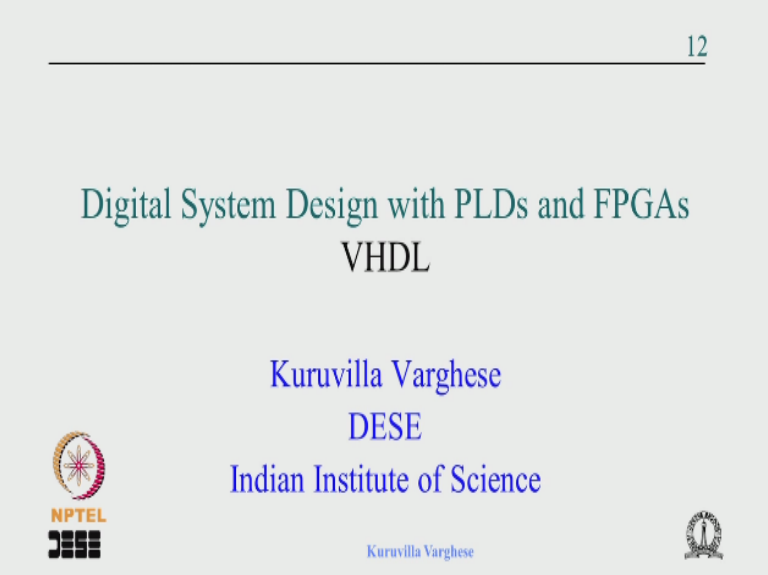
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science - Bangalore

Lecture-11
Structural Model, Simulation

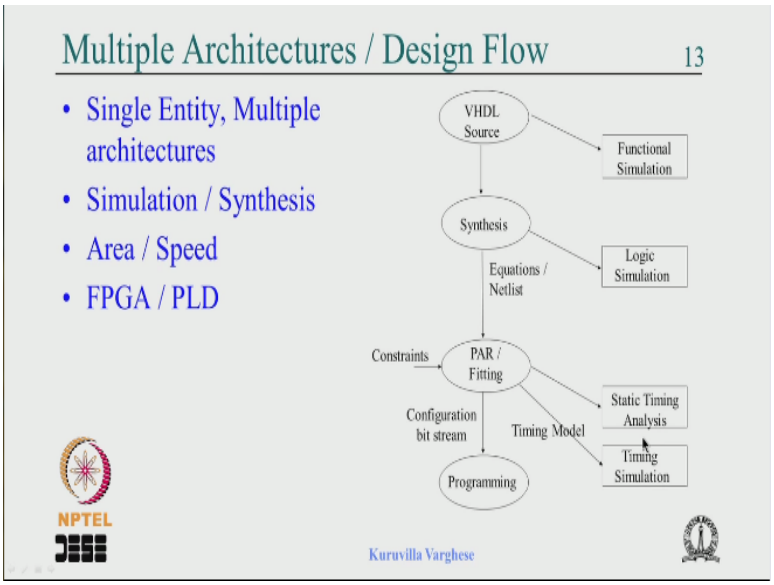
Welcome to this lecture on VHDL in the course digital system design with PLDs and FPGAs.

(Refer Slide Time: 00:27)



The last lecture we have gone through the use of multiple architectures, 2 models of description 1is you know data flow model and behavioural model. So let us look at the slight when you have the multiple architecture for a single entity it can be use.

(Refer Slide Time: 00:52)



