Digital Systems Design with PLDs and FPGAs

Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science-Bangalore

Lecture-20
Delay modelling

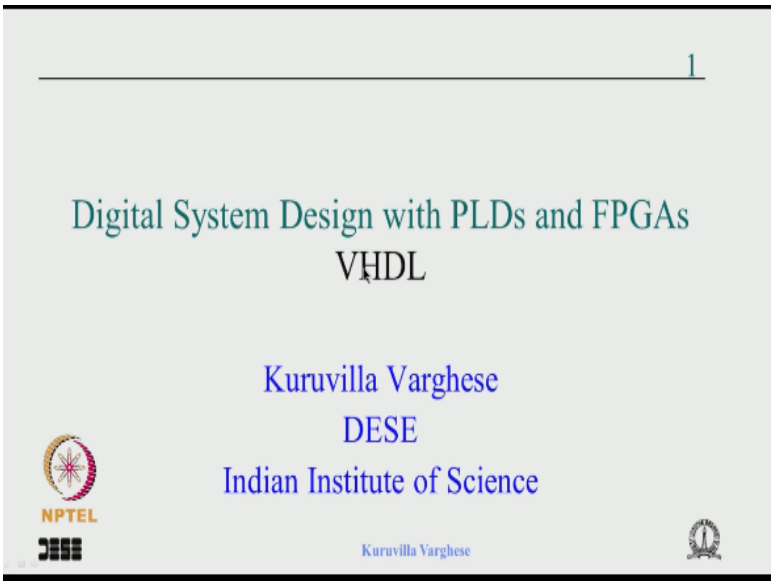
So, welcome to this lecture on VHDL in the course digital system design with PLDs and FPGAs. The last lecture we have seen various packages and various operators and how to use them and so on. And then we have started with the delay modelling available the constructs available in VHDL. We will complete that delay modelling today, also we will see some examples you know that we have discuss quite a lot of things.

And we will try to integrate to some kind of data path, sequential circuit, this whatever we have learned. We will apply we will take simple example. So, that is not that we have going to do big things. But the examples to illustrate what we have studied. So, that the basics are clear then you can do big things repair the complex things are nothing but many simple thing integrated put together.

So, there is nothing very serious about it. So, we will see simple things which illustrate the basics in the class**.** So, that we have limitation you know of then this slides suppose if I show you a huge code with which runs into some kind of 10 slides. You will not be able to keep track of it maybe when it comes to some case study.

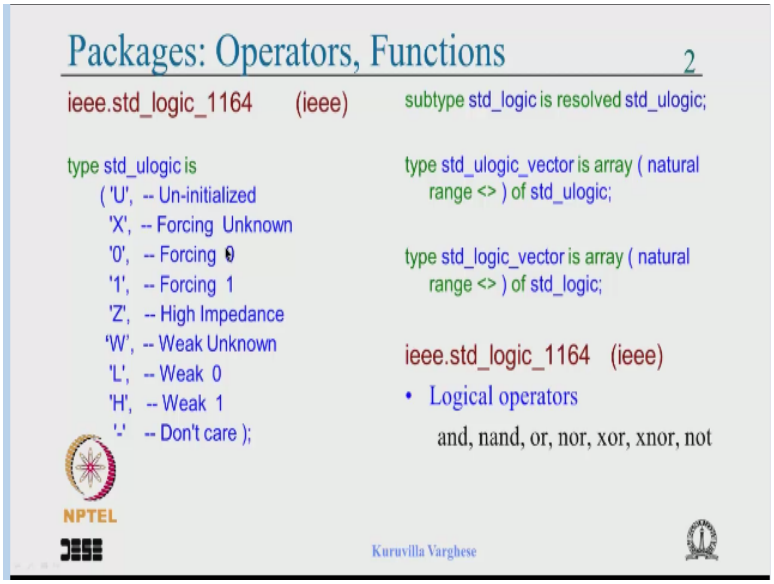
I will still show the whole kind of VHDL code for reasonably complex designs. But for the time being we will stick to simpler example which illustrates the concepts okay. So, let us run through the previous days lecture that is the packages, then delay modelling and then you will complete the delay modelling today**.** And see the examples or let us run through the slides**.**

(Refer Slide Time: 02:38



So, we have look at the packages.

(Refer Slide Time: 02:35)

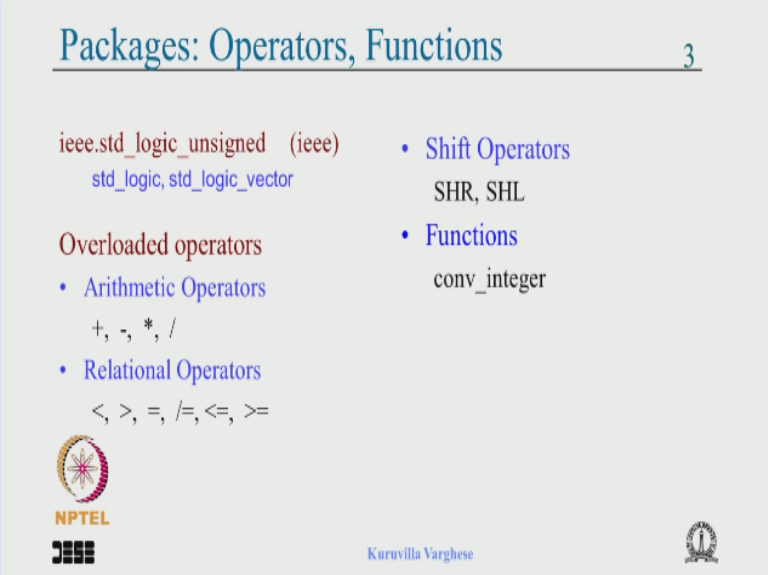


And this is the primary standard logic definitions were involved in defined in this particular package. So, you have standard u logic, standard logic, standard logic vector which is an unconstrained array, standard u logic take all these values as standard logic is a resolved standard ulogic that it resolves the multiple drives , standard logic vector, standard ulogic vectors are unconstraint array.

And what extra defined in this mainly are the logical operators in standard logic 1164, does not contain any arithmetic or relational operator. So, if you use only standard logic 1164 package you

will not be able to do much, you have to do very basic things or you have to build everything of your own. But the moment you start using other packages many other things will be available.

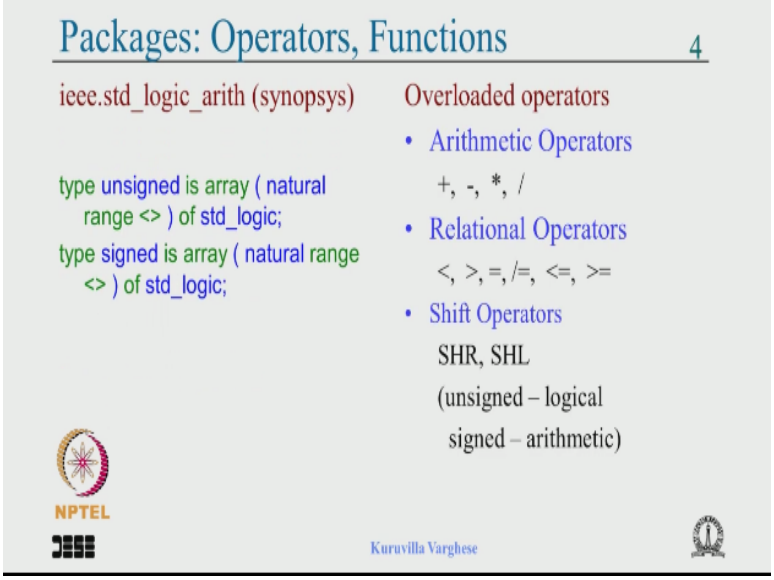
(Refer Slide Time: 03:38)



So, we have the next in line is a standard logic unsigned package that uses standard logic and standard logic vector, you have all arithmetic operators, relational operators, shift right, shift left and a functioned convert to integer okay. Because what extra defined other than the standard library is only the standard logic and standard logic vector.

So, what required normally is a conversion function from standard logic vector to integer, that is what this does you know you can given a standard logic vector you can given give this as a argument to this function. And it will return the integer which is very useful as I mention maybe to refer to some indexes of memory, array and things like that.

