and flow data types. In the composite we have array and record. We have seen a numerator, so you say type which is a key word given name for the numerated data type.

And in the bracket you write what are these values this 3 type can take, so in this case this is a straight diagram with 5 states the names are the states are in it sample, wait, interrupt, out . Whatever it means but internally this will be coded in kind of 3 bits. Because it is 5 states and it takes values like 0, 1, 2, 3, 4, and this assignment for whatever reason if you want to change it. You could change using attributes we will see when we come to the functions and operators there this is useful.

(Refer Slide Time: 12:47)



And we also have seen this particular data type here the Boolean is written which is takes a false and true value and the type width is defined with 2 values one is 0, 1 with course around and this quiet respective. So we use standard logic but the first thing is defined is standard u logic and that can values like u, x, 0, 1, z, w, l, h, and the dash and the comments are written here.

And most of it is probably this is a bit of an over hill, but then most of it is useful like a U when simulate a start simulating at the beginning before the signal get initialise, it can be shown as uninitialized 0 and 1 are the whatever you know 0 and 1. When 0 and 1 clashes you get forcing unknown clashes in the sense that you have tri state gate both are enable to the same line and one is driving and the other is driving 0.

Then the result in value is unknown, so that can be represented by simulator as unknown . This for the same line but l and h generally means weak 0 at weak 1 essentially it means that the drive is not a lower resistant drive it is a higher assistance drive, so when you have a line which is pulled up to the VDD, then you have a X pulled down to the ground of VSS.

Then you have the l, and if l and h clashes as in the earlier case. Then you get weak unknown. Now mind you that just because you write an output is assigned with l. You will not get a pull up resistor or pull down resistor or a weak drive or something like that . This is just a standard u

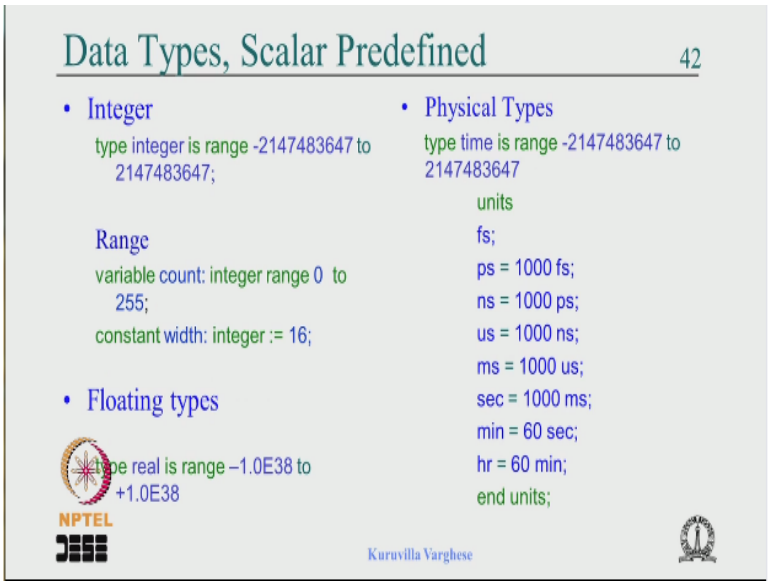
logic values. The synthesis tool is not going to obey all that. It is going to treat this as 0 and 1. Z is the high impedance very useful.

And for synthesis we need like many a times like we need to say is do not care and then the do not care is use. For simulation all these are meaningful but may not be the do not care. But synthesis normally we use 0, 1, Z and do not care**.** So, that is a kind of standard u logic definition and standard logic is defined as subtype standard logic is resolve standard u logic, we will see that what is Z1.

But essentially it is that when multiples standard U logic output at driving a line . In synthesis again it does not matter because you put tri state gate the no issue. But when you simulate it more than 2 outputs are driving it by mistake. Then the resultant value is not has to be computed by the simulators. Simulators know knowledge of that because simulator just know that the standard logic takes this values. So, this function this particular function resolved kind of resolve that multiple drive.

You know that if 0 and 1 come together this function will tell what is the resultant value okay. That always need not be x. Because 01 is x but then if both are 0 then it can be 0. And similarly if if there is an x drive and 0 it can be still 0. Because the weak one and a forcing 0. Then it the resultant value can be 0 nothing not short circuit happens like even a 0 and 1. Then you have a short circuit but then whenever there is l or h. Then there is a there is assume to be a kind of high resistance drive. So, it not a short circuit.

(Refer Slide Time: 16:54)



So, let us move to the integer and floating and physical types. The internally this is the internal definition of the integer, type integer is range like -2 raise to 31 to + raise to 31 okay. You have you know the kind of truth compliment range 1 is the positive site is 1 less. But then you can imagine that and the way the wait use it is variable.

Suppose for a variable we are using integer, then variable name count, integer range 0 to 255**.** Always we have to specify the range because this integer range is defined as the it is a big range. So, you have to find the equivalent width of the bus. So, always we limit the range or you say constant width, integer colon equal to 16. Whenever you say colon equal this is not like the assignment we used for the signal**.** This is an immediate assignment.

So, when we assign a signal we use the symbol less than or equal to for assignment. That has a significance like delta cycle and all that. But this does not have any kind of delta cycle assignment or timing assignment. This is assigned immediately. Whenever is colon equal to is use**.** Let us come to floating type, the floating type is internally defined in the standard library as type real is range – of 1 into temper is 38 to + same range.