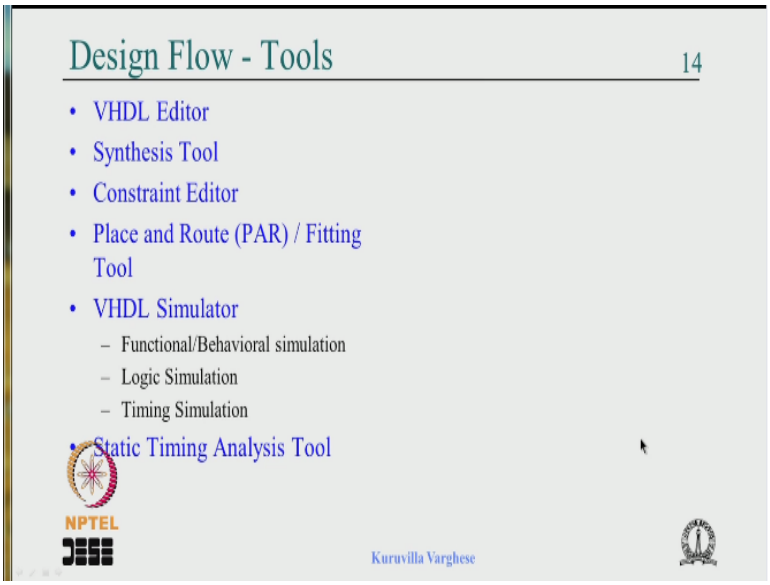
You know you can have a scenario where you write a quit code for simulation or another architecture which is kind of after the simulation you can do it for synthesis. May be you can write one architecture for minimum area and the one for maximum speed, one could be FPGA, one could be PLD or ASIC and so on. And we have seen the design flow start with the VHDL source code which is functionally simulated.

The code is simulated not the circuit to remove the logical syntax errors and it right here. Then it go for synthesis where the logic circuit is created. And you can do a logic simulation if you correctly code if you correctly write the synthesisable code this step may not be necessary. So you can do a logic simulation which is a both the simulations are not delay because it is not map to the device.

Then once you do the synthesis you can go for the process of place and raw out fitting place and raw is called PAR and you give IO constraints and timing constraints do quick static timing analysis depending on the delay of the block and when you trade there and when everything is okay you do timing simulation which might take time for complex you know circuits.

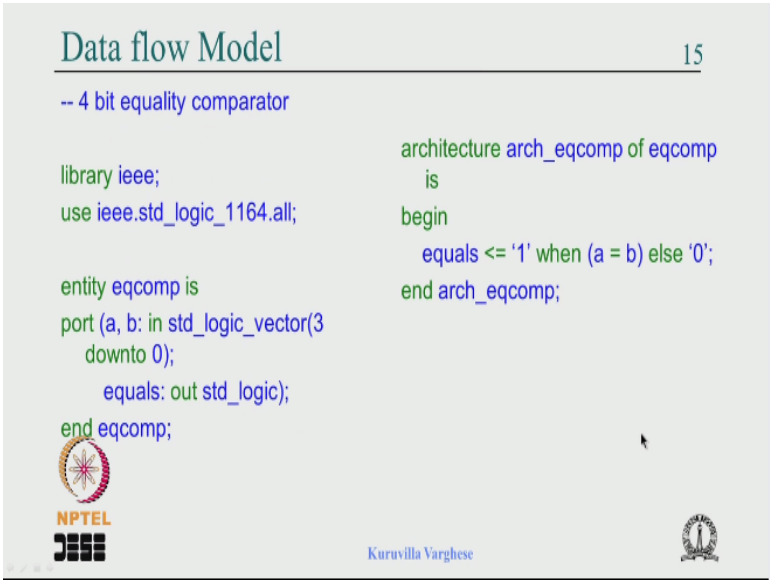
Then everything works fine then you can program the FPGA. So basically you have an editor, you have a synthesis tool, constraint editor, a part tool, a programmer, a static timing analysis tool, and a simulator which does all these, you know functional logic or timing simulation.

(Refer Slide Time: 02:41)



So, these are the kind of the tools we used.