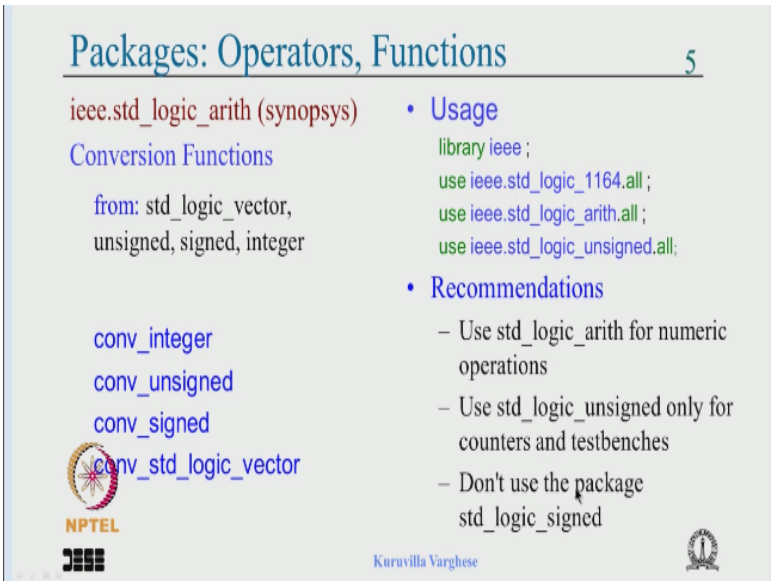
(Refer Slide Time: 04:42)



Next the package which is not ieee kind of define package is a synopsis package in ieee library. Now instead of standard logic vector it uses array, of standard logic as unsigned and signed. And you have also signed would mean it supports 2 complement for negative numbers. And you have arithmetic operators, relational operators you have exactly like standard logic unsigned.

As in the previous case you have SHR and SHL here . But this works as for as shift is concern. If it is unsigned data type, then it will be a logical shift, if it is signed then it is an arithmetic shift. So, it takes care of the basically the sign extension. So, if you have a compliment number if you write shift the sign is shifted along I mean this sign bit is shifted along. So, that the value is correct for that width you know that is a basic idea in signed you know arithmetic you know. Otherwise basically it is something do with a right shift not the left shift okay**.**

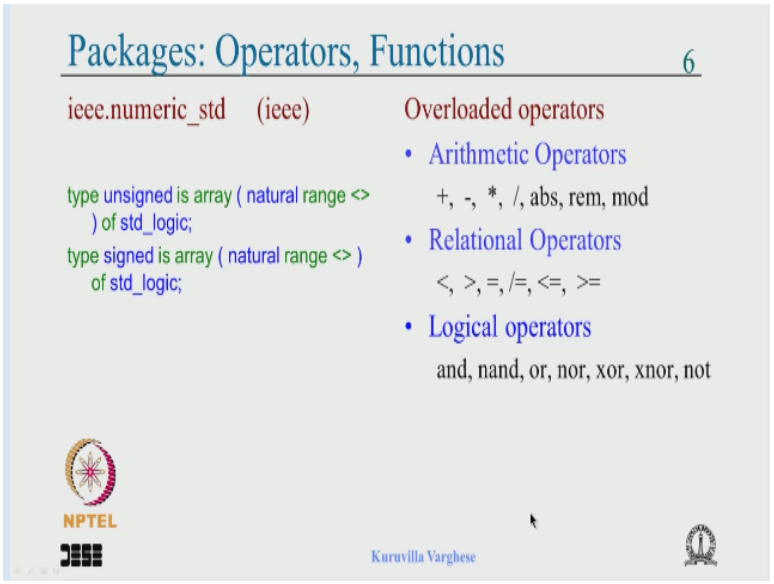
(Refer Slide Time: 06:07)



Then the next package is no there is a the conversion function in the standard logic arith package is this one covert integer, convert unsigned, convert signed or convert to standard logic vector. So, these converts like if you take conv_integer, it converts to integer all other 3. Similarly if you take convert standard logic vector then it converts all the other 3 to this things like that.

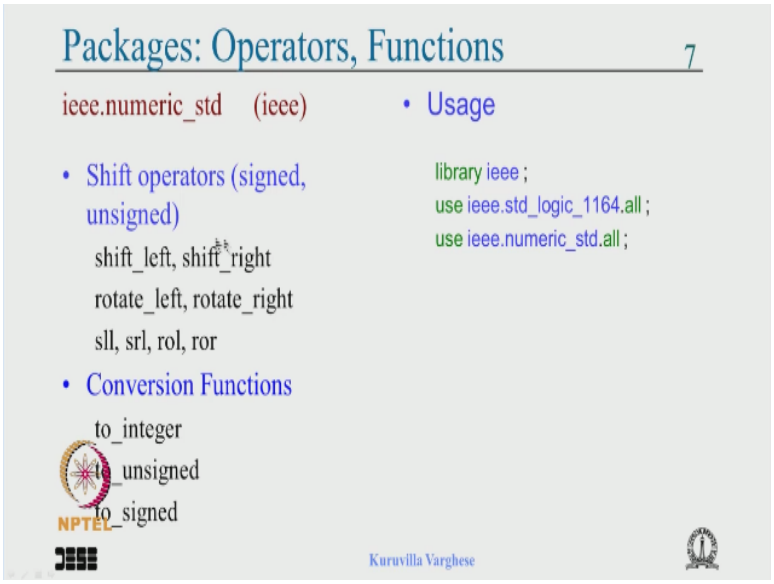
And the usage is this has to be in this order, standard logic 1164, standard logic arith and standard logic unsigned. So, there are some recommendations use, basically arith for numeric operation that is why it put at the top of the unsigned. And for counters and test bench you do not use this arith you put just use unsigned and do not use a package standard logic signed that all.

(Refer Slide Time: 07:06)



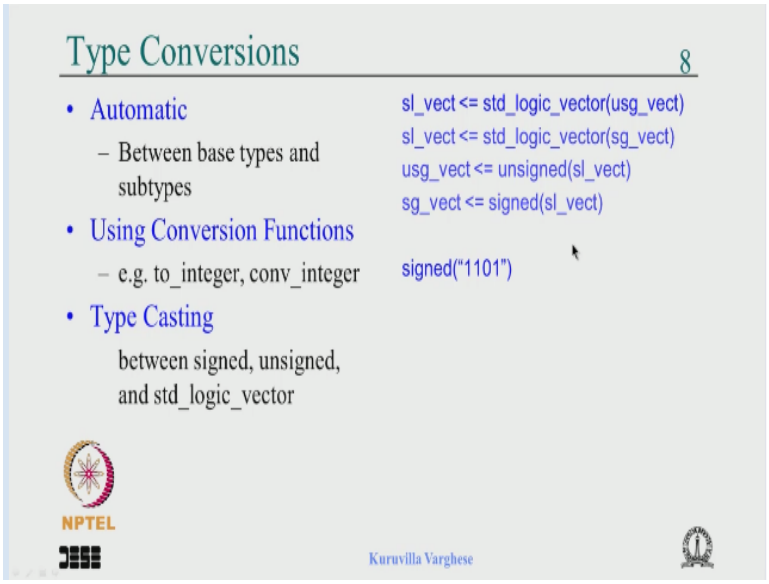
So, you have an alternative ieee numeric standard instead of using standard logic unsigned and standard logic arith you can use this, that also contains a similar definition unsigned and signed are the arrays of standard logic. Then you have arithmetic relational logical operators.

(Refer Slide Time: 07:29)



You have shift left shift, right rotate, left rotate right and you have the conversion function and this is a usage, you know you have the standard logic 1164 package, then you specify the numeric standard.

(Refer Slide Time: 07:43)

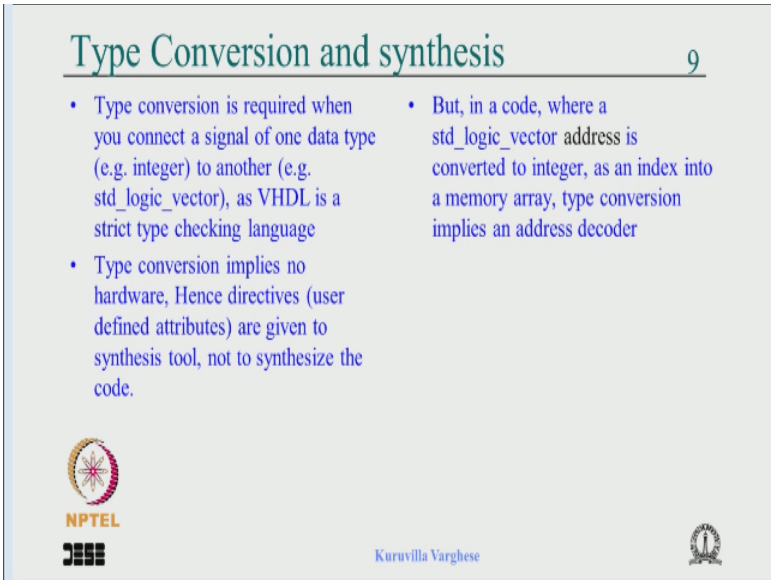


And when comes to the type conversion between the base type and the subtype it is you do not have to do anything. But if it is between say standard logic vector and integer you use functions

in the appropriate package like 2 integer, convert integer and so on. But thing to remember is that suppose you want to kind of do type conversion between signed and standard logic vector or unsigned and standard logic vector.

