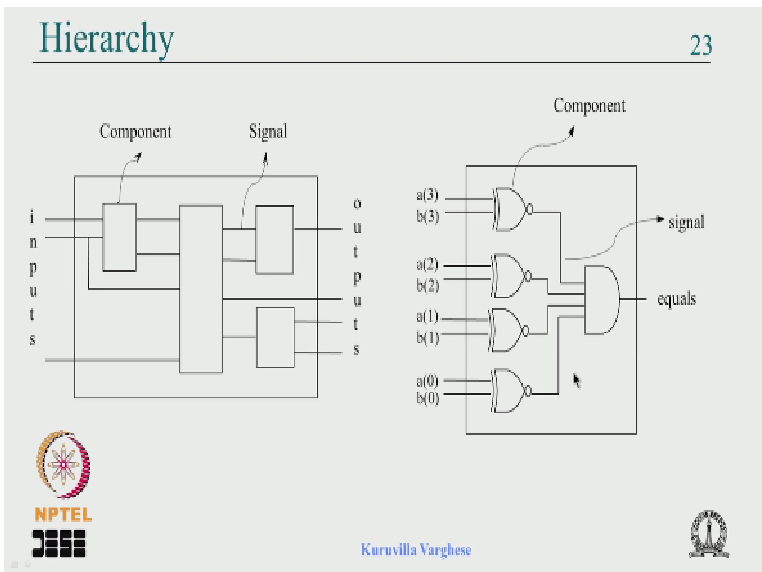
**Digital Systems Design with PLDs and FPGAs**
**Kuruvilla Varghese**
**Department of Electronic Systems Engineering**
**Indian Institute of Science-Bangalore**

**Lecture-12**
**Simulating Concurrency**

So, welcome to this lecture on VHDL the course digital system design with PLDs and FPGAs the last lecture we have seen structural model of according in VHDL then we looked at the different kinds of simulation and we have taken looked at how at least in timing simulation the simulator try to resolve the concurrency by looking at the sequential written state concurrent statement. So let us quickly run through the slide before we get into today's lecture.

**(Refer Slide Time: 01:08)**

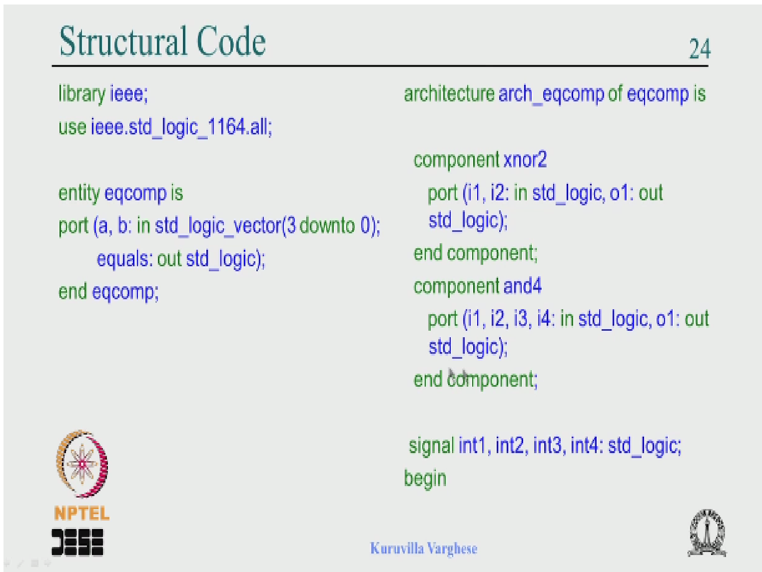



So, **m** this is what we have looked at when in a when you have a structural coding which is use for kind of top down design. You have a top level entity with names and the ports this is composed of some interconnection of components. So you need to declare the architecture declaration region the various components used that basically a name again the input output port and various internal signal.

And in the statement region we instantiate distinct components as required and interconnect them. So we somehow show a necklace that is what we have seen. We have taken the same equality comparator as example though in real life you do not need to do you do not do a structural coding for such a small circuit one statement is enough and concurrent statement is enough. Here the components are the two input XOR gate and 4 input AND gate.

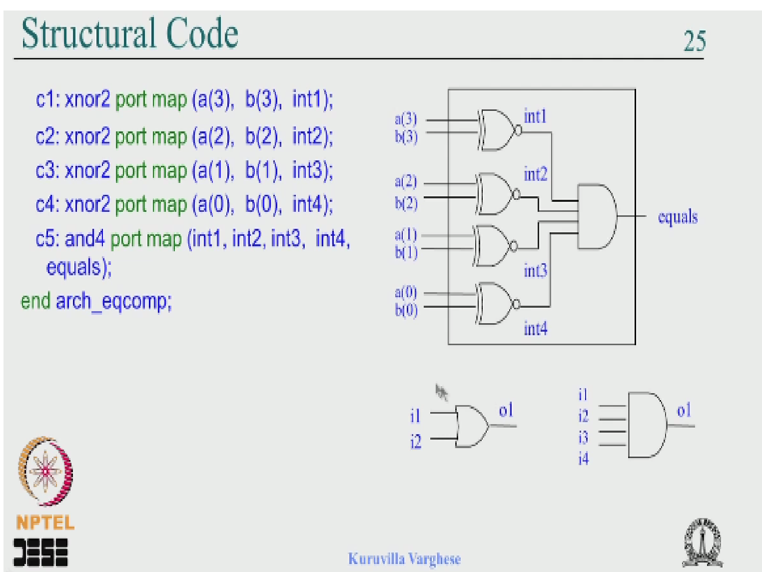
And, there are 4 internal signals so we are going to declare these two component this signal and in the statement region.

**(Refer Slide Time: 02:23)**



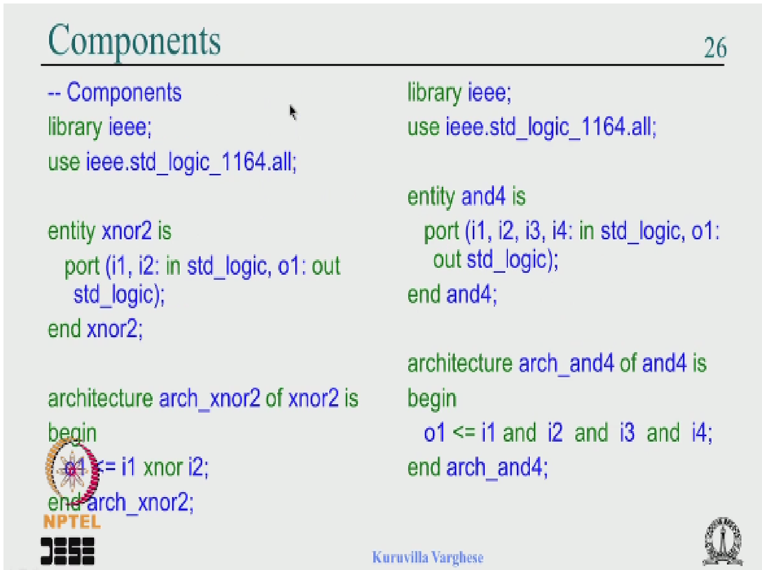

Describe the necklace so that is what shown here this is a library entity in the architecture declaration region. We are declaring the XNOR component with its port and for component with the ports for internal signal.