(Refer Slide Time: 02:45)



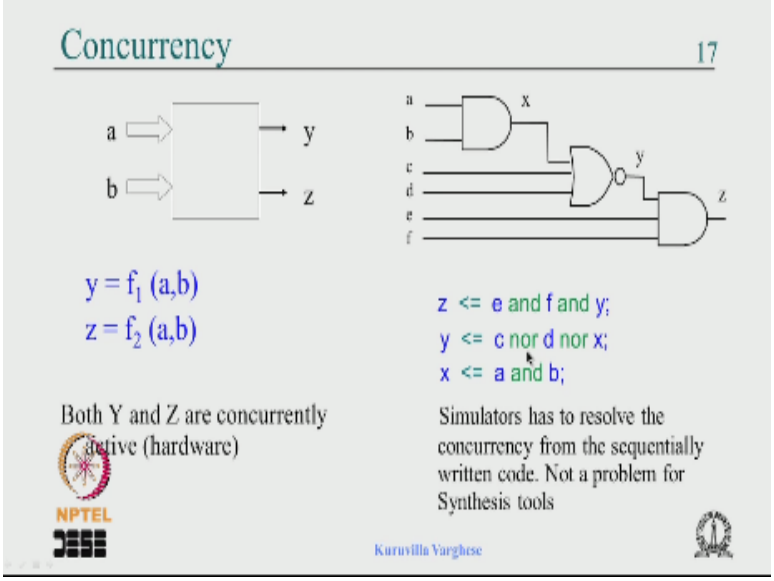
We have seen the data from model, the first code as an example we have seen is a data flow model, where it shows the data flow. This is the input and you know the output get assigned and this use a construct called when else statement which is used in the concurrent body. So this is the concurrent body. Everything in the architecture declare a statement region is concurrent and this is the concurrent statement.

There is another statement called which select which we will see later there is a another way of writing using the logical kind of operators as I said this can be combustion when the data size is

a large and so here even if whether it is 3 or 31 it does not matter. This what so this is very concise you need not worry about the circuit you end up these are very simple thing to synthesize.

So this synthesise 2 will make sure that whatever you written the equivalent circuit depending on the device in an optimal ways implemented

(Refer Slide Time: 03:52)

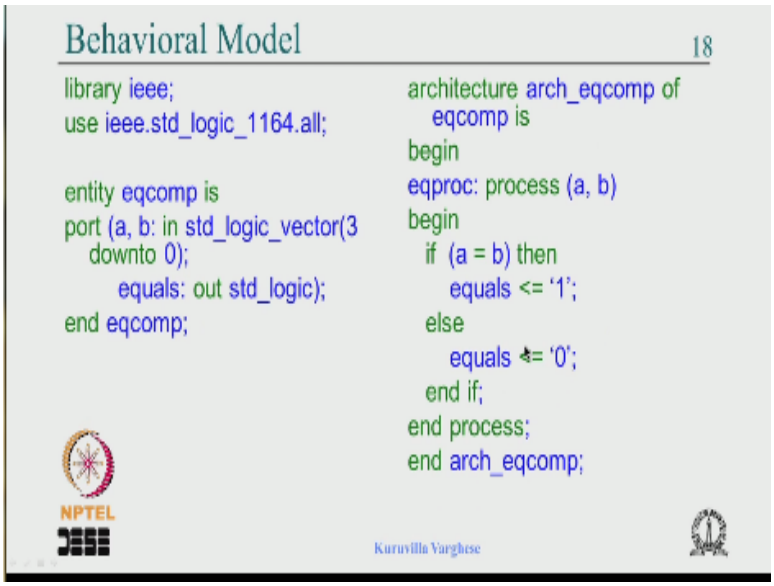


So that is the data from model and we are talked about the concurrency as I said there is an issue like a it is not unlike the sequential languages when you write some statement in a specific order in a sequential language one after the other is executed. But when you simulate basically simulate a VHDL statement sequentially written concurrent statement like this a top to bottom evaluation will not yield correct results.

So this simulate as through result concurrency from the sequential written code, because here it is written in the opposite of the data flow. Data flow is from ab to x, x to y, y to z we have written in the opposite way. So z and y will be wrong X will the correct thing, if you do the top to bottom evaluation those simulators job is to resolve the concurrency from the sequential written code.