Both being defined as a arrays of the standard logic, you can do a type casting as in c, that means suppose you have an unsigned vector, you want to convert into a standard logic vector, you just say standard logic vector, (usg unsigned vector) then it is converted to standard logic vector. Similarly about the others and if you have a numerical value you can cost it to any of them by using these kind of syntax.

(Refer Slide Time: 08:54)

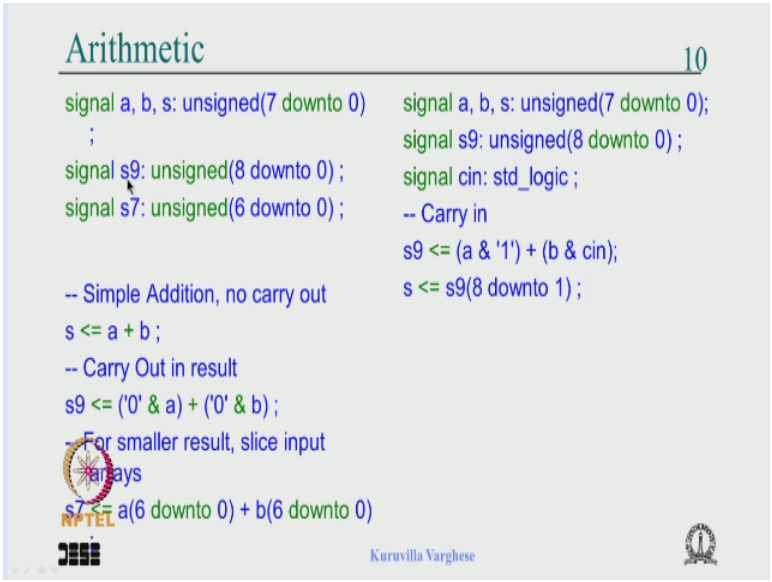


So, that is about type conversion, now mind you the type is a requirement of the VHDL. Basically to guarantee that the interface is proper like you have an 8 bit data bus, it is connected to only an 8 bit data bus not like you define something in integer, something in standard logic vector then if you connected together and one is 8 bit and another is 16 bit there is an issue. Similarly there is an issue with the most significant bit and lease significant bit and all that is kind of guaranteed by this.

And the code we write for conversion should not be synthesise. So that is indicated in the library, that synthesis should be off during that conversion and but I said when you convert an index many a times it can imply a decoder kind of thing. So, that is not to do with synthesis, but then it

is better to understand these things. So, that you have clarity which width which you can solve lot of problem. At least you do not get a false alarm.

(Refer Slide Time: 10:04)



And this is how the arithmetic is done, normally if you have a, b, s as 8 bit say s7 is 7 bit number, s9 is a 9 bit number. So, when you do simple addition like a a+b the result is also 8 bit you do not get a carry. Suppose you want a 9 bit along with a carry which is a requirement when you do any kind of shift and add in multiplication, you need the carry bit, carry out. So, in that case what you do is that you upend convert this a and b to 9 bit.

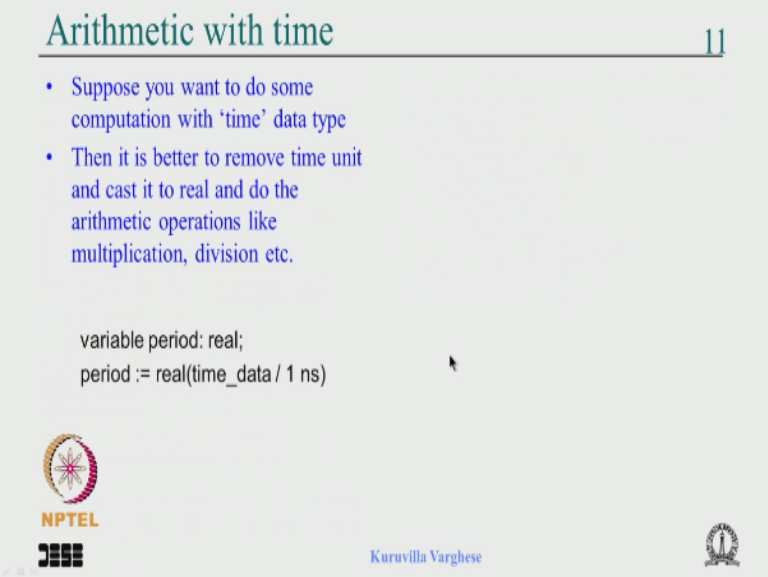
And the only way to do is that upend 0 at the msp so, 0 and a + 0 and b. Then these are 9 bit and you know that automatically the carry will come in the most significant bit. And that is assigned as 9 suppose you want only you have an 8 28 bit numbers you want only 7 bit result then you can take the 7 bit of a and 7 bit of b add it together. So, a6 down to 0 + b6 down to 0 is assigned to s7 and some case you want a carry input which is a required say you have forming a truth complement numbers.

So, you have to invert it invert some number and add 1 to 8 you know then you get a truth compliment number for kind of signed arithmetic then that can be achieved like this you have a and b are 8 bit and result is 9 bit. And what we do is that we upend the lease significant bit of a

with 1 and then cin is the lease significant bit of b then this being 1, if the carry in is 0 no issue nothing will be propagated to the next one.

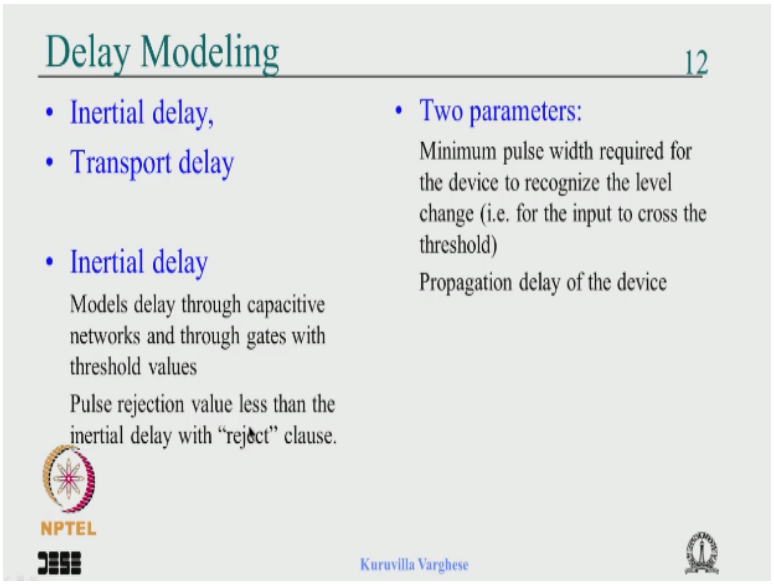
But if carry in is 1 this the result at the lease significant bit is 0 but there is a carry into the next place. So, we have to ignore that last bit because that was you know added to get the carry into the chain. So, we ultimately cost I mean are assigned s9 is down to 1 to the result, ignoring the 0 part because 0 part is just we have kind of manipulated to get the carry into the number okay.

(Refer Slide Time: 12:40)



That is how the arithmetic is use and when you want to do arithmetic with time because the time is a physical unit which is which uses integers okay. So, if you want a period to be multiplied for whatever reason then what you do is that you take the time data divide by 1 nanosecond. Then you get the number the without unit cost it to real then you get the period then. You can recast it back to the time at the end of it or many a times. This number is enough to keep on the computation so, that shows how to convert a time to real number to do some computation.

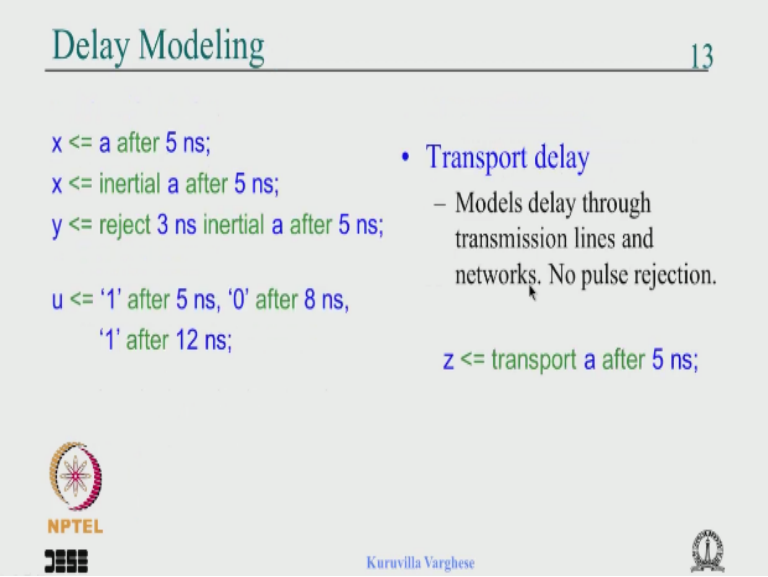
(Refer Slide Time: 13:22)



Then this was what we started last lecture, we have in VHDL 2 types of delay Inertial and transport delay. Inertial delay is what a delay through the capacity network so, if you have a on a bus say capacitor and you apply a pulse it takes some time to build up. And so, there is a delay okay or if you have a gate with the threshold value. You apply a pulse for the input charge above the threshold it takes a time.