**(Refer Slide Time: 02:37)**



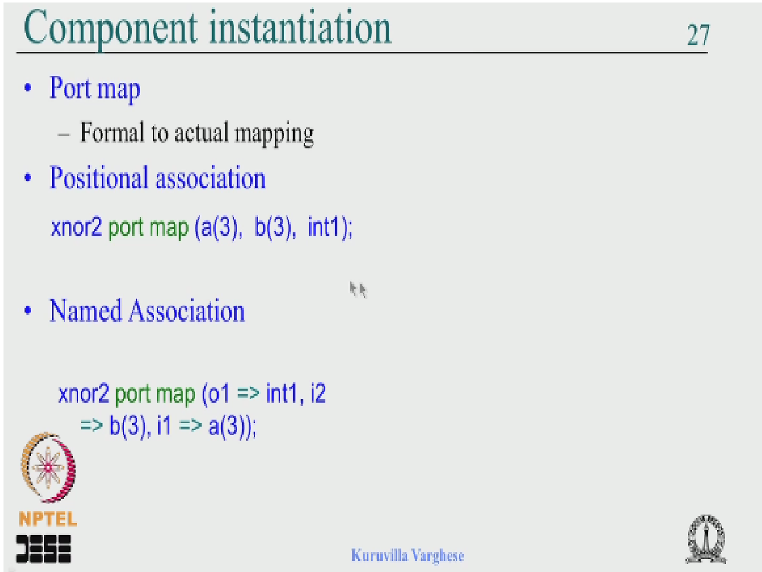
And the statement region we have forming this necklace by instantiating and forming the real formal to actual pin mapping that means i1 is map to a3 i2 is map to b3 and o1 is map to int1 and so on for all the four components. And, this is called positional association the formal and actual are associated in a positional way.

**(Refer Slide Time: 03:08)**




And ultimately you have to sometimes somebody has to describe the components may be it is done before it is in a library it is in a different project or its in a same project in a different file on so on. But this shows a component definition the entity for XNOR architecture similarly for the AND gate once this is done the structural coding cum complete. You can simulated you can synthesis it.

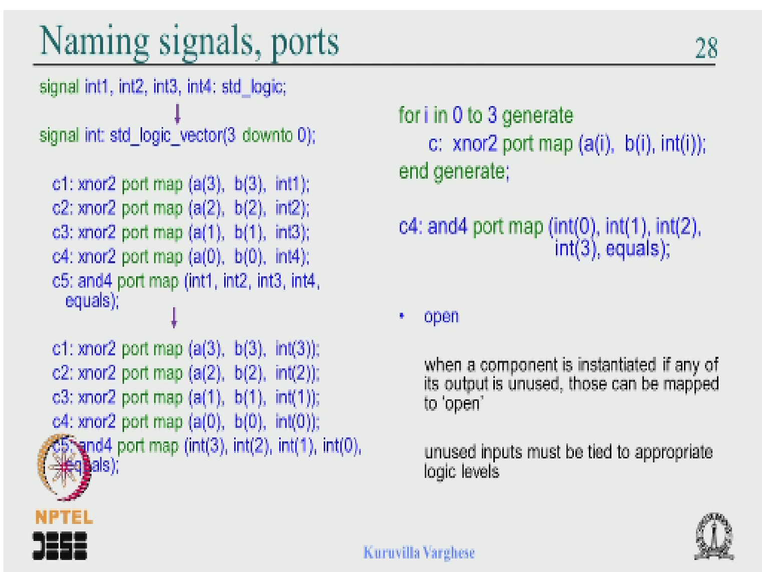
**(Refer Slide Time: 03:38)**




And I said about the positional association which is very difficult in a complex case to remember the order of the ports particularly when you put this into a library it is enough if you like normally people say what are the input and output port what are the data types. So a convenient way of associating is to exclusively give the formal to actual mapping. So one need not kind of keep any order.

You specify what is the formal parameter a which is define in the component and what is actual, actual signal which it is mapped. So, that is what shown here so very convenient way doing it.

**(Refer Slide Time: 04:23)**

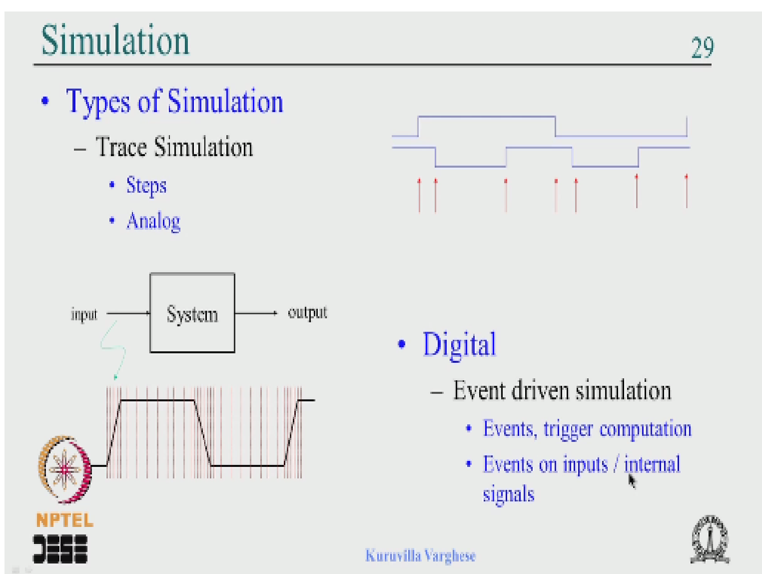


And, we have also seen if you properly define the signal, you can get a concise code in this case we have define internal signal as a vector which allows 1 to rewrite the earlier instantiation in a kind of a in a very symmetric way. So, a3 b3 int of 3 comes and then one proceed to write a generate loop which captures this four statement okay. So, i in 0 to 3 generate and this port map a of i, b of i, int of i and this not a loop in a conventional sense.

It is it is literally replicated for times okay. So, whenever there is cemetery you think of defining the signal properly writing the instantiation properly. So, that this can worked out, it is not that you need to write this way and then rewrite this way, but then you put it down then you see the trend and you can write it and I you can a kind of repel order as an example and this can work not only for the component instantiation.

You can write concurrent statements like if you have a kind of full adder. You can write the equation for the full adder and that can be in a generate loop that will be replicated that possible and whenever you instantiate the component a particular output is not use you have to say open and do not leave any inputs hanging, So unused inputs should be tight to the appropriate level for the active kind of to make the input active. In the case of AND gate it has to be 1and OR gate it is 0

**(Refer Slide Time: 06:18)**

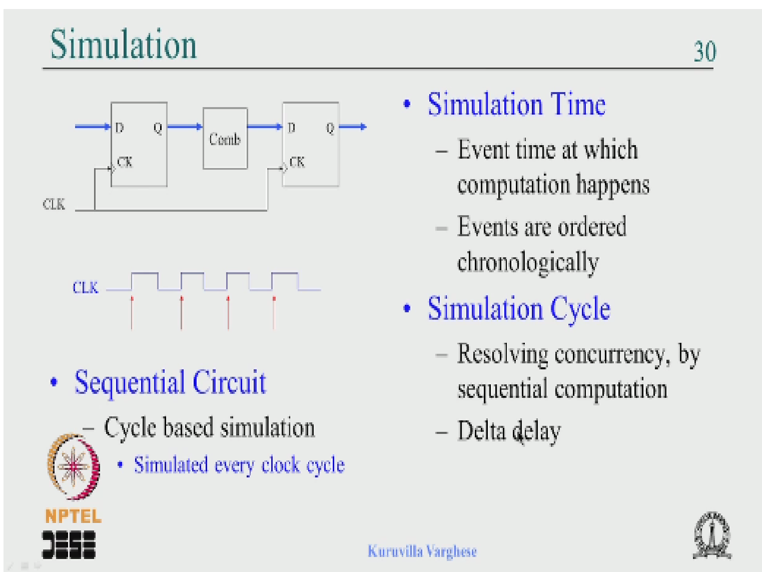


And, we have looked at different t types of simulation I am assuming that you would have done some kind of spies simulation in that case this is the system you have simulating you give an input and you get the output. In spies you have wave form and being an analogue signal it is divided into intervals. And, for each interval the computation is done and wherever there is more detail required there will be no kind of narrow intervals and nothing can be ignored.