So, that is a kind of floating point number. Once again just because we have a data type, do not imagine you take 2 floating point numbers and multiply the synthesis tool is going to implement

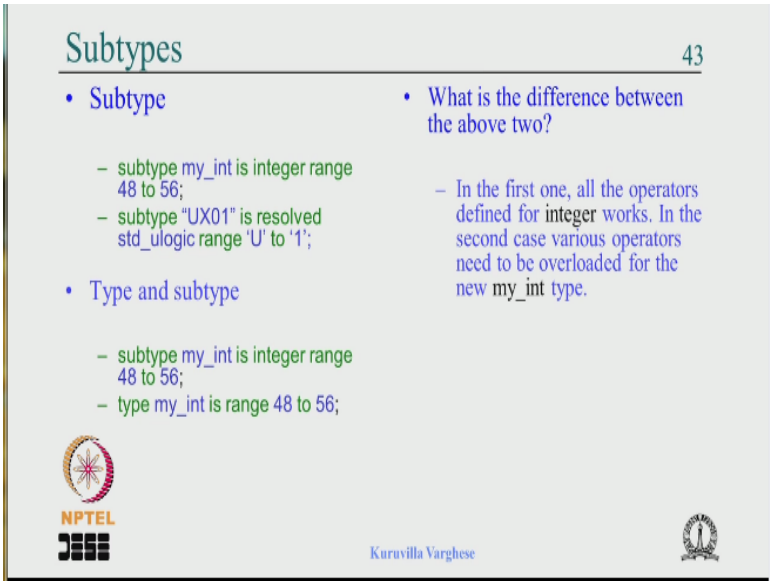
a floating point multiplier. It may not happen at all. So, there is unlike the sequential languages we need a time in VHDL or hardware description language.

So, that is specified as time, so that is called a physical data type. So, you have type, type is a keyword, time is range. Again this same range is this. And the units 1 important thing is that this is integer and floating point. These are near numbers. But this is physical type. So, you need a unit, so the units is defined like that, you start with units keyword the end units is a end of that kind of body.

Then you specify the units like the basic unit is fs which stands for femto second, then ps which is pico second, equal to 1000 femto second, nanosecond is equal to 1000 pico second, microsecond so honour to hour. So, in a simulator and the VHDL code you are able to specify, you are able to say after 10 nanosecond, after 100 femtosecond, after 1 minute and so on. Whether it makes sense or not, but you can specify the signals with the time delay you require.

And that is enable tool this physical data type, so these are the kind of predefine the data type internally, you have a numerator, you have integer, you have floating type which is called real and the physical type like time . One more thing is that you can define, suppose you have any of this type you can define a subtype of the type.

(Refer Slide Time: 20:47)



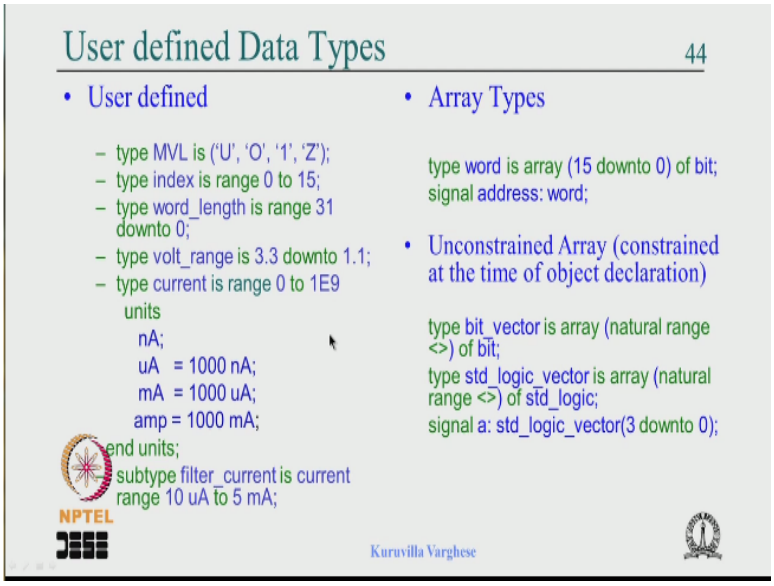
So, here you see the definition is a subtype, which is a keyword mind is integer range 48 to 56. So, wherever use my int it is the integer with range 48, 56 . Similarly you have sub type UX01, it just a name, any name is resolved standard u logic, resolve means it cause a resolution function range u to 1. That means it takes in the standard logic all the values from u, x, 0 and 1 you know, that is the meaning of u to 1**.**

So, like but then you might ask you could have written like this you know we have written subtype a name is integer range 48 to 56. But, you could also define a type of your own like saying type mind is range 48, 56. So, kind of what is the difference between this 2 if you can think about it some of the might know **.** When you use subtype you are a kind of you are a part of that my inspect.

So, essentially whatever operators define for my int can be use here. So, you can use the +, - division multiplication. Whatever was define for integer can be use by this particular, when you use this. But if you define your own range. Then you have to overload those operators for your type. So, it is not a kind of simple difference. So, if you can live with the subtype, always use a subtype.

Then defining your own data type because if you define your own data type, then you have to overload all the operators, whatever you need for that particular data type like suppose you have defining your own integer you have to go to the integer operators, and overload for this new data type. So, you will end up writing lot of functions to overload for +, - multiplication, exponential and so on. So, always try to use subtype if at all is required.