And also propagate through the devise so, unless the input value exceed that threshold the output one comes. So, there is a minimum pulse width which is required for this any gate to respond okay. So, this inertial delay capture the minimum pulse width also the propagation delay both should be there.

(Refer Slide Time: 14:23)



And the syntax is suppose we say x gets a after 5 nanosecond it means that if a will appear at the x with the delay of 5 nanosecond. But here it is assume that you require a minimum pulse width of the 5 nanosecond for the output2 to com so, anything below 5 nanosecond is it makes no I mean it does not appear at the output okay. So, the another way of writing that is x gets inertial a after 5 nanosecond inertial being the default.

Even if you do not say it is same as inertial but if here the assumption is that the propagation delay and the pulse width as same. But if you have a different pulse width and at the gate level as I said it is difficult to model that you have to look at the transient level implementation do the spy simulation and get those parameters. But having if you know the parameters this is an accurate depiction of the reality okay.

So, y gets reject 3 nanosecond inertial a after 5 nanosecond which say that anything less than 3 nanosecond should be rejected it output does not come. But if it is greater than 3 nanosecond greater than or equal to 3 nanosecond, a will appear at the y after delay of 5 nanosecond. And we also said that you could write a continuous assignment like this, you gets 1 after 5 nanosecond 0 after 8 nanosecond, 1 after 12 nanosecond.

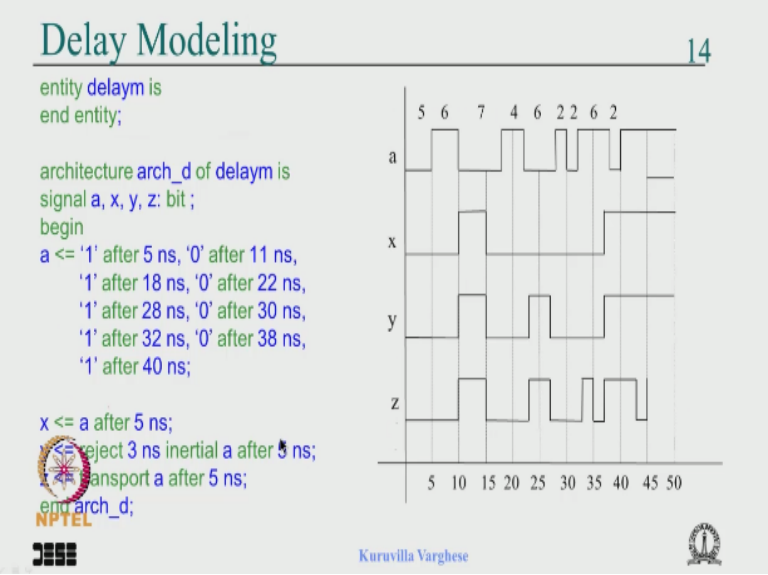
It means that from 0 to 5 it is 1, 5 to 8 it is 0, 8 to 12 it is 1, so, you have a pulse width of 5 nanosecond of the beginning then 0 width of 3 nanosecond again 1 width high bit high 44

nanosecond. So, this is very useful creating waveforms when you do delay modelling especially it is useful in applying some input waveform in the test bench. Because in a test bench we giving input to the various input of the entity and check the output okay.

So, this is quite a useful syntax and transport delay is that delay which is model the delay through transmission line. So, suppose you have a long transmission line even if you give a narrow pulse it travels all the way to the end but it does not reject any password so, that is done through this model. This syntax which say z gets transport a after 5 nanosecond okay. So, which just you know z is an exact replicate of a with the delay of 5 nanosecond.

That is the meaning of it that is what we want in a transmission line case and more than I mean do not think that we are going to probably model the transmission line yes it can be like in an ic you have a wire delay. And the wire delay can be kind of model a bus. But there are some kind of sometime some limitation with this when you do the timing model with these syntax. So, the transport can be use in such places

(Refer Slide Time: 18:16)



So, let us see an example the behaviour you know the behaviour of this inertial delay and transport delay. So, this is a code `I have written for simulating that. And you can write this code in 8 tool and try to simulate you will get the result I have not done a demo of a tool. But we will

do it in the coming lectures and first thing is notice that we are only doing a simulation and not a synthesis.

Because it is just pearly simulation this cannot be synthesise, because this talks about time delays and all that. So, there is no entity, so entity is empty. We are not defining any port we could define only signals I mean we can define signal that is good enough okay**.** So, entity is kept we give some name it is kept empty, the architecture we give some name and of this entity is and before the begin we declare all the signals we are going to use.

So, we have an a signal to which we assign some values and make wave form. And to x, y, z we apply these syntax of inertial delay and the transport delay. So, here you look a gets 1 after 5 nanosecond, 0 after 11 nanosecond, 1 after 18 nanosecond and 0 after 22 nanosecond and so on. So, to start with you get 5 nanosecond 1 pulse, 6 nanosecond 0 pulse, 7 nanosecond 1pulse and so on okay. when it comes here.

You see here the 28 to 30 is a small 2 nanosecond small pulse then 30 to 32 is also only a 2 nanosecond pulse and so on okay. Now x is we have written a after 5 nanosecond it means that x is a delayed version of a by 5 nanosecond but it rejects all the pulses below 5 nanosecond, when it comes to y it is again like x it is a delayed version of a by 5 nanosecond. But anything less than 3 is rejected not 5 now okay.

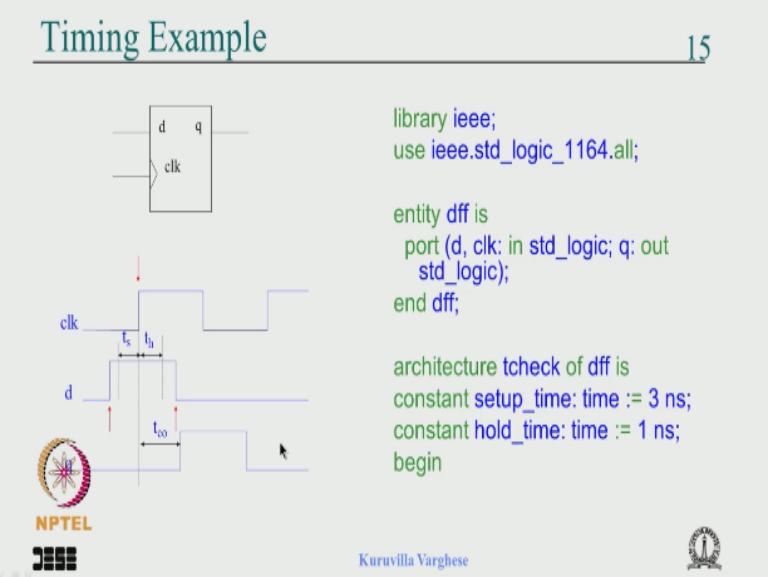
And z is transport a after 5 nanosecond and that is delayed version of a by 5 nanosecond no rejection so, if you simulate this you will get a waveform something like this. So, this is the original waveform you have created. So, 5, 6, 7, 4, 6 then 2, 2, 6, 2 and it remains okay, now if you look at the a az delayed version of 5 nanosecond. So, we have 5, 10, 15 time is shown here so, x is delayed by 5 nanosecond. But you see because minimum pulse width requirement is 5 is rejected this 4 goes this 2 goes and this 2 goes and you get like this 6 pulse width and this 6 combined.

Because this goes it gets merge with that and you get that okay. And the next one rejects below 3 nanosecond so, you can see it is exactly like a x, y is exactly like x but now the 4 appears, earlier

the 4 was rejected because of this 5 nanosecond so, the 4 appears and the last one is transport delay. So, z is an exact replicate of a with the delay of 5 nanosecond so, that is how the various delay models and when it comes to simulation.

I will show some kind of peculiarities what happens because of certain syntax sometimes it make give if you use this for a simulating somewhat is say not so real life effect, you should understand that we will see that when it comes to the some demonstration with the tool.

(Refer Slide Time: 22:46)



Next thing we are going to try is that we write a kind of timing check a VHDL code to check the timing of a flip flop okay. Once again this is used only while simulation it cannot be use for synthesis. So, you know that in a flip flop you have a clock coming and the data for proper transmission of data to the output. Data has to arrive at the input some time called set up time ts before the clock edge and also it should remain there.