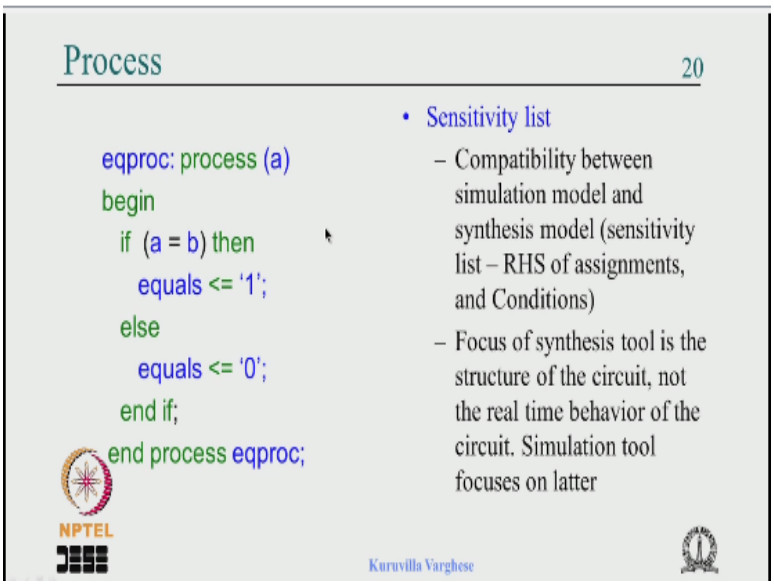
But, is not a big issue for the synthesis tool because a the expressions are there and you know which ever order it is written does not matter you end up with this same circuit. So in this case at least for **m** in this case the simulators as a ad job.

(Refer Slide Time: 05:08)



Then, the synthesis tool we looked at the behavioural Model where we use everything else is same where we use a sequential body called process and process as sensitivity less. We input the input step when anything changes in the input. It computes from top to bottom once and it uses sequential statement like if then case when and all that and before the begin you can declare variables, you can declare constant. You can define procedures of function which ever visible to the body.

(Refer Slide Time: 05:46)

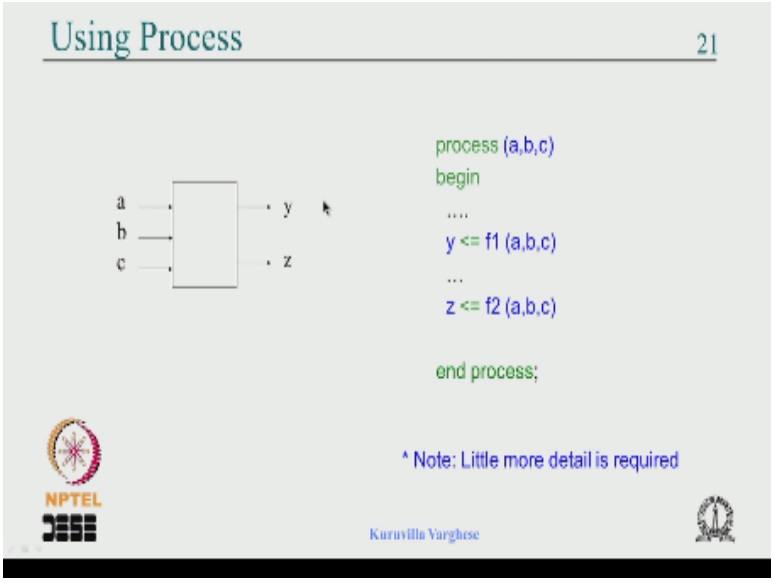


So, and you have a problem if you miss something in the sensitivity less. Because here you see in the statement there are two inputs but if you don't write both inputs then the simulator will give a behaviour which the synthesis should want be able to kind of you know produce because synthesis tool does not bother about real time behaviour it looks at the code and generate the circuit.