Actually, the ignore the interval the better the accuracy of course depends on the system time constant and so on. But, and also the input signal it depends on both and but in digital scenario you have say 2 signals like there no point in dividing the whole range in to minute intervals as small intervals. Because, you know the a signal changes here the it remains everything remains steady till the next change.

So, we need to the simulated need to look do the computation only when the signal change and that is called event driven simulation, a new event will trigger the computation. Event can be on inputs or internal signal within the circuit.

**(Refer Slide Time: 07:45)**



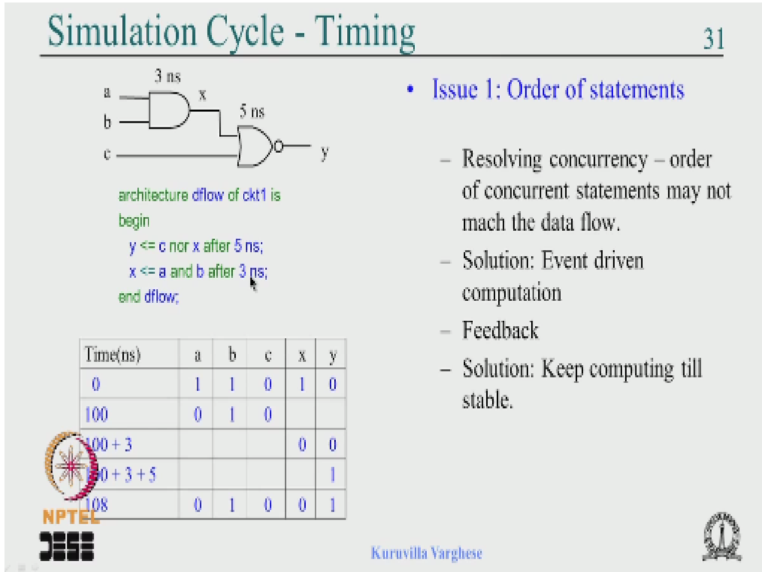

And, in specific when you have a sequential circuit this is updated every clock cycle. So in principle you can do the simulation the computation on the active clock edge which will definitely kind of ignore some details But, this very fast way of simulating the sequential circuit and we have also discuss the notion of simulation time it is not the real time is not the time the simulated takes to do the computation it is a that the time at which the event happens.

And as I said normally events are ordered very sequentially suppose there was an event at the input at 100 nanosecond, 110 nanosecond, 120 nanosecond then the simulation start with 100 nanosecond completion and then moves to the next simulation time and so on okay. And in between for computation at the particular simulation time might take any duration of time that is not the consent as for as the terminology is concern

But definitely one would like to speed up the simulation so the algorithms are important the cad algorithm used for simulation is very important and also you should know that events are ordered in a sequence suppose there was an event at 100 nanosecond and 110 nanosecond. The computation of the event at 100 nanosecond changes an internal signal at102 nanosecond then that event need to be inserted between 110 nanosecond properly.

So, as the computation happens new event occurs because of the computation that is appropriately ordered properly in the time sequence as so that one by one it is handled that is how the simulation proceeds and you can work out you can think what happens in real time work yourself out and convince yourself and you know grasp that concept and simulation cycle is basically resolving the concurrency by sequential computation. And there is also a terminology called Delta delay we will see that.

**(Refer Slide Time: 10:12)**




And, this is how the last part of the lecture the last class we have looked at a case of a timing simulation and a, b is producing an x through an AND gate x and c is producing y through XNOR gate but issue was that this is written in the opposite direction that the data flow is from here to here. And if the simulator handle this from top to bottom y will get the old value because x is not yet updated and x will get the new value.

So if goes from top to bottom and as I said there could be multiple statement in any order. So systematically handling it is that whenever there is a event on a particular signal the simulator looks at the right hand side of the assignment and you can view this as something like a sensitivity less in a process. So if in an event happens on a x is computed and x is assigned after say if it is a is 100 nanosecond x will get an assignment after at 103 nanosecond.

So, at 100 nanosecond, if all the computations are over this simulation time will move to the next event that is 103 nanosecond. So look for x on right hand side then x comes here, so the y is computed and assigned after 1 not at 108 nanosecond. Since there is nothing nod at computed 103 the simulation times moves to 108 nanosecond at the point there is nothing compute because y does not come on the right hand side of any assignment.

Everything becomes stable that is how it proceeds, so it proceeds into two steps here and that is what shown in the table at 0 nanosecond the a, b was 1,1. So x was 1c was 0 so y was 0. But then a goes to 0 at 103 at 100 nanosecond and that require computation which gets an assignment on a x at 103 nanosecond change which trigger computation on y. That assignment happens at 108 nanosecond and that stabilizes the business okay.

So, that is how the simulation cycle happens now basically it is resolving the concurrency from the concurrent statement in may be any order may not match the flow of the data it is solution is event driven computation look for event at the right hand side and keep computing and if there is a feedback. You might ask a question what happens if there is a feedback no issue you keep computing.

If you are lucky it will be or the simulator is lucky it will stabilize otherwise it will gone there is no way simulator can do anything suppose you can think of an inverter which output is connected to the input at least in simulation it can be done and it keeps on if you give a delay of 5 nanosecond generate a clock of period 10 nanosecond. You know it in but that not way to synthesise kind of oscillator.

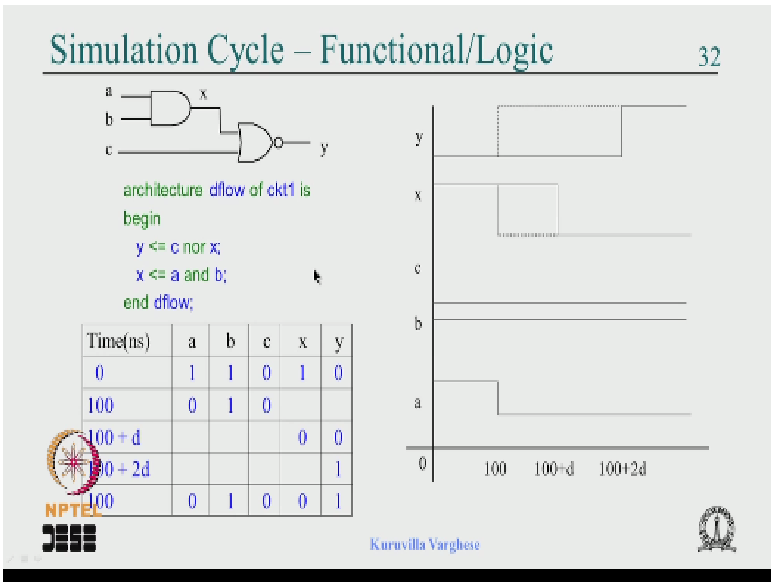
But, you can if for simulation you can make a clock by you know connecting the inverter output the inverter input with the certain delay that will generate a clock way form and we use that for a test benches and so on. So, let us move on as I said so there could be oscillation but if there is oscillation it keeps happening and simulator does not know whether it is intended or not intended because when you generate a clock it is intended.

If there is a mistake it is not intended so simulate a as no way to know whether it is the right one or the wrong one. So let us move at this is the timing simulation, but let us come to the today's part which is the logic simulation or the functional simulation. The circuit is same a b through an AND gate produces x. X in c produces y through an x XOR gate okay. But, you see here and the y is written first this is written first x is written later.

As I said if it if you give to synthesis tool absolutely no problem will generate the circuit because whichever order you write does not matter. Because you say y c NOR x x is a and b. So that is created but for simulator though order is not matching and we said there no issue look at the right hand side and compute NOR like if there is an event on a 100 nanosecond look at the right hand side compute.