The data should remain the sometime after the clock edge it should be stable during this window and this time after the clock edge is called hold time that means you hold the value at the input and the time before the clock edge is called set up time that means you set up the data before sometime. Now we want to write and if that happens the q appears at the output with the delay of t0 okay. Now in our code we are going to incorporate to check this timing violation that means.

We have to check when the clock comes, the set up time was correct or not that means data was set up sometime before, whatever the set up time before the clock and when the data changes, we want to make sure that the data is change after the hold time okay. So, the trick is basically we will whenever there is an event like a positive edge on the clock not the negative edge, what we will do is that.

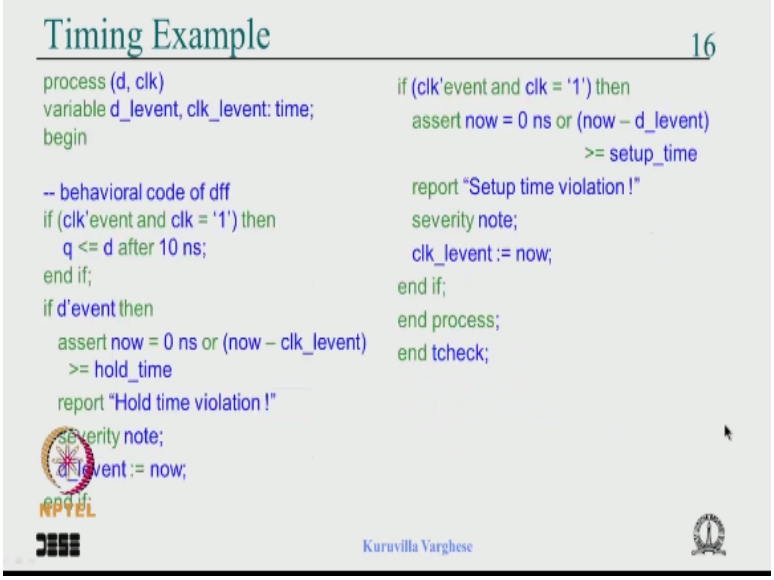
We will check the difference between this time and the last event on d if that is greater than set up time no issue otherwise you just flag a message saying that the set up time is violated. Similarly whenever there is a data change and of course we will update that time as a clock time clock last event on the clock. And whenever there is an event on the data we will check the that time – the previous event on the clock time is greater than the whole time.

If it is violated we print a message saying that it is violated and we also update this time as the last event on the clock. Because whenever there is an event on the clock, we have to make sure that the difference between that and the last event on the data as be greater than set up time. So, we will keep 2 variables one is to keep the last event on d, one is to keep the last event on the clock anytime an event happens.

We subtract the last event on the opposite check greter than that time that is the game okay. So, we will look at the code so, this is this is a code which can be use in principle for simulating the behaviour as well as for time check. So, the library package close entity is d in clock are input q is the output. Now in the architecture declaration region, we need to specify this set up and hold time. So, we have specifying for easy instead of hot coding it.

We could have said that this time – that time is greater than 3 nanosecond. But to make it easily modifiable easily configurable we are defining a constant. So, constant set up time some name is type time equal to 3 nanosecond. So, we kind of define the that constant as 3 nanosecond constant hold time is type time equal to 1 nanosecond so, then we begin.

(Refer Slide Time: 27:11)



And now you know that we write a process, wherein we write the behavioural code for d flip flop, and we also check what happens if there is an event on d. And if there is an a positive edge on the clock. Now if you write only the behavioural code you need only the clock in the sensitivity less, since we are right checking the event on d. We are including the d in the sensitivity less. Otherwise you will not be like this process will not be computed.

Whenever there is an event on the d, so, we define to variables d l event and clock l event of the type time to kind of remember the time the simulation time when whenever there was an event on the d and event on the clock. And then begin this is a process begin earlier was a architecture begin. Now we write the behavioural code for flip flop if clock tic event and clock is equal to 1. Then q gets d of the 10 nanosecond end if.

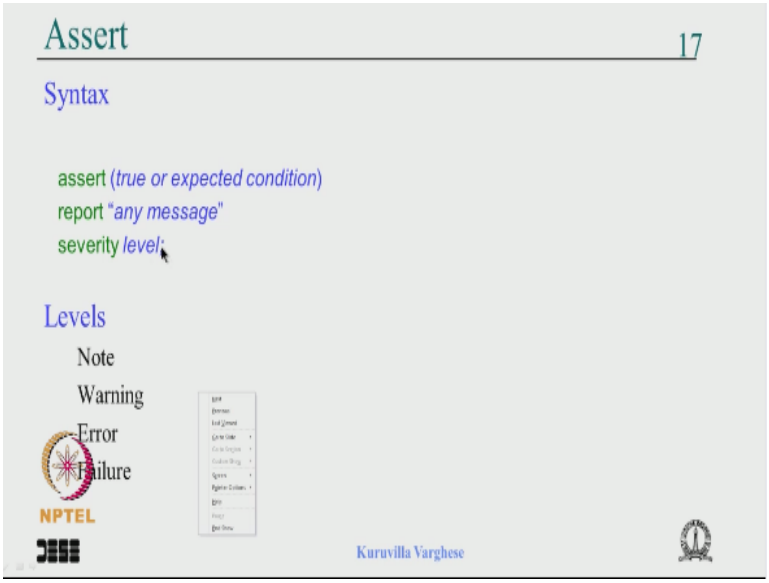
As I said when you write any code please include reset to make the code short. Because I have less screen space I am avoiding it so, that is the behavioural code and we are not like in this code. We are not checking that whether there is any violation or at the input actually if the input is violated. Then we cannot say q gets d after 10 nanosecond but our interest is in doing that checking the set up and hold time violation.

Then reflecting that at the output okay anyway the message will be printed so, we know that the output is gone not going to come. So, let us check the hold time violation so, that is a if d tic

event then now we are using a new syntax which say assert for a moment you forget about this. Now assert now is the simulation time the current simulation time okay, now means the current simulation time- clock last event.

So, that means that we are here there is an event on the d and we are subtracted are here it does not matter in anywhere we subtracted the last event on the clock which was told. And we have to check it is greater than hold time that is what it does now – clock last event is greater than or equal to hold time. So, the syntax of assert is like this.

(Refer Slide Time: 29:58)



You say assert some condition which is we are expecting it to be true we want it to be true. So, and report some message severity some level so, the game is that, if the condition is met it gives quite okay does not do anything but if the condition is violated, it will report that message on the screen. And depending on the severity say if it is there are 4 levels of severity is called note, warning, error and failure.

If it is note and warning it will just print a message, if it is a error and failure it is stop the simulation okay. So, that is what we are going to use here which a assert is now – clock last event greater than hold time. That means if it is true if there is no violation do not do anything, if there is a violation report hold time violation severity note. That means it will print on the screen and the console of the simulator.

And we have to know update the this variable so, d last event is now okay that is what we do similarly we check for the positive clock edge whenever clock tic event and clock is equal to 1 assert now that is the event on the clock – d last event. So, we are here so, there is an event on the clock – the last event on the d is greater than or equal to setup time. So, just if it is violated then report setup time violation severity note.

Then we have to update this variable so, clock last event is updated with the current time end if end process end time check. So, it is done so, if you give this code and if you give simulate assigning some clock to the clock input assign some input to d then if it is violated that will flag the violation or if this is interconnected to some other logic or registers and together you simulate.

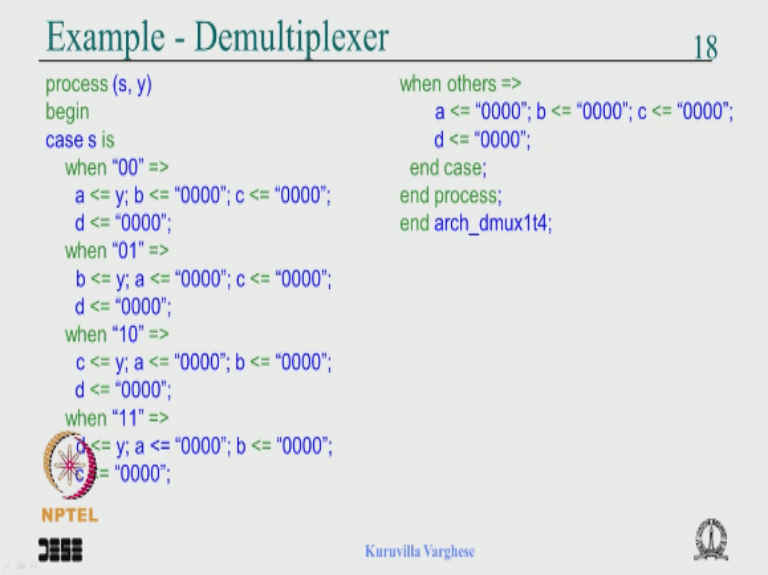
Then this again shows the timing violation you can catch the timing violation by message then you know on the otherwise you have through the waveform and something is wrong then you have to catch it. So, this is a one way of doing it one useful way of doing it. So, it illustrate this delay modelling or timing violation check just an example.