(Refer Slide Time: 23:28)



So, let us move on, let us see some we have seen the definition of what is standard. But you could define your own data types say take this case just is an example, but when you do design try to use the inbuilt operate, inbuilt data types. Because operators are specified you will have less head-ache, then defining kind of your own data type.

So, these are some of the user define examples of user define data type. So, a type MVL is U, 0, 1, Z. It is a user define enumerated data type. Similarly type index is range 0 to 15, this is a user defined integer. The next one word length is range 31 down to 0, it is a user define, integer data type. Type volt range is 3.3 down to 1.1. So, it takes all the real values, so it is a user define real type.

Similarly this type current range is 0 to 1 in tend ration 9 which is again user define real type. And this is a user define physical type. Which is type current is range 0 to 1 E9. Then you define units nano ampere, micro ampere, milli ampere and ampere and there is which shows a subtype kind of **h** declaration. So, that is the user kind of data types.

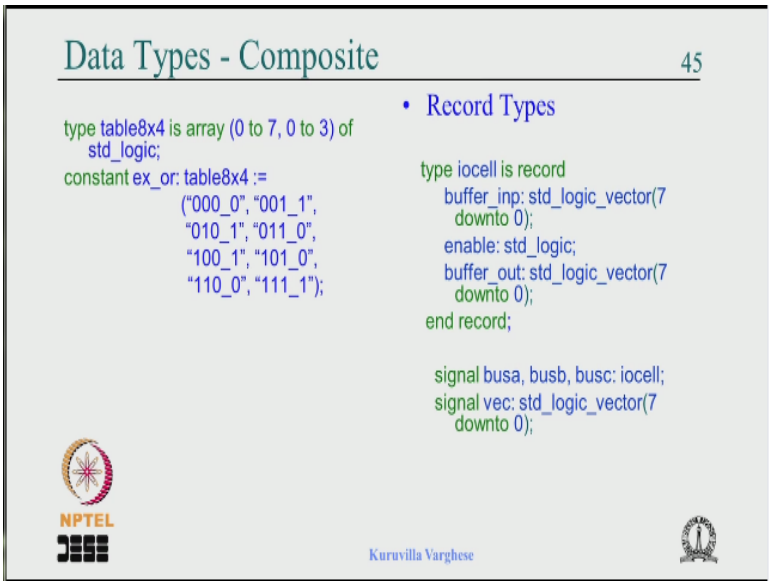
The scalar 1, a numerator integer, real then the physical types, this can be pre-defined, this can be user define. So, let us look at the composite data type. Look at this definition of an array. So, here it is define as type, word. Type is a keyword for a data type, word is a name is array again

keyword 15 down to 0 of bit, which we are saying that whenever we use word it is a 16 bit **h** bus you know that is a meaning of it.

And then you can declare signal, address is of type word, that means address is a 16 bit bus or a 16 bit signal and like in libraries many a times arrays are declared unconstraint. That means here you say take this example, which is in the standard logic 1164 package, which say types, standard logic vector is array, natural, range. This means unlimited, that means whatever is the natural number defined in the standard library**.**

Mostly that is 0 to 2 raise to 32 -1 of course off standard logic. That means that this standard logic vector means such a huge un constraint array. But that is why when we declare we use standard logic vector, we constraint to proper size. Like here signal a is standard logic vector 3 down to 0. So, it is 4 bit because at the time of definition it is unconstraint. So, we constraint it while using it**.** That is a kind of standard logic vector**.**

(Refer Slide Time: 27:14)



So, this is the kind of single dimensional array. But, nothing stop you from you know using it for 2 dimensional or multi dimensional array. So, an example is shown here, type a name is array 0 to 7, 0 to 3 of standard logic, so you can imagine this is an eighth element array. Each element is of 4 bits, you know that is a meaning of it like 4 bits of or 4 standard logic value. So, it is an 8 by 4 2 dimensional array.

And you see how this data type is used constant, ex-or that is the type this particular 2 dimensional type equal to. Then you have 8 values which is in kind of double codes. Each value because it multiple bits, and each value is a 4 bits. And an underscore is port 2, just to separate, you can use underscore whenever there is a kind of strength to separate and if you carefully look at it .

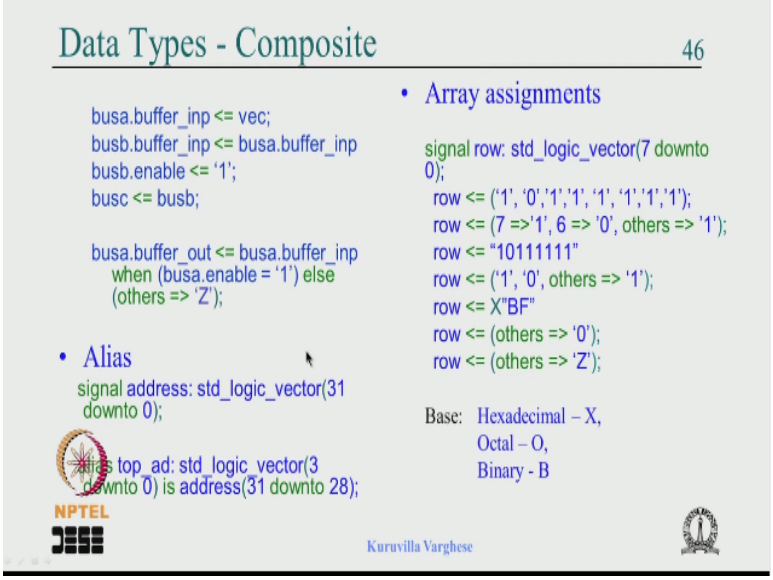
This is nothing but the truth table of 3 input xor gates you have. These are the inputs, and this is output, so this is the clever way of writing the truth table. So, that is array, so nothing stop you from using it for multidimensional array. If it is useful there are you know kind of 2 dimensional structure sometime used in hardware. So, in such places it will be advantages used 2 dimensional or multi dimensional array.