But, in this case the earlier case that delay close help us to keep the sequence like in this case the 100 there was an event on a at 100 then x change in response that at 103 nanosecond. So and because of change in x the y change at 108 nanosecond. So there was this delay allowed us to drive the sequence properly. But in this case there is no delay okay.

**(Refer Slide Time: 15:52)**




So, a changes at 100 nanosecond but x is also changing at 100 nanosecond. And, if you look at this y is also changing at 100 nanosecond and that you know if you write your writing the software for the simulator there is an issue because a changes at 100 nanosecond. Now simulation time is 100 nanosecond then x is change at 100 nanosecond and simulation a compact 100 nanosecond you know there is an issue okay

Because, everything is 100 nanosecond. So, the trouble here was that there is no delay. So the solution is just added delay okay at least conceptually the there are many ways to handle it. But like you just added delay some delta delay and we say there is an event at 100 nanosecond x is assigned at x + delta delay. So, now x + delta is greater than you know sorry 100 + delta is greater than.

So, event happens on a at 100 nanosecond x is assigned at 100 + delta. Now 100 + delta is greater than 100 and may be less than even the smallest one 101 nanosecond, SO that is squeezed in between 100 and next event. So next simulation time will be 100 + delta so this is computed and then the y is assigned at 100 + two delta okay. So when everything then you look for the right hand side there is nothing to do it is stabilize at that point delta is made 0.

So, you get correctly the x and y there no issue. So you get 0 delay simulation or the logic simulation, function simulation by adding an extra delay everything is solved so that is what shown here so in the text book you will see a delta delay some people confusing and is it not a big deal it is very simple. So we have at a 0 nanosecond a and b are 1 at 100 nanosecond a is changing from 1 to 0 that trigger a computation on x, x is assigned at 100 + delta.

So I will not written instead of going to equation edit I just put the d but it is the take it as delta 100+ delta and x is moving from 1 to 0. So, the simulation time moves from 100 + 100 to 100 + delta and this is computed which is assigned at a 100 + two delta and the simulation time moves to this there is nothing to compute. So, everything is stable the delta becomes 0 and at 100 nanosecond. You get now a wave form which is 1,1 like the y as 0 then y move to 1 okay.

And, a moves from 1 to 0everything happens at 100 nanosecond with no delay neatly so that is what is shown here you know in the picture a is going from 1 to 0 at 100 nanosecond and 100, 100 + delta x is going from 1 to 0, 100 + two delta y is going from 0 to 1 and ultimately delta is made o. So you can see a goes at 100 nanosecond and y goes from 0 to 1 at the same time now there is 1 thing you have to keep in mind delta cycle is

You know use by the simulator to kind of keep as the proper sequence in computation okay. So you do not have to worry so you just the write statement with simulators way of resolving the sequence by adding the delta delay okay. Delta cycle it is not too much relevant in the synthesis when you write a code I have seen students get got up in this delta cycle. Okay. Now delta cycle is not a magic by which you can create different behaviour

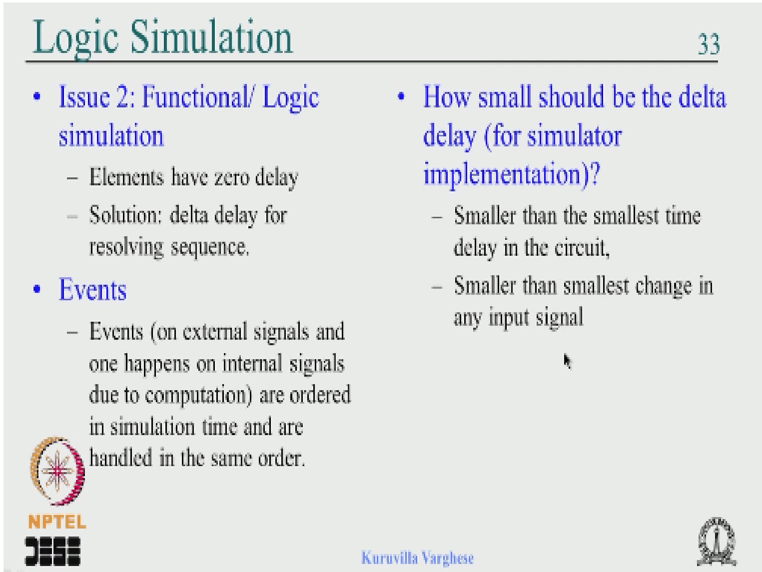
Different the circuits and so on so you do not worry too much about the delta cycle and say this assignment happens after the delta cycle what circuit you get nothing you know you write the statement which you are going to see what this statement means as for synthesis consent you strict to that rule do not worry about how the simulator you know resolve the concurrency by data cycle.

Because, if you get cot it as some implication in the synthesis definitely we will discuss that but is not a I mean when you write a code the last thing you should be worried about should be something about delta cycle that is for the simulator to resolve the concurrency and it is a concept. I am in the simulated need not stick to this to keep track of the sequence. Basically you have to keep the sequence of computation in the simulation time sequence.

So, this is one way of resolving there could be a other way of resolving at So, please do not worry when you write a code think about the circuit think about what a concurrent statement mean that we are going to handle in the coming lectures I will teach you very clearly how each statement means what it means as for as combination circuit concern as for as sequence circuit is concern.

So, do not worry this is for understanding how the simulator resolved the concurrency okay. So, let us move on. So an intelligent student will ask s what happens when there is a feedback, So let us look at the feedback scenario just for clarity there is nothing new per say.

**(Refer Slide Time: 22:11)**



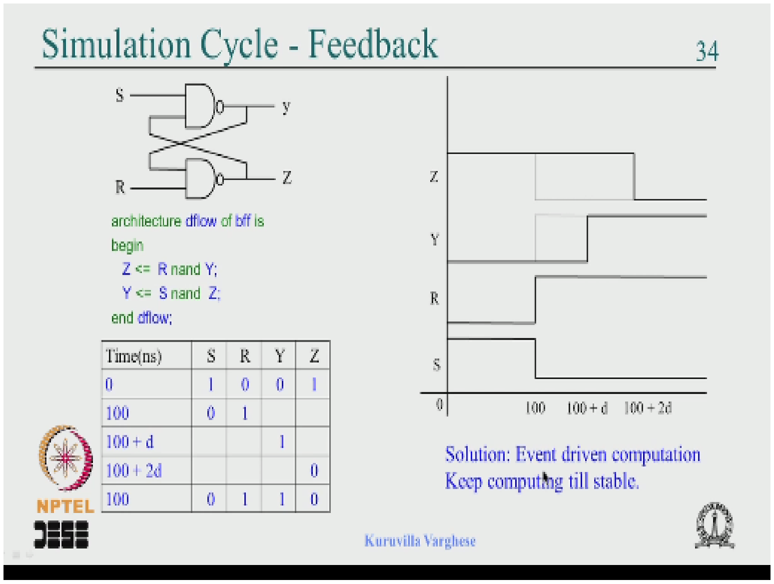

But, then let us look at it okay. Now may be there is one question before that feedback thing like really suppose we are CAD tool is trying to implement this delta delay how small should be the this delta delay. Because, ultimately if you are trying to resolve this concurrency. But, delta delay you have to give some value how smaller should be. So, definitely it has to be smaller than any gate delay.

Because, suppose you are giving a signal is changed at 100 nanosecond and the output of that gate changes at 101 nanosecond and you cannot have a delta delay which is 2 nanosecond because then the when you order this the kind of event it found be proper. So, it has to be smaller than any gate delay any delay in the real circuit if you are walk in with the SPGA which as a 1 nanosecond as a basic delay.