So, I think that is where we kind of a brief look at the delay modelling as I said there are two delay models one is inertial delay which is use for the gates which as two parameters minimum pulse width for the output to come a propagation delay. In the simplest case the propagation delay and the minimum pulse width is assume to be same which is somewhat kind of near the kind of real life.

But in need not be a kind of exactly the same and you can specify a different pulse width, then you have a transport delay which is which kind of model the wires of the transmission line the delay through that. Because it does not reject it appears at the other end of the transmission line and we have seen a waveform to kind of the study this effect by writing a code. Then we have seen a timing violation check on the setup and hold time of a flip flop.

We have learned a new syntax called assert. So, that is what we have done ass I said as we go along and we do a tool demo then I will illustrate some kind of peculiarities of this constructs. So, let us now since a I think I would like to soon go back to the digital design to complete little more their then we can probably come to this VHDL back again for a short term send and then we can go to the PLDs and so on. So, let us take some example.

(Refer Slide Time: 34:48)



So, we have seen the example code for a de-multiplexer so, I am not event showing the entity definition but if you remember if you refer to the last lecture this is a 1 to 4 de-multiplexer so, here the y is the input which is 4 bit a, b, c, d are the output that means it is a 1 which is de-multiplex to 4 output okay. Y is input abcd are the output s is the select line so, since there are 4 outputs 1 to 4 the select line is 2 bit okay.

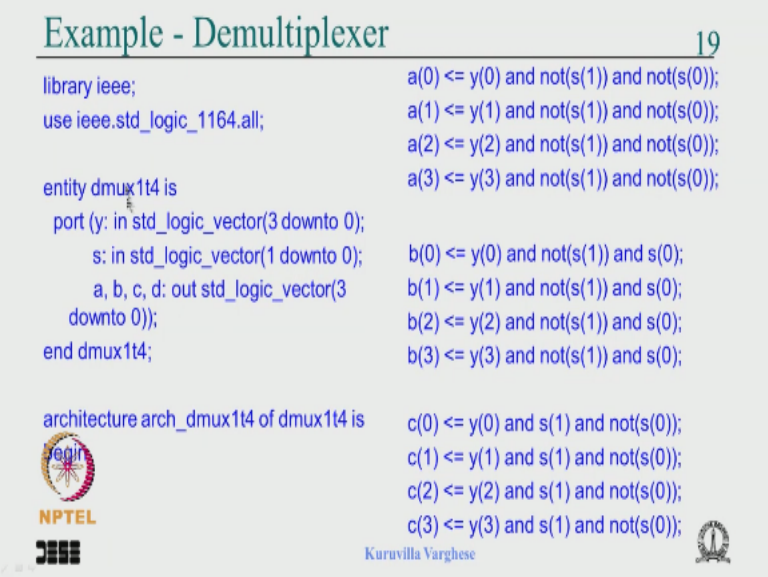
So, we have this code the code is that since select line and y are the inputs we write in the sensitivity less and process s, y begin. And the ideal the best syntax for it is case because there no priority and it for all the select line values. We get the various output so, that is what case s is when 00 a gets y or all others gets 0. 01 b gets y, you know all others gets 0, when 10 c gets y and others get 0. When 11 d gets y, all others gets 0.

And when others everything gets 0 but this is more of a kind of completion of the syntax z, you know but note matters to kind of the synthesis so, that is the code now let us look at all possible

ways of writing it, even we will write if using it if though the case is the ideal syntax just to get a practice. So, let us look at it the first let us write the simplest level the Boolean equation. So, you know that select line 1 is 0, select line 0 is 0.

Because you have s1 and s0 in select line then a gets y. So, the equation is a of i because there are 4 bits a0, a1, a2, a3 or a3, a2, a1, a0, y3, y2, y1, y0. A of 0 is nothing but y of 0 and s1 bar s0 bar okay. Similarly for other things when it comes b of 0 it is y of 0 and s1 bar and s0 so on okay, so we write the equation.

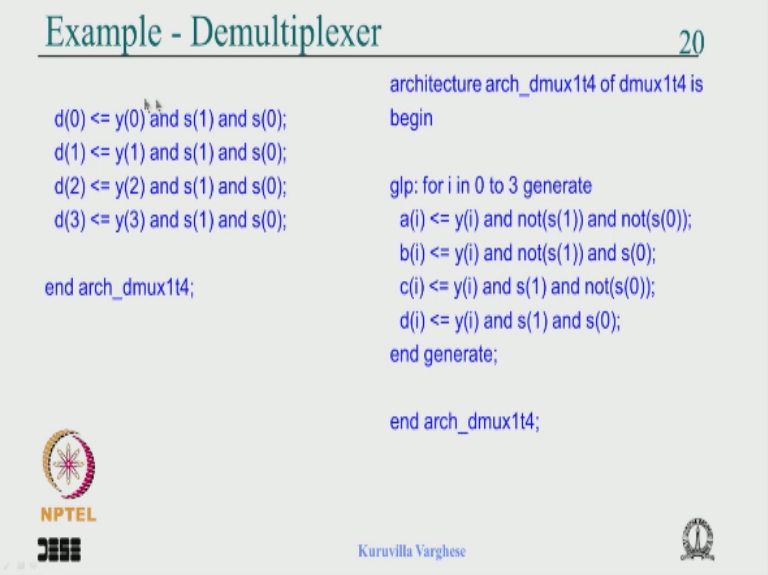
(Refer Slide Time: 37:25)



So, this shows a library the entity where y is input which is 4 bit vector, 3 down to 0, s is select line which is 2 bit vector abcd are the output which are all 4 bits. So, it is a 1 to 4 de-multiplexer with two select line architecture begin and simple. We write a0 is nothing but y of 0 and not of s1 and not of s0, similarly a1, a2, a3 b of 0 is y of 0 and not of s1 and s0 all others are similar except for you know these index changes of 0 is nothing but y of 0 and s1 and not of s0.

Because this is the 10 and when it comes to d of 0 it is y0 and s1 and s0 okay. Now this clearly shows if you write you can write equation but there is no point in repeating that this syntax is very neat a0, y0, a1, y1, a2, y2 a3, y3 and this remains a same. So, we can write a generate loop for this, this, this and the next one okay.

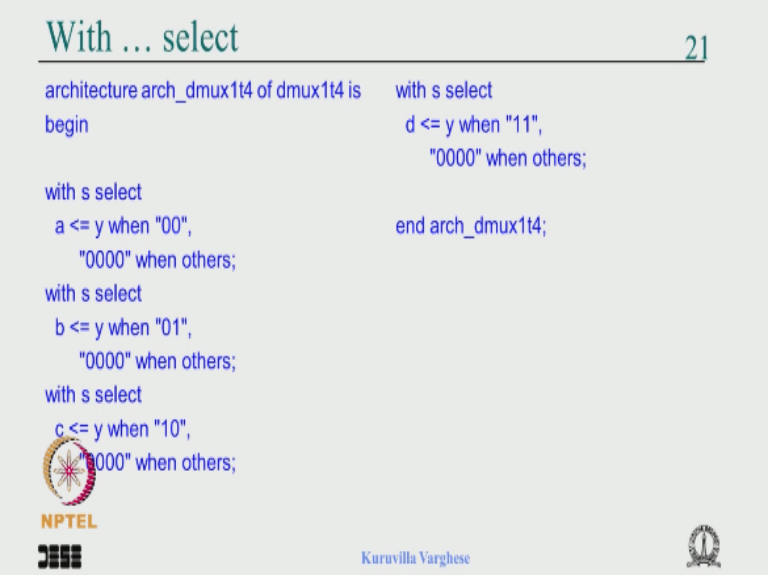
(Refer Slide Time: 38:36)



So, that is what in the architecture now onwards I am not going to write the entity and library and all that. So, you could write a generate loop which is say a label GLP for i in 0 to 3 generate and n generate in that we say a of i is y of i and not of s1 and not of s0, b of i is yi and not of s1 and s0, c of i is yi and s1 and not of s0,d of i is y of i and s of 1 and s of 0. So, this is simple enough I mean you can write an equation and if you are a comfortable with it.

Though there is nothing wrong with the earlier one you know that is nothing but at truth table and we have seen how the equation of the case s and that is exactly similar to this. So there is no absolutely not difference between this equation and the case statement so, that is another way of writing which is mainly kind of playing with the syntax so, that we are through.