And let us come to this record type which is similar to the structures of c, so here how it is defined is an example is shown by iocell is record and you say end record and within it, you can include multiple signals. So, here you have an input signal called you know buffer input is standard logic vector 7 down to 0, enable is standard logic, buffer out is again a 8 bit vector, just comprises of two 8bit signal and one single bit signal.

You just meer signals there is no logic. But you can imagine this as a say you have a tri state gates 8 of them. Gang together to connect when 8 bit bus maybe. Then this can be treated as a input of the tri shade buffer. This is the output of the tri shade buffer. And enable is the enable signal of the tri shade buffer. So, this is something you know it hardware which comes together.

So, maybe it is worthwhile to put at together in a record . So, not very much use, but then can be useful. So, this shows a usage you have a signal which is busa, busb, busc is of type of iocell. It means that there are 3 kind of similar structures you know input, output and enable and we have a vector which is 8 bit. Now in principle you can assign this vector input 8 bit vector to the input, input of bus a, output of bus b and so on okay. So, let us see how it is used.

(Refer Slide Time: 30:42)



So, here you say busa.buffer input means for this particular bus the buffer input is assign with vector, which is an 8 bit you know vector, which is define here. Similarly you can say bus b buffer input is bus a buffer input. It means that there are tight together. You have 2 inputs tied together, busb enable is 1 means that you are driving 1 to it, bus c, bus b means that again these everything is connected together you know that is same**.**

Suppose you want a logic you really want to synthesise iocell logic, between this input and output, then you say bus a for particular bus, busa.buffer out is assigned busa.buffer input, when bus a.enable is 1, else others z okay. Now we will see soon what is others z, others z means all the values all the 8 bits. If it is in the case of 8bit is tri stated, that is a meaning of it. So, we will the record is not much use.

But then in principle you could combine kind of signals with the correct contacts, you can combine put it together which may be useful to a kind of modular code useful to understand and all that. And let us come to this keyword alias**.** This is nothing but suppose you have a 32 bit address signal, address is standard logic vector31 down to 0.

And many a times you want to use some top bits of address for something maybe memory decoding like a when you have an address bus, the lower address line goes to the addressing the memory and peripheral. It is a higher bits which fix the memory map. So, that is use for chip

select decoding, so may have an suppose we need we are doing the top 4 bit for this kind of chip select decoding.

And we want to we do not want write all the time, address 31 down to 28. So, we give use an alias or an another name called say top add. Then how it is defined is you say alias the name how many bit it is standard logic vector 3 down to 0, it is 4 bit, is now that 4 bit comes from where is written here, is address 31 down to 28.That means that we have 4 bit signal which is 31 down to 28 of address is now known by a different name.

So, wherever you stop add, you will get this properly okay. That is a basic idea. Now let us come to when you have an array how to assign the array, so please have a look at this various types of assignments. So, the signal is defined as a row which is standard logic vector 7 down to 0. So, this a row signal which is an 8 bit vector. So, it can be assigned in these ways you know say row assigned 1, 0 like individual bits.

And if you remember our component instantiation, this is like a positional association like seventh bit is 1, 6 bit is 0 and so on okay. So, that kind of (()) (34:34), you can use then kind of named association, so we can say row gate, seventh position is 1, 6 position is 0, others is 1 means all others. Other than 7 and 6, all others are 1 okay. And you could even write like a complete string**.**

You say you put double codes and write the value like as a single string say 10111111like that you could write you could use others along with the positional association like you say row is 1, 0 and all others are 1. Others like equal to an greater than 1 okay. You could use various basis you say row is X, BF that means row get you know kind of 10111111 which is the hexadecimal equivalent of this whatever.

Assignment done here you could use octal, binary specifies also the pieces also and you could say like this. In a many times at the initialisation you may want make everything 0 or tri state, then you can say row gets others assign 0, that means everything is 0.row others z that means

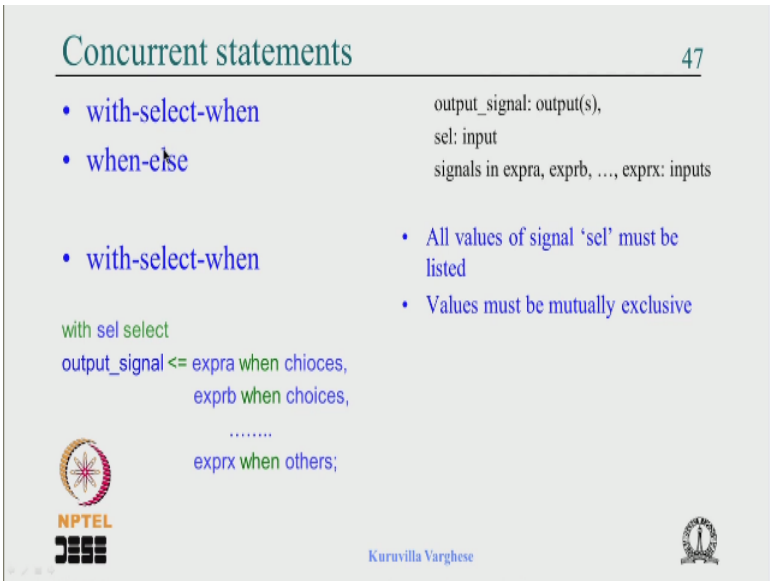
everything. All the 8 bits of row are z, that is what is you know written here. Bus a. buffer out is bus a.buffer input. When bus a enable is 1, else is tri stated is the thing.