The delta delay has to be specified or internally by a simulators more than that also you have specifying the input signal to the simulator which has some resolution may be you have specifying that a changes at 100 nanosecond, then 101 nanosecond and so on. The minimal kind of resolution you specify the input is at order of 1 nanosecond then definitely the delta also has to be smaller than that otherwise again.

When the events are ordered it will ordering want be prospered because delta is 2 nanosecond then which is supposed to come before 101 nanosecond will not happen that has to be smaller than the smallest change in the input signal. So that has to be kept in mind.

**(Refer Slide Time: 24:10)**




So, let us look at the case with the feedback scenario I have shown a very simple circuit a cross couple NAND gate, please have a look at it. So you have a cross couple NAND latch and as you know that for NAND latch the input is active low okay. And when suppose when you want to set this then you make it 0 and this one this 0 will forced output be 1and this 1 and 1 will make this 0.

And it comes reinforces then it can go to the memory node 1, 1 it will be memory mode. If you force both 0 to 1, 1 there is raise. So, it is kind of the when it comes back to memory it is undefined. So you do not do that so the set is 01 reset is 1,0 and memory is 1,1 okay. So let us look at this so here we are writing z as R nand y, and y as S nand Z. So, here there is since there is a feedback there no order you know there is anything everything is messy.

Whether you write y first z first since there is a cross coupling everything is as bad as anything, so let us look at it okay. So, assume that time 0 the it is in reason the latch is reset that means S is 1 R is 0. Because, R is active low. So, y is 0 this is 0 and Z is 1, So let us move from reset to set okay, So you know that this was 1,0 now it is move to 01, So both S and R changes at 100 nanosecond okay.

Now two signals are changing you know so there is good it has some complexity. So, but you see under right hand side S comes here R comes here, So simulator has to compute both Z and Y at 100 nanosecond. But, you see let us take the Z that R is changing at 100 nanosecond Z is nothing but R nand Y. So, it is 1nand 0 okay which is nothing but 1, So Z is 1 so at for this 100 nanosecond change Z does not change that was already warned.

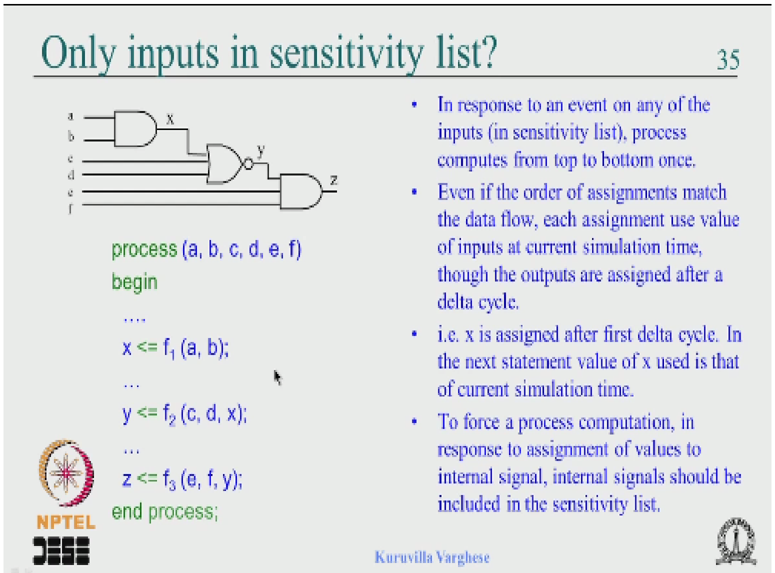
So, this makes no effect but if you look at the Sn Z S is 0 NAND Z is 0 NAND 1 which is 1, because any 0 is1 out output is1 y was 0. So there is change in Y. So the Y is assigned at 100 + delta. But now this simulator computed both 100 nanosecond events simulation times move to 100 + delta, So there is an event on Y. So, this need to be computed which is nothing but R NAND Y.

So, that is nothing but 1 NAND 1 which is 0, So Z is change from 1 to 0 which is assigned at 100 + two delta. Now since 100 + delta computation is over move to 100 + two delta Z is going from 1 to 0 again Y need to be computed So that is nothing but S NAND Z, So 0 NAND 0 which is a 1 So, Y was already 1 there is no need to compute everything is stabilize now the delta can be made 0 then you get 100, So S goes from 1 to 0 R goes from 0 to 1 and Y goes from 0 to 1.

And Z goes from 1 to 0 that is shown in the picture this was 1 0 which is going from 1 0 to 0 1 at 100 nanosecond 100 + delta Y goes to 1 100 + two delta Z goes to 0 ultimately when you make it 0 the Y goes to 0 to 1 Z goes to 1 to 0 at 100 nanosecond. So, it was in kind of reset mode it goes to set mode at the same time so there no issue actually absolutely no issue whenever there is an feedback two things can happen.

One is dot it stabilizes it converges or it oscillates in either case this as no issue with feedback, So this kind of delta cycle concept handle the feedback also feedback scenario also that is what is shown here. So, solution is to keep computing till everything becomes stable if at all otherwise it goes one and one okay.

**(Refer Slide Time: 29:31)**




Now, so that is what we have seen now very quick recapture we have seen how the kind of like event driven simulation, event driven computation resolve the concurrency. In the case of no delay we add a conceptual very small delay, delta delay which keep tracks of the sequence of computation and there is a feedback with no issue depending on the events the computation continuous two cases it oscillates or it becomes stable okay.

So, now let us come back to the process and when I am mention the process I said that to implement a circuit okay. So, look at the slide we have this scenario here a b is driving an AND gate And gate output is x along with c d an XNOR gate drives y and along with e and f it drives Z through an AND gate okay, now I said that to write a process for like assume that this is a block at to write a process for you put all the internal signals sorry.

All the input signal in the sensitivity less and we write the statement okay now we could written in single statement but to illustrate this idea I have assigned I will written as statement for x like using some if case or something like that or you just write sequential assignment. So, x is written as function of a b essentially a and b y is functional c b x, XNOR you know and Z is a function of e f and Y. Now you see that we are writing it in the proper order.

But you look at it if an event happens on a see how the simulator not the we are talking about simulation there is an event on a which is in the sensitivity less so the rule is that goes from top to bottom once okay. Now you know that there is an event on a. So, this x is computed but x is assigned suppose event happens at 100 nanosecond x get the value at 100 + delta nanosecond, but when it comes to Y the Value of x used is that of 100 nanosecond.

Not 100 + delta nanosecond, because the current simulation time is 100 nanosecond. So, the Y is computed using the old value of X. Similarly Z is computed using the old value of Y doesn't matter in as I said we have written it in the proper order but you see that X get the correct value Y N Z is doesn't get it all. Now that means there was an event on X this processes not responded to it.

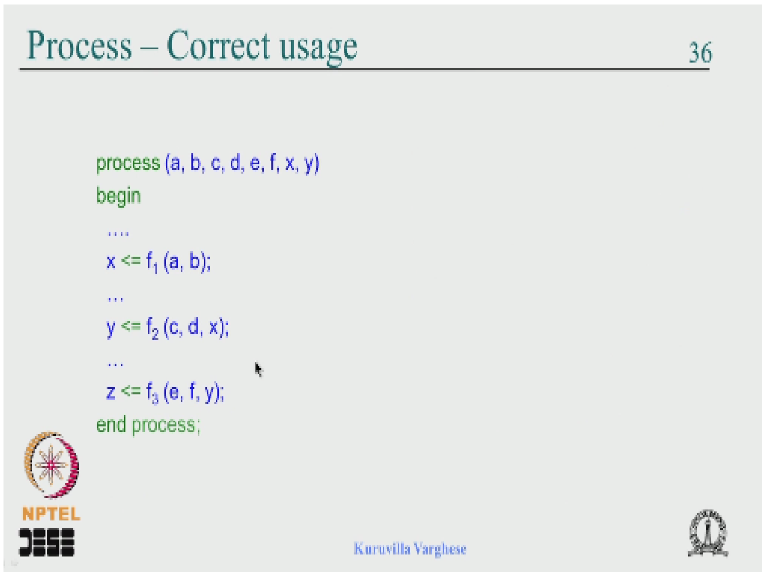
So, we know how to make a process respond do in event on a particular signal that has to be sensitivity less, So the solution is to write X and Y the internal signal in the sensitivity less. Then what happens in the first time and event happens on a it goes from top to bottom there is a change in x which is assigned at 100 + delta once it is completed there nothing node to do at 100 nanosecond. So the simulation time goes to 100 + delta.