(Refer Slide Time: 39:53)



And so, let us try with the concurrent statement the same thing now you know that when we use concurrent statement with select and when else unlike case and if you can only specify one output okay. You can say a but you cannot say a and b together okay so, the one main problem with the concurrent statement is that. You have to write a code for a another code for b, another assignment for c and d okay.

So, let us use the with select so, here we say with s select a gets y when 00 that means when s is 00. In all other case is it is 0 a0 when others okay. So, that is the code for a similarly for b you have to write with s select b gets y when 01 00when others with as s select c gets y when 10. 0000 when others and with the select d gets y when 11 00 when others there is a absolutely a nothing wrong with this coding .

Because you know that though we say 1 to 4 demux if you look in sight circuit for a, circuit for b, circuit for c, circuit for d can be separated. Only thing is that abcd all are the functions Boolean functions of y and s okay. So, you did natural like you know that a, b, c, d all are the functions of the same a, b, c, d it is very natural very good that if you put it together in a case statement or if statement.

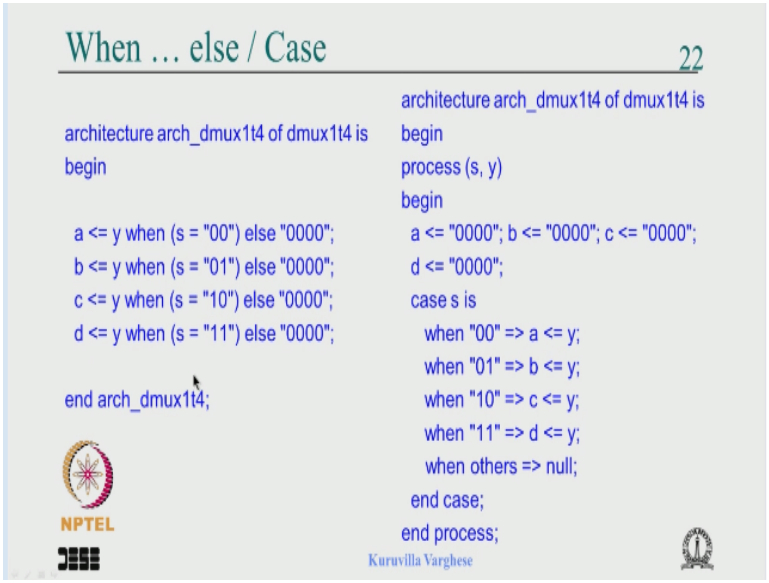
It is very easy to understand then write it separately okay it depends on synthesis tool as for as the circuit is concern there is no issue that it should kind of you should get the same effect. But

definitely there is an issue that synthesis tool should understand that this together is a function okay. So, maybe a naive very simplistic kind of synthesis tool will give you the correct result, correct circuit.

But it may not infer that it is a 1 to 4 mugs that could pour some problem when you it will be good, if synthesis tool knows that is a 1 to 4 mugs. Because when ultimately you try to implement that in an FPGA using primitives you can map it to the correct primitive, if you do not know that it is a kind of multiplexer or de-multiplexer to kind of to place it to the correct primitive to map it to the correct primitive it could be a problem .

But otherwise there is no issue with it I have in kind of looked at it whether synthesis tool you know infer that it is a mugs I am sure it more synthesis tool will do that. But you can check it.

(Refer Slide Time: 43:19)



Like let us see the when else exactly like with select you have to write the four concurrent statement for when else. So, it is say a gets y when s is 00, else you know 0 okay. b gets y when s is 01 else 0, similarly c and d exactly same effect though a, b, c, d, y are functions of same i same s, we write it separate . But you know that is one kind of issue with this kind of coding.

So, I would suggest that you stick with for my opinion is this is the best code, of course we can have a concise version we will see that. And you could use yeah this is the concise version. So,

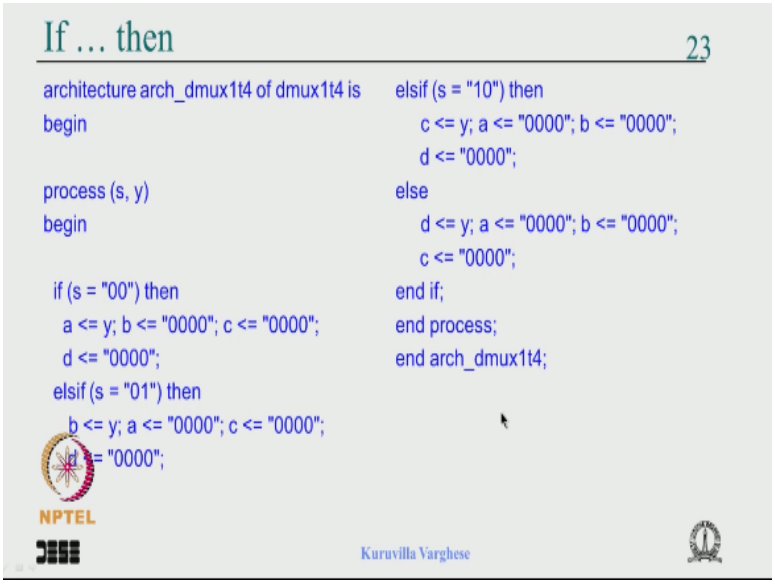
we initialise a, b, c, d to all 0 at the beginning and you say case s is when 00 a gets y rest all is 0 here. Since whenever there is an event on s or y. The process goes from top to bottom.

Say here a was assigned to 0 at the beginning for say the event happens at 100 nanosecond a was assigned at 100 + delta. But when it comes here a a is reassigned with y. So, that is replies with the y, so that no issue. This is the very nice code you know when 00 i gets y, when 0 get 1, b gets y, when 10 c and so on. And here you can instead of when 11 you get say d d gets y I have said when others you know it is null.

But you could combine others along with this it is better to be precise with coding then take liberty. So, in my opinion others here, that is a better coding then writing like this. So, this is perfect but then you can write if also. Because instead of case if you write, if though it kind of as a kind of priority but since these are mutually exclusive and if you specify all the condition in if it will kind of it is redundant like you say.

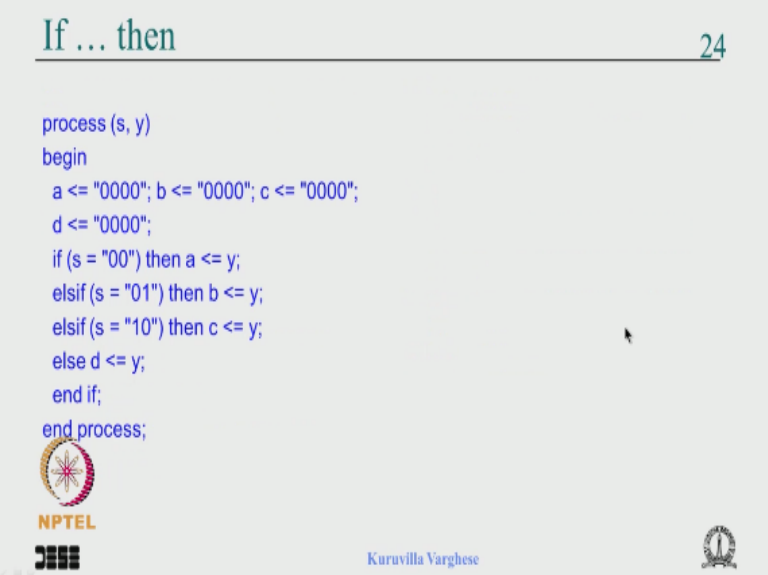
When 00 like if say if 00 else if 01, but else if 01 is not of 00, but not of 00 will kind of when you make the truth table and kind of you know optimise it everything comes back to the same thing. So, you could write with the if there no issue.

(Refer Slide Time: 46:08)



There because a priority will be thrown out because all the conditions are specify it will default to the same as whole. So, in principle you can write in the process sy instead of case you can write, if so, you say if s is 00 then a gets y all others get 0, else if s get 01 then b gets y, everything gets 0, else if s is 10 then c gets y everything get 0, else d gets y and see this else will capture everything else and there should not be an issue.