So, I think that concludes the data type which is little boring, the class and you know the objects data type. So, the last part we have seen the composite data type, array you can have array is a multidimensional arrays, you can assign them in various ways. You can use record to combine something **h** kind of which comes together like in an iocell and things like that.

So, that is we have seen this array assignment for the row, so let us move on now with the serious the VHDL, so all kind of different models of description, how the simulator works what are the classes, what are the data types. All that is kind of we have covered. Now let us get down to the serious business of writing the codes. Which need basically the constructs or the statements and we will go to the statement.

So, first let us take the concurrent statements when I say concurrent statements these are the statements written in the architecture declaration region as a concurrent kind of statement, that means it is not written in the process not written in the function, not written in the procedure okay. Procedure can have concurrent statement . But any of the sequential bodies you cannot have the concurrent statement. So, normally it appears in the architecture statement region. So, let us come back to the slide.

(Refer Slide Time: 37:51)



You have 2 types of concurrent statement. One is called with-select-when, and the second one is called when-else okay. These are the 2 concurrent statement there are loop like generate loops which we have seen we will take that later, after maybe the sequential statement also . So, let us look at the syntax for with-select-when. The syntax is like this with that is a keyword.

Sel, sel is some input it can be single bit, multiple bit select okay, that means we are going to define an output in terms of the various values of this input signal that is a game, and you say some output signal it can be y, z or any name assigned some expression of inputs a when choices. Choices mean the values of select line okay. If select is 2 bit, you say y gets 1 when 00 okay.

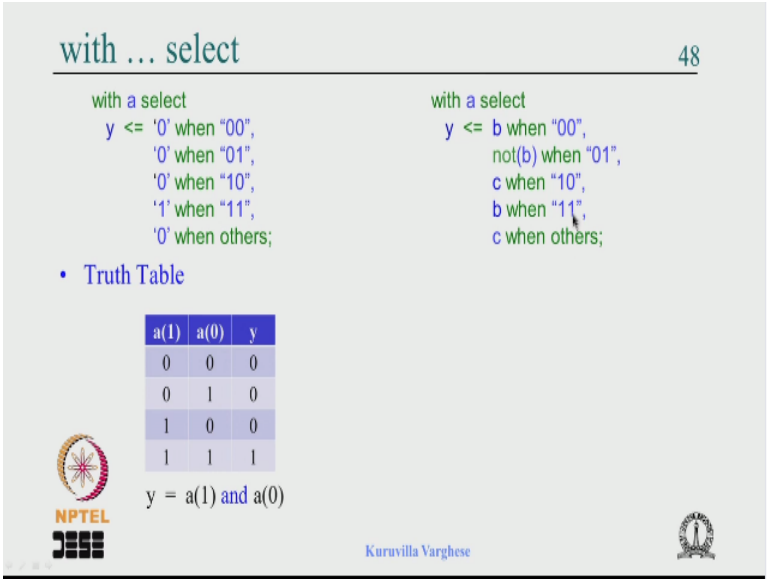
Y gets 0, when 01 and so on okay, so the rule is that so this is the output signal, output signal single bit or multiple bit, this is an input signal. And the choices means all the values of input signal, all mutually exclusive values. You cannot specify 00 here and again here. This has to be mutually exclusive. And one more thing is that if you write an expression then this is composed of inputs.

So, you have inputs here, you have inputs here your outputs here and now you think what is happening when you have some input signal you are telling the output, for all the values of input signals. And this can be as I said this need not be an expression in terms of input. It can be some

1 and 0s are numerical values. So, you must be getting the idea, that is for some input signal, all the values of input signal is specifying the output.

This is nothing but the truth table okay. So, I am bringing it early on because many text books show some typical examples which we will discuss it, which looks as if the which select is kind of meant for that particular hardware and so on. So, that is to avoid this confusion I am talking about the truth table. So, basically we has specifying the truth table for an output signal in terms of the input signal and let us see some examples.

(Refer Slide Time: 40:41)



Say here you take this example which say that there is an input called a, which is let us take it as 2 bit with a select y is output, y gets 0, when a 00. 0 when01, 0 when 10, 1 when 11, 0 when others okay. I will come to this in a moment, which says that y is 1, when a1 is 1 and a0 is 1, else it is 0. So, you quickly get this is nothing but and AND gate. So, it is very simple it is the truth table of an AND gate.

Which is quite complete here there is nothing to add. But you may be wondering about what is this game like the syntax also we said some expression last one is when others. That means when you come to the last one, whatever is not covered here can be covered you know can be written here. So, in principle we could have written one when others instead of when 11 okay.