So, if x is specify this is computed again and Y gets updated. Now, in the next ration there is an 100 + two delta event and that if Y is a trigger and Z is updated so the rule is that when you have internal signal you need not write I mean you could write you could think of writing Z as a function of abcd are directly. You know you could you could as for write Z is nothing but a and b x or c and d like the proper of bracket you know and Z f you can write single statement.

Then you no internal signals need not this walks but when whenever there is internal signal for proper simulation happen you have to put internal signal in the sensitivity less of the process that is what I said please keep that in mind. So earlier stated that you just write the input signal in the sensitivity less that is not enough you have to write the internal signal but I waited for this delta cycle business to be handle then if I state that that become clear that is why this is handle later.

So, basically this is what I am stating in response to an event on any of the inputs the process compute from top to bottom. Even if the order of the statement match the flow of the data the all the values used is of the current simulation time this newly assign value does not affect the computation in the subsequent Sap So you have to put them in sensitivity less that is what stated here.

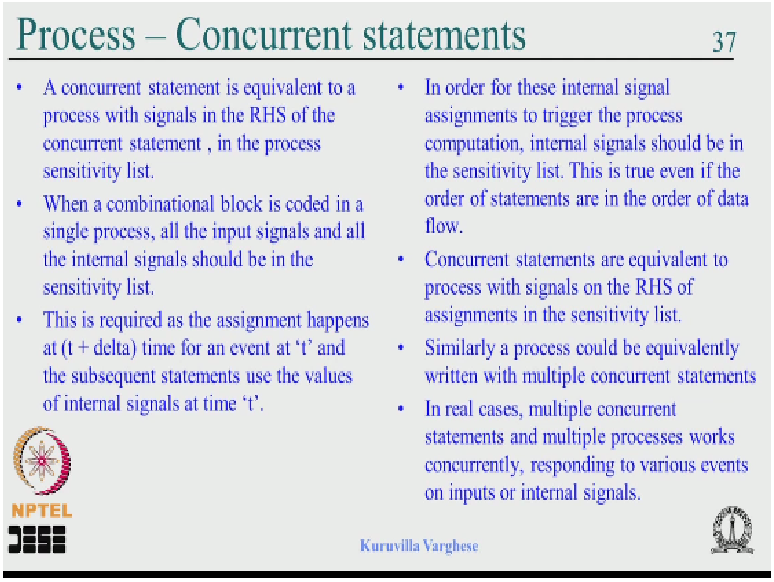
**(Refer Slide Time: 35:16)**




So, that correct usage is at in this case you have to put a, b, c, d, e, f and x and y in the sensitivity less and then everything works it rates three times and everything is updated properly. Now you see that there is a kind of equivalence as for as simulator is concern, the processes and concurrent statement. Because, whenever there is an event on the sensitivity less signals in the sensitivity less the process computed.

We have a concurrent statement in the earlier case, whenever there is an event on right hand side of the assignment this happens okay. So basically you can think of you take this concurrent statement you can write it is as a process with the R and Y in the sensitivity less and you write a sequential assignment. Similarly, it is possible that when you have a process you can write multiple concurrent statement equivalent multiple concurrent statement to. So, there is an equivalence between the processes and the concurrent statement that is what I want do to say

**(Refer Slide Time: 36:34)**

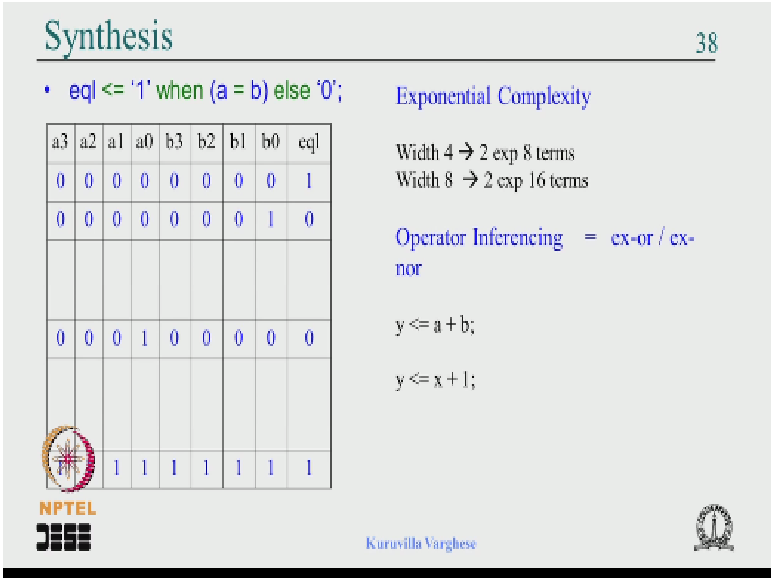



A concurrent statement is equivalent to a process with signal in the right hand side of the concurrent statement in sensitivity less. So, you take concurrent statement you can write equivalent processes like all the this is stating about the sensitivity less internal signal in the sensitivity less. Even if the order is kind of MAT and we could like if you have a process you can break into equivalent concurrent statement.

And you should remember this fact in real life you may have say for processes and 10 concurrent statement everything is concurrent each other like processor are sequential body. But all processes all concurrent statements are concurrent if a particular event happens wherever that signal is on the right hand side of assignment and sensitivity list of a process that gets computed at that same simulation time.

So, that you should remember that processes, and concurrent statement are concurrent to each other. And, depending on the event everything is computed at the same simulation time lot of computation like if event happens at 100 nanosecond simulator may have to computes some 10 processes and 15 concurrent statement depending on the complexity of the description or complexity of the circuit. So that should be kept in mind.

**(Refer Slide Time: 38:13)**




So, that was a about synthesis sorry simulation now let us come to briefly very briefly we will take it later when we discuss concurrent statement look at the statement this is what we have written for equality comparator which say that equal get 1 when a is equals to b else 0. Now you think how this can be synthesis okay think of a mechanism by which anything you write any at least for a now think of a combination circuit.

Don't worry about the sequential circuit think of a combination don't worry how the what is a VHDL code for a flip flop and so on that we will handle but for a combination circuit. You write some statement what is the like sure way of synthesising like we have this statement like give come out the scheme which will work for any statement okay, So let us come back to the slide think for a while you know.

I will give you a little time just think how you know efficiently how a sure way of synthesising this please think about it for a moment. So, I can give a hint like you think of how did you design a combination circuit like in the basic course you know you have some description like somebody say in words this is the scenario and how do you generate how do you, design a combination circuit.

And you know that from the description verbal description you would have written a truth table. So, like 100% sure way of synthesising from a description to write a truth table. So, in this case at least in our case we had to input signal each of which is for bit like we have a of 3 to a of 0 b of 3 to b of 0 these are the two inputs equal is a 1 bit. So, make a truth table input verses output. So, we have a3 a2 a1 a0 b3 b2 b1 b0, this is a input part.