(Refer Slide Time: 46:42)



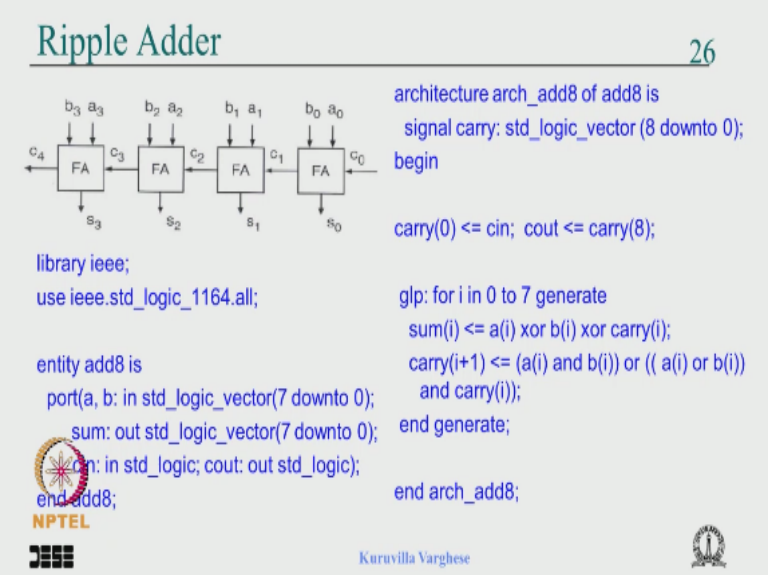
And similarly you can have instead of repeating all the other signals here, you could write you know initialise everything at the beginning. And only what is changing could be written like in the case, and so, these are kind of different ways of coding. But you know that like you can use if but the case is better, you can use with select and when else. But definitely the case are if is better in that the case is better in that.

There are two ways of writing it one is very concise less error easy to understand so, I think you should be essentially using that so, sometime when you are face with I think it is important that you stick to the correct coding. So, let us take another example let us come back to the coding, let us take in example of an adder we will look at a ripple adder the code for ripple adder we will also look at the code for a carry look add adder.

And how to write the adder though we may not use a carry look add adder anywhere in our design it is not required because huge it is exponentially complex in terms of the product terms,

you so, we do not use it. But then it is a good example good practice for writing the codes so, let us go to the slide.

(Refer Slide Time: 48:17)



So, let us look at this is the ripple adder say I am considering I am showing the picture of a 4 bit but we are good trying to implement an 8 bit adder so, I am only showing the 4 bit you know that, this is the full adder you have a0, b0, c0 as a input. You have some as the output and the carry as the output to the next stage, and here a1, b1 comes some 1 is a output carry2 is the output to the next state, now that is the equation.

Some is some 0 is a0, xorb0, xorcarry0 carry1 is a, a0, b0 or a0 or b0, and carry1 carry0 okay. So, basically the carry1 is generated if both are 11, then definitely there is 11 1+1 is 0, there is a carry or if either of them is 1, then it is 01 is some is 1, a carry input 1 will make it 0 and generate a carry that is the game. So, this is many a times is called generate a, b is generate, a or b is propagate okay, if a and b are 1 it generates a carry from this stage.

If a or a and b are kind of either of them is 1, then it propagates the previous carry to the next stage, and that is how the rippling happens suppose this input is 1, and this is in propagate mode, this is in propagate mode, this is in propagate mode. That means you know you understand what it means it is all 101010, then this there is a rippling there is a delay through this you know and or circuit.

Then that goes delay through the true level, and or circuit delay through the true level and or circuit. So, you get by the time the carry input comes here c3 comes here there is 6 levels of logic and it delays everything. This sum is delay because this sum is going to uc3 so, when it comes to the fourth one it is much worse okay. So, that is the ripple adder so, let us look at the code first then we will go with the carry look ahead adder.

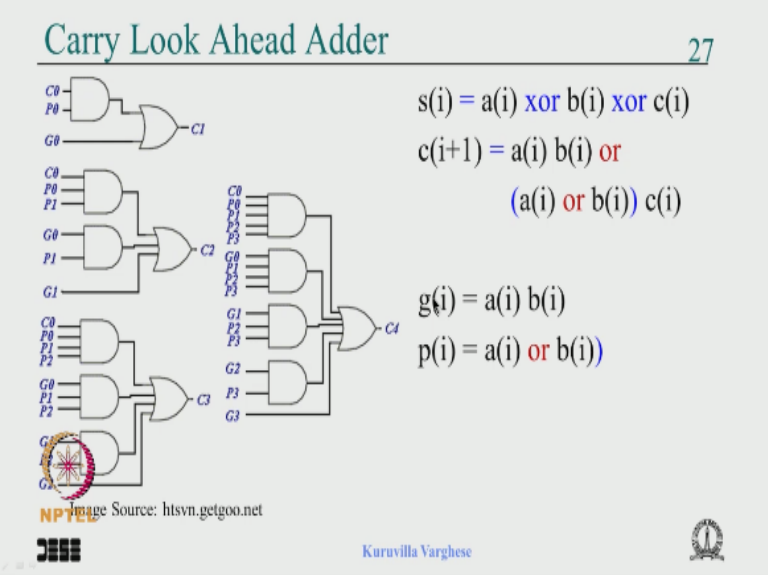
So, here there is library and we are going to write a neat code so, we have a carry in and we have a carry out. So, we are going to have a0 to a7, b0 to b7, some0 to some7 a carry in and a carry out okay. So, that is it port a, b is 8 bit, some is also 8 bit, but we have a cin which is a carry in standard logic cout out standard logic okay. Now when you write the equation it has to be elegant now if you start**.**

We are going to write a loop generate loop to say if you keep this as cin there is an issue, because then this that cin part has to be outside the loop because that is that will not work. Because then a0, b0 should be kind of combine with the cin so, it will be good if you declare an internal signal carry. Because this is an internal signal and we say carry 0 is cin okay, similarly we have carry 8 ultimately we assign into the cout.

So, that is what we have done signal is carry which is standard logic vector 8 down to 0 now so, because we have c0, c1, c2 all the way have to see 8 which is a 9 bit vector that is it, we say carry0 is nothing, but the cin, cin we are assigning it to carry0. It essentially like a changing the name you know there is no as such there is no this assignment does not mean anything, It is just we changing the name similarly the carry8 we are assigning to cout.

Now we can write in neat loop for GLP label for i in 0 to 7 generate, we say some of i is a of i, xor b of i, xor carry i. Carry i + 1 is ai, bi or that is generate or ai or bi propagate and ci. So, it becomes very neat if we had not made this assignment not define the carry like this. We will not able to write everything in a loop something has to be outside so, a very time to you write a code make sure that you define the signal such assignments. So, that things are very generic kind of concise elegant and things like that so, now let us look at the carry look ahead adder.

(Refer Slide Time: 53:30)



So, in the carry look ahead adder the game is that what we are trying to do is that in the ripple adder, when it comes to the carry2 first the carry0 is generated with a0, b0 and sorry carry1 is generated with a0, b0, c0. And that is used in this a1, b1 to generate c2, now what we do is that we kind of when it comes to c2 instead of rippling through what we do is that we expand this c1, you know what is the equation.

Equation say, that carry1 carry2 is a1 b1 or you know a1or b1 and carry1. So, if you just expand it in terms of the carry0 and flatten it, you will when it comes to carry2 will generate it straight away from b1, a1, b0, b1, a1, c1 b0, a0, c0. So, that it is a true level structures so, instead of kind of what to say kind of modular adder this is an adder which is this is it is also surprising ,you know it should many a times people do not question like any combination circuit.

You design you know by from the specification you write a truth table and you try to implement it, but surprisingly when it comes to the adder we make a 1 bit unit then you use that modular unit to cascade and build kind of ripple adder okay. But that is not what we have use in a basic combination circuit design, the essential thing we used to do as that make a truth table and work out equation and try to implement.

And frankly if you have a 8 bit adder you write an 8bit truth table yeah it is quite complex, then you make the equations out of it you try to implement you will get a carry look ahead adder. So, in at least if you are kind of doing it properly in any learning the carry look ahead adder should come first and we should be saying that there is an issue with such a scheme. Because it is a exponentially complex in terms of area and then.

So, we adopt such a ripple adder it is not that the ripple adder comes first from the principle, and the carry look ahead adder come second definitely the carry look ahead adder comes first. And because of these particular issue, we default to the ripple adder that should be understood so, maybe we can quickly look at the carry look ahead adder structure in the next lecture maybe we will look at it in detail.

And **m** kind of look at the VHDL coding which is quite straight forward but then it is a good exercise to do. So, let us quickly look at the carry look ahead adder so, this is basically say the c1. So, the equation is s of i is ai xor bi xor ci no issue with it, I mean for each stage that the sum is same. But main issue is with the carry which it you know when it is modular it when it cascaded it ripple through.

So, we are trying to kind of expand it and try to bill so, as c2 comes we have building a0, b0, a1, b1 when c3 comes it we have building you know it into terms of a0, b0 a1, b1a2, b2 and so on okay. So that is the game maybe that we will see in detail in the next lecture so, the last part we have looked at the various coding and example we have tried various syntax of de multiplexer.

I mean we applied for the case of de multiplexer various syntax though there is an straight forward easy elegant, solution we have looked at the ripple adder code reasonable code once again we will not be using any of these. But then as I said is a practice. We briefly looked at what is wrong with the ripple adder though as I said the otherwise should have been a better thing to do. And so, I windup here, please revise it and I wish you all the best and thank you.