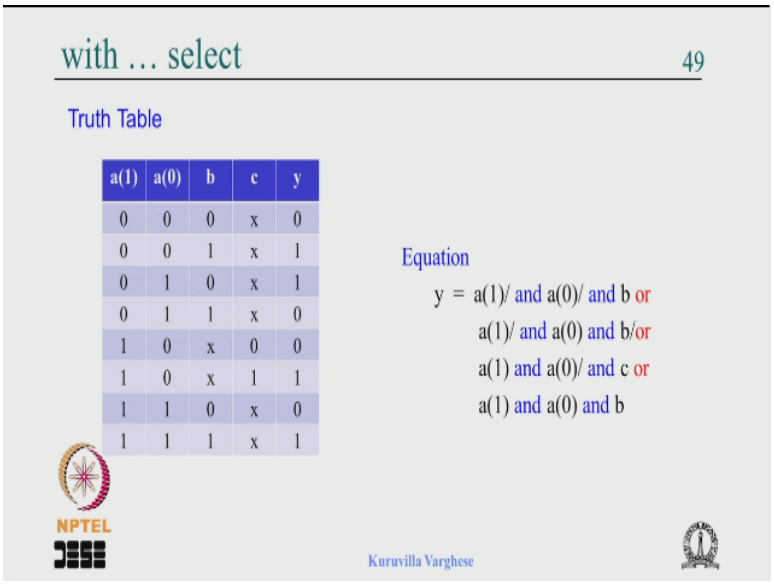
Now mind you this is if you do not write the tools are going to complain. Because you should know that assume that we are using standard logic, then each standard logic bits can take the 9 values. So if a is 2 bit as a combination it can take 81 values and we are only specifying 4 of them. And the simulator should know what should be the output if none of these condition match. So, you need to say what is like for 9 into 9 -4, 77 cases are not specified.

What is the output for those 77 cases need to be specified. That is what this doing, and this you could combine with the last one. There is no absolutely no need to write an extra kind of statement okay. So, basically the truth table of this looks like this you have a1, a0 because it is 2 bit 00 is 0, 01 is 0, 10 is 0 and 11 is 1. So, the equation is nothing but y is a1 and a0. So, this is the very simple case of with-select.

So, let us add some kind of expression and see what is happening, like we said some expression in terms of the input. So, let us write one statement for that. So, let us see this say with say with a select y gets b when 00, y gets not of b when 01, y gets c when 10, y gets b when 11 and when others. It really does not matter and this definitely helps in debugging and all that something goes wrong.

When you simulate like if you give a proper value then this can be easily kind of you know debugged from some the whatever, is happening at the output. So, that is a one kind of side effects of this when others. So, if you look at this you know that 00 is b, 01 is not b, 10 is c and 11 is b.

(Refer Slide Time: 44:01)



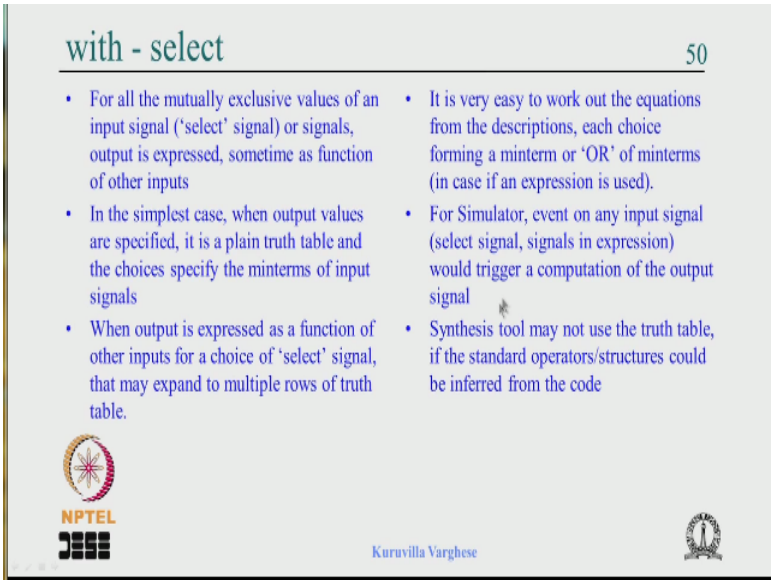
So, the truth table you have a as 2 bits a1, a0. You have b and c, so y is written in terms of this. When it is 00, you can see y0 when b0, y is 1 when b is 1, so that is the meaning of the first one y gets b when 00, y gets not of b when 01. So, when these are 01, b0 the y is opposite of that. Similarly come to the 10 when 10 is there the y is nothing but exactly same so, but only thing is that each statement you can see now.

Because of this inclusion of b and c find out in a multiple rows, and when it is 11 this is nothing but y is b, so 001 is 1. So, if you write the equation the equation comes like this, so here the equation is y gets a1 bar, a0 bar, a1 bar and a0 bar and b. That comprises of this 2 rows which is combined. So, basically equations comes at a mint terms okay. You like in this wherever there is 1, the mint term is there.

But here we do not know we are not walking with the numerical values of we have to write everything. So, this captures this particular 2 rows and the next like a1 bar a0 is capturing this, y is nothing but not of b, b0 y is 1, so, that is and b bar. So, that is this 2 rows and this or so, if it is 10 then the y is nothing but c. If it is 11, then y is nothing but b. So, that is shown here, so each of this kind of equation is translated to is coming from 2 rows.

In the simple case if there is numerical value, then it will be coming from a single row. So, that is how the truth table is formed in this case. So, let us go to another example. So that you have thorough you should not be think, you should be confuse about with select.