The equal is the output part and there are so this is a 8 bit. So, you have you can imagine there are 2raise to 8 rows. And we know that when wherever these a pattern and b pattern match we are write 1 rest all places it is made 0 okay. And then as you know the tool should synthesis tool should make an equation, wherever there is pick them in them minimise it in so on okay looks very nice it works for all the cases.

But, then you can imagine what happens I say that instead of a3 downed to 0 I suppose I say a15 downed to 0 okay. So, the moment I say that 15 downed to 0, then you have a15 to a0 b15 to b0. There are 32 values and the number of rows in this truth table will be 2raise 32 which is for gig and you can imagine if synthesis tool is timed to make such a truth table in the memory and may be you have a 2 gigabytes physical memory it has use.

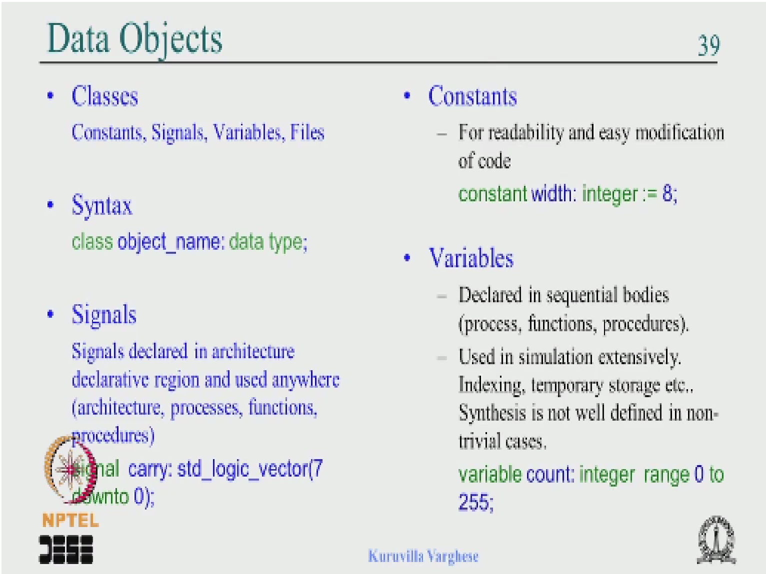
The versatile memory properly the no versatile memory then the system can crash. So I may be I execrating. But, you see that a very simple solution has doesn't work here use of the lot of memory. And, so because of exponential complexity of the digital case, whether means them it goes exponentially complex. So, most synthesis tools does not built to truth tables okay.

So, it is as you would have guess when the synthesis tool this see this operator equal operator. This operator is written in a proper way to reflect this circuit of XNOR gate or XOR gate okay. Now so when the internally the operator is implemented using the XOR gate, So the it is called Operator Inferencing synthesis tool try to look at the operator and operators are written with exclusive OR gate already.

So, that is how it is synthesis So similarly somebody write a+b it is not at a truth table of the adder is written and you end up with the carry look at adder or something like that at the + is implemented in a simple way as a ripple adder similarly you have a you write x +1. Then this is replaces by an incremental. So, basically the synthesis works on you know the standard structures of the operator like it heavily depends on to in a catch.

The operator which is used and equivalent circuit which is written in the library for that particular operator. That is how the synthesis works, so that this kind of impossible situation is not reach but at times when this synthesis tool can to this and where things can go wrong. When it cannot make sense out of what you have written or you cannot guess the correct operator then the things can go wrong okay. So that is how the Synthes is happen and assume more long. When we come to the statements we will see how it happens.

**(Refer Slide Time: 44:38)**




So, that is what I have talked about the synthesis, So please keep that in mind then we come to this part which is little bit boring but for completion sake I will run through it and the VHDL is very properly define the syntax is very what is say very precise. And, we have 4 classes of 4 classes, Constants, Signals, Variables and files.

And, you know there in a normal language of Constants, Variables and Files but we have being hardware we have signals which we extensively use. And, syntax for all classes are you write what is the class then give a name of the object and incense of a class an object. So, a game name for an object and what is the date type. So, you might sake constant with integer and so on okay, so that is a the syntax.

And, if you look at the c code may be the classes not specified in some cases that is in the case of variable. But here it is especially stated, So let us look at the signal class signals are basically declared in the architecture declaration region. And, it can be used in every where it can be used in architecture statement region processes, functions and procedure the syntax is signal a name and the data types.

So, here we I write signal, carry, standard logic vector 7 down to 0**.** Similarly, constant are is for readability and easy modification. So, we can define a keyword is constant in name width and then what is the data type integer, Since integer can take any value we have to kind of we are same what is value here the 8. So, the wherever the width is use suppose you have defining a counter.

Then you can define the counter output in terms of the width you can say counter output is width-1 down to 0 okay. So, you can come and change you know it is an 8 bit counter. Now, you just change this to kind of 16 then you get a 16 bit counter that is a use of it, So the variables cannot be declared in architecture declaration region it can be declared only in the sequential body like process, function and procedure.

And, this is used in simulation extensively it is use for indexing temporary storage and all that for synthesis it is not very well defined in non-trivial cases okay. Now, this is it not that you know some time people say make statement like variables are not synthesisable. It is not true like a you take a symbol you know combination circuit and you try to replace this signal with variable everything works fine.

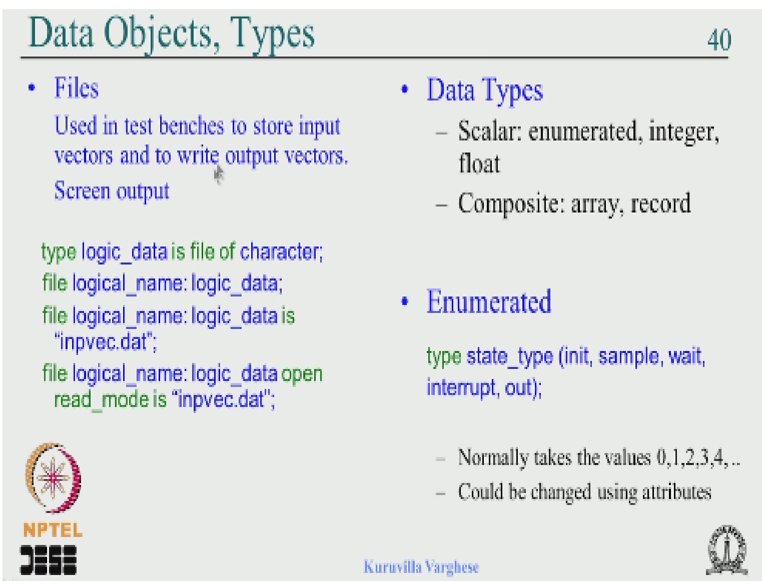
What I mean in the non-trivial case is that suppose you take a algorithm standard algorithm which is written in C code using variable and try to bring that into VHDL and try to implemented. You may not get an equivalent circuit which does that business that is the meaning of that is what is suppose to be the meaning of when somebody say the variables are not synthesisable okay.

Whether, the person whose says nose are not I do not know but these are very blanket statement. Which sometime people make and the there nothing kind of philosophical about it that the essentially that is the meaning you take an algorithm which is specified in terms of the variable and you write that rewrite that in VHDL using the variable and you hope to get a circuit to implement that function you may not get it at meaning of it.

We will see again synthesis using variables as we go a long. So, this syntax of the variable declaration is variable that is the keyword count and integer is an internally define integer. Integer because integer can take lot of values we have saying range 0 to 255 because our aim is to synthesis a circuit out of this. And since this circuit should have boss of fix it. We have to say what is the range you cannot like when you say 0-55.

It is an unsigned boss which is an 8 bit boss. So, always a you have to specify the range of the variable when you specify So that this synthesis tool can convert this in your bus of fix it and let us come to the files.