(Refer Slide Time: 46:13)



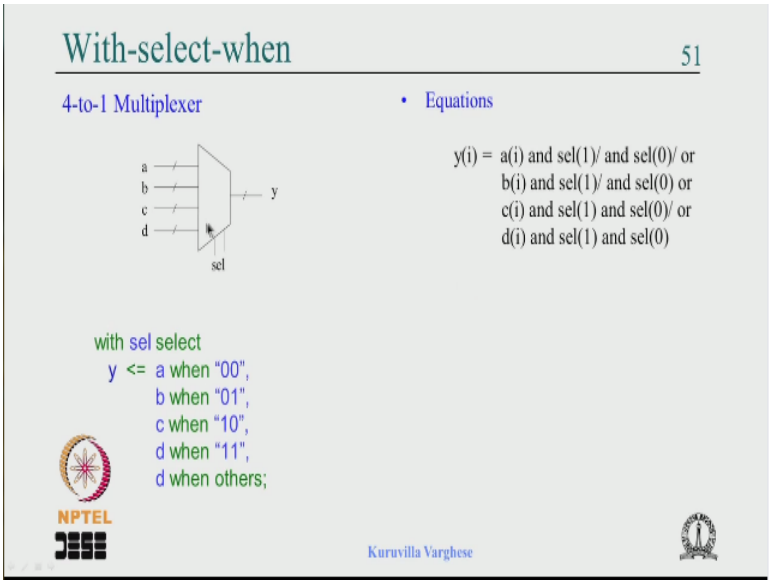
So, let us okay, this is what I have told now mind you one thing when you write such a code. If anything any event happens on the input okay. Like input can be a, input can be b then this the simulator will compute this particular concurrent statement. So, if there is event on a, event on b, event on c, anywhere this will be computed. That you should keep in mind.

But when it comes to synthesis tool it does not matter, looks at this code and you know come out with the implementation. Either truth table are through the operator whichever is a kind of ECR. So, that is what is summarise here for all the mutually exclusive values of input signal, output is specified which is nothing but truth table and each y is represented mint terms.

And the output equation is that mint term or another mint term. Because both condition cannot happen at the same time. so, you have or of mint terms, wherever there 1 happens. When output is express as a function of some input. Then this can be translated to each row. Each statement can be translated to multiple rows in a truth table. But in equation can be written as a single statement not an issue.

Similarly you can work out the equation looking at the truth table. And we said that simulator whenever there is an event on input simulator computes. But synthesis tool look at the the code and try to kind of generate the circuit.

(Refer Slide Time: 48:06)



So, let us look at the another example, which is a 4 to 1 multiplexer. So, you have 4 inputs a, b, c, d of some bit can be 4 bit or 8 bit whatever, or arrow going in a single bit and y is again the same sizes abcd. And select line is 2 bits, and now we write y in terms of the various values of select line. So, with select that is this. Y gets a, a when 00, when this is 00, a will go to y, 01 b 0, 10 c, 11 d, so that shows the kind of multiplexer implementation using the with select.

Now this can be little unfortunate, because most textbooks kind of specify this as a standard example. But the trouble is that people will think that from the syntax is this is something specific for multiplexer. But as we said that it is a truth table, we have specifying in some kind of basic level or little abstract level. So, basically any combination circuit can be implemented using with select multiplexer loops little natural for you know the syntax.

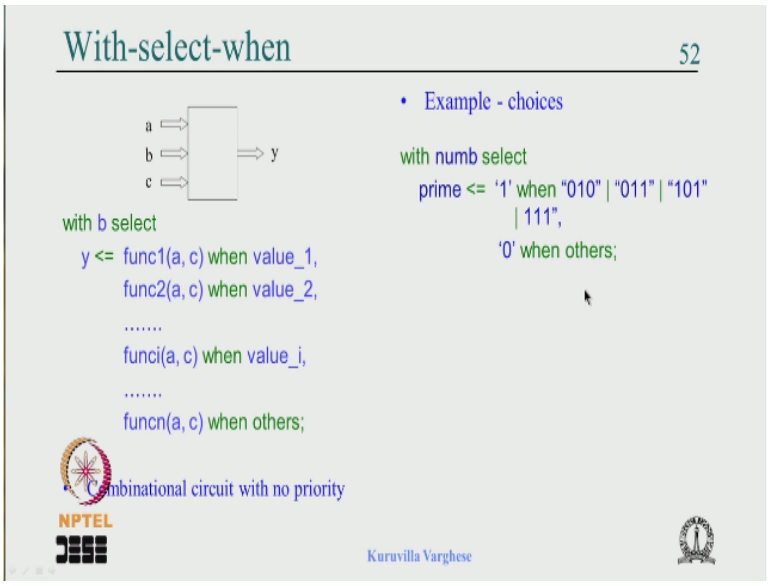
So, that is the multiplexer with select 00, y gets a and so on. Now let us derive the equation and you know that irrespective of the number of bits, the equations are going to be same. Suppose you have y3 to y0**.** Then the part between a3 and y3, a2 and y2 is same as the part between a1

and y1 and a0 and y0 and so on okay. So, the equation is that you go row by row, so you have yi is nothing but a of i like **y** if it is y0, a of 0, when select 1 bar and select 0 bar okay.

That is this or like b of i when select 1 bar and select 0 okay. So, that is the second kind of choice or the third and so on. So, basically you work out the equation as the mint terms or the product terms here or the next product term and so on. That is how the equation is define as I said. Absolutely this syntax has got nothing do the multiplexer, this is used for combination circuit.

And the multiplexer looks like a natural fit for this kind of syntax. But mind you should know always remember the case of multiplexer. The select line is going to the input of the AND gate and sometime the picture convey a message saying that this is something control in which when it is become 1, the input goes to output and so on like a the rest but then it is unfortunate because we know that the select as also an input abcd is also input okay.

(Refer Slide Time: 51:19)



So, in principle, so when I want to kind of summarise. So, with select it can be used in any combination circuit. Suppose you have multiple inputs. So, a, b, c and an output y, then in principle you can choose any of the input and you say with let us pick b, you can say with b select y gets. Now we have to specify the y because we are going to specify 00 of b. So, when you decode that mint terms properly.

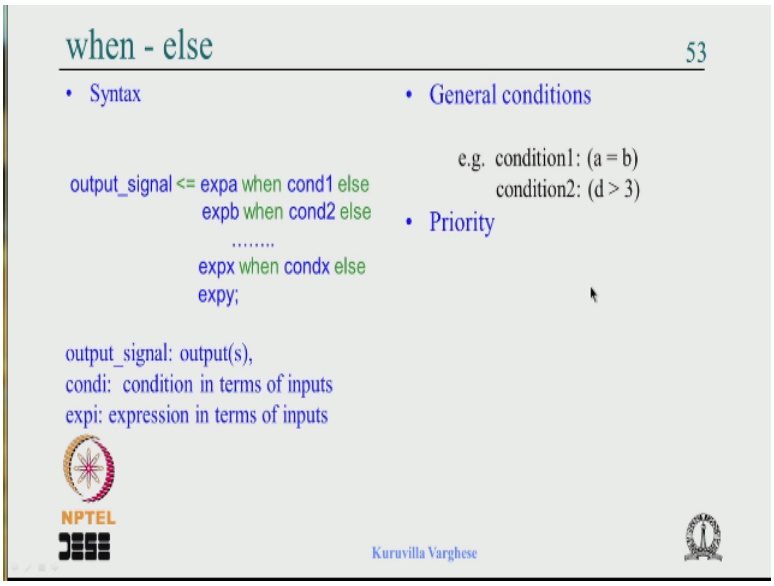
But when you write y now you have to write at the function of a and c because that is not specified. So, for this case when the value is 1 the y is function 1 of a and c. While use to another function of a and c and so on. Ultimately we say when others you know comprising of all the cases which is not discuss and in principle, you can even write a multiplexer with a as a select line okay.

So, there is nothing you can write say with a select y gets function of b, c, d and select. Various functions of b, c, d and select. It is like in the case of multiplexer if you do that, it is going to be quite composon. And it may not make much sense. So, it is natural that we use the select signal as a real signal of the concurrent statement. So, that is about the concurrent statement.

And this is used for combination circuit without any priority and you can also it is not that you have to specify. Suppose you have this function and this function at kind of saying. Then you can say same function when value 1 or value 2 okay. That is what is shown here, you can have multiple choices in a single statement, say here we say we have a prime number kind of detection for3 bit input.

So, dumb as a 3 bit kind of input, prime is output, we say when it is 2, 3, 5 and 7 the primer is high, so you say prime gets 1. When 010 or 011, or 101, or 111, 0 when others. So, you could use the choices in the case of which select to combine.

(Refer Slide Time: 53:58)



So, the next one is when else may be we will take that in the next class. Because we will not be able to cover it. so, a quick recap of what of we have done today , like this is what we have done the syntax with select and output signal is specified in terms of the mutually exclusive values of some input signal, the output can be numerical value or some expression of the various input signal.

And the rule is that you have to specify the mutually exclusive, everything mutually exclusive values, and when you say when others it captures everything which is not specified. And when you standard logic this is quite useful because there are lot of each standard logic position taking 9 values, so if there are multiple bits, then there are lot of unspecified thing which simulated like to know how to handle it.

So, that is why we specify that, and we have seen a simple example of an AND gate which is straight forward you know. You have the truth table and how the equation comes you write the mint term of wherever there is 1 or mint term of the you know if there is a 1 and other places. But here it is only in one place and we have seen a little more complex example, where the output is a function of not only the select line.

But the input then, you have the equations coming as a1 bar, a0 bar, and b and so on. So, we have seen a truth table, where a single statement expand multiple rows and when you write equations

again it becomes concise and we have seen that a multiplexer you know the code to be familiar. But as I said this is with select as nothing do the multiplexer.

In principle any combination circuit can be coded with the with-select. So, you have multiple inputs, you pickup one input write the output as different functions of other inputs for each values of b that is a the basic idea of which select and it is used. Whenever there is no priority, so any combination circuit without priority you can implement you can specify choices using kind of multiple choices using vertical bar**.**

So, I think I wind up today's lecture please go back and read that whatever we have covered in the data type. Basically the integer, physical, user define, then the arrays and record. Then we have looked at the syntax of with-select please go and revise it, write some simple program if you can with with-select. Very simple combination circuit you can take.

And write the write it using the with-select**.** So, we will continue with the concurrent statement and the sequential statement next class. We will have a look at the other concurrent statement called when else. Then we will follow it up the if then, case-when sequential statement and try to complete the combinational circuit part very soon. So, thank you, wish you all the best.