**(Refer Slide Time: 50:00)**




Files are used in test benches to store input vector and also to write output vectors. We will see the syntax when we go the test bench, but may be quickly we will see how that is use say normally you define a data type called type some name is a file of character a short form file of character. So, wherever use logic data its says that it is it is a file with characters inside then the weight use the file is give the keyword file a logical file name or you can say it is a file handler.

And you say this logic handler and you say this logic data that means it is a file of character. Then you can say if you want you can say if file logical name logic data is like in codes some file name this is a physical file name in the current working directory and known that is the meaning of it or you can say file this file handler

The logic data is data type. Open remodel is physical line file name that means that you opened is file in the read mode. Only for reading and what is physical file name you could when you just write the file name in codes it means added comes from the current working directory. But, if you have to as specific location you have to give the full path. So, that is how the file is use and when we discuss the test bench**.**

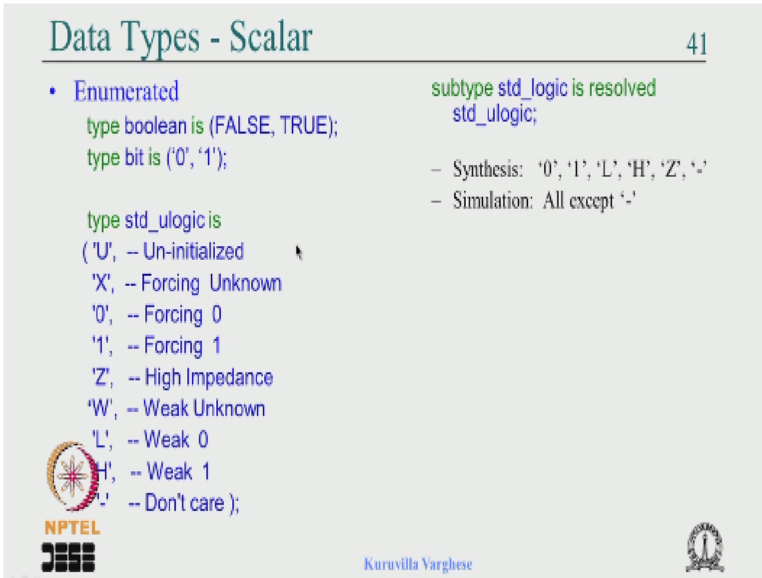
We will see in detail. So, that you know talked about the 4 classes basically the signals constants, variables and files. So, let us look at the data type there are two types of data type one is Scalar and the second is Composite and the Scalar you have 3 types one is numerator data type, and one is integer, one is float okay, the real numbers. The composite you have array this is very useful record which is not as used as array.

So Scalar and Composite Scalar you have numerator and integer and float. In composite you have array and record. So, let us look at the numerator data type say normally for state machine state diagram. We can use this kind of thing you can say type any time you say type it is a new data type. Some name and in the bracket you give the numeration like you numerate the states like in a comma sample comma ait comma so on okay.

These are the states of the state diagram but internally normally when you say this is for convenience internally takes a value 0, 1, 2, 3 and 4. And, there are5 states are it will be definitely represented in 2, 3 beds okay that you should not forget internally converted to 3 beds and normally the assignment is sequentially like 0, 1, 2, 3, 4 for some reason you want to change the assignment to some other order.

This can be change using the VHDL attributes okay. We will see that how to do that may be when we take the state machine coding the VHDL coding of state machine handle may be some kind of assignment will solve some issue in the state machine implementation, So at that point we will how to use attribute to change the this particular assignments state assignment.

**(Refer Slide Time: 54:13)**




So, let us move on may be discuss and wind it up. So, the numerator we have already seen the bit and the standard logic and Boolean is also there. So this is define in the standard library. So, type Boolean is false or true. So you could have something define as Boolean which takes value false or true. You can have the bit is to defined as type bit is you know 0 in codes comma 1 in codes. So the bit takes only 0 and 1.

And, this is what we have used the types standard U logic okay. Now standard logic is little different from standard U logic, So this is an numerator type which takes you know open bracket you this is the comment comma x comma 0 comma 1 Z W L H and dash which is do not care okay. So, these are the 9 values the standard U logic can take and let us pick up one by one. So, U means un-initialize which has no use in synthesis.

But, maybe simulate at beginning of the simulation if a signal not initialize then you can be shown as you 0 it is a 0 you know 1 is 1 you know and similarly L and H are also 0 and 1. But, this is forcing this is weak okay essentially it means forcing means it has a high current drive. So, you can imagine and output driven through an n mass or a p mass can be thought of as a low resistance drive.

So, this represent 0 and 1. But, for some reason you have a pull up or a pull down then you are driving a signal with the resistor high resistor then this can be treated as weak 0 and 1. And, x is when something is unknown suppose you have a line to which two output through a Tri-State gate is suppose in simulation. Some of it happen that both are driving and 1is driving, 0 1 is driving 1, then the resultant value is x which is unknown.

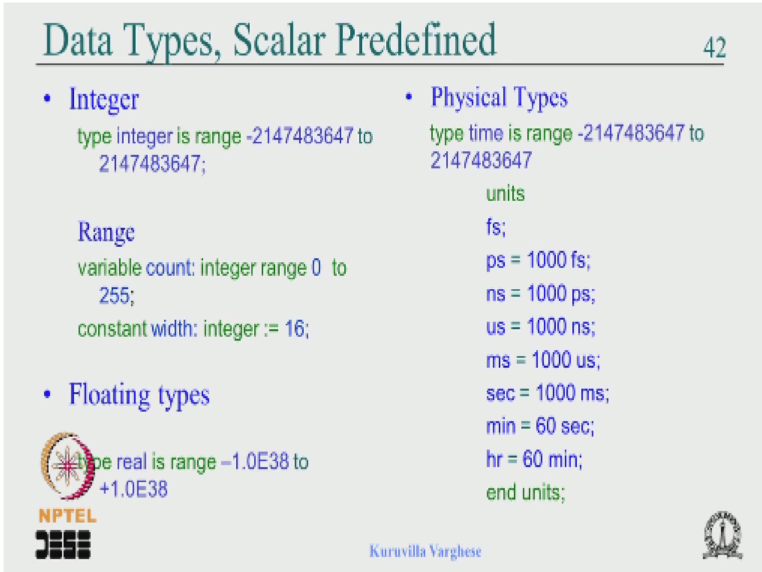
So, that is 1 so in simulation can show x is unknown then you can kind of figure out what is going wrong and W is when L and H clashes. X is when 0 and 1 happens. So, W is a weak equivalent of the x and z is you know the high impedance and dash is don't care which is very useful in synthesis for optimisation. So, in principle synthesis use 0 1 L H Z.

And, don't care simulation can use everything except don't care like if you assign something don't care it doesn't make sense simulation you can be shown up like that. But, there no particular meaning in it. But, you should know that these are just the values as standard U logic takes but that is in mean if you specify something else L. You will get a pull up resistance or something like that.

You would not get that. So, this is just a meaning of the standard U logic and depending on how you write like whether you write a L H are 0 or 1. Everything is treated by synthesis tool of 1 and the standard logic. We have use is a sub type of standard U logic which is going through a resolution function called resolve. We will see a as we go head what is that. But, for the time being you can treated as a exactly say only thing is that.

Whenever, there is multiple drive it goes through another function to resolve the resultant value that is the meaning of it. We will see later.

**(Refer Slide Time: 58:21)**




So, I think with this I will wind up for today. So, the last part was the data type you have seen the classes for classes the variables, constants signals and files. Then data types Scalar and the composite in Scalar numerator integer and float and you seen Scalar. So, please go through go back and revise refer to the text book learn it well**.** So, I stop here and thank you and wishing you all the best.