Code say if a=b equal to b equals get one, else equals to 0, so just create an equality comparator but the simulator simulates an equality comparator which works only one a changes, if b changes next time a changes it works correctly if b does not change till a changes. Otherwise if the b changes it does not compare you know something like that, so so there is a problem a compatibility problem with the stimulation and synthesis.

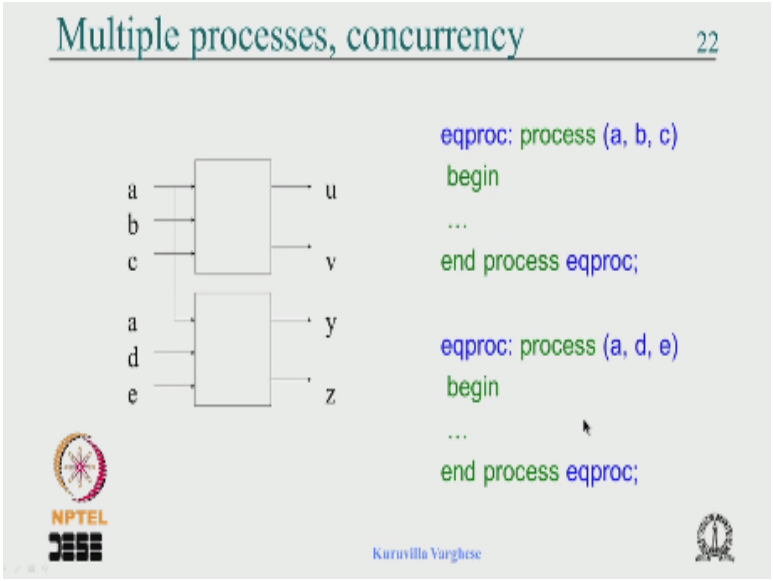
The solution is to write all the input signal in the sensitivity less and some tools warn you and if the tool does not warn you it is a dangerous thing, because you simulate and see some behaviour and this in a complex case, sometime it can be very simple case will be able to identify. But in a complex case it can create an issue, so one need to take care of this particular issue of missing signals in the sensitivity less.

(Refer Slide Time: 07:20)



And, as I said there is nothing great about using processes do not get kind of you know worried about this what processes if you have a block with inputs abc, you write abc in the sensitivity less, you use if k is loop etc., to define the behaviour of this output in terms of abc that is what is required. As I said when I say write the input signal in the sensitivity less little more detail is required may be today's lecture will throughout like on it.

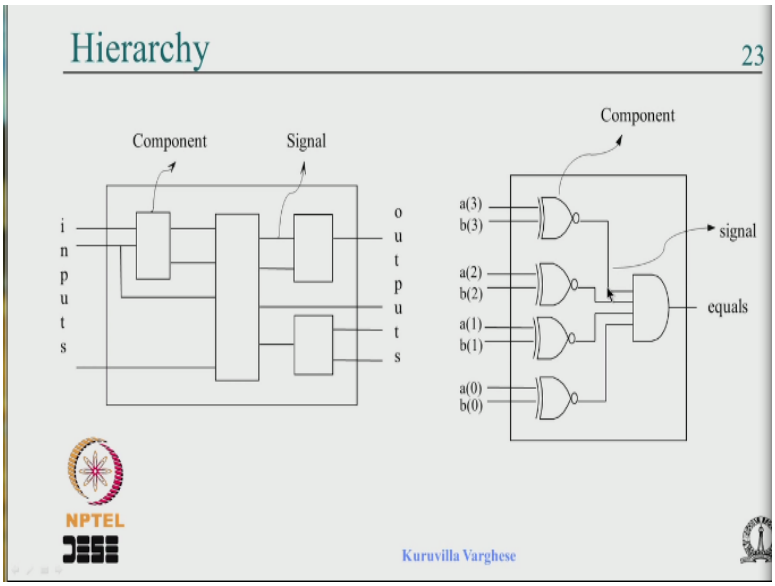
(Refer Slide Time: 07:59)



And we will go ahead and we also said when you have multiple processes both are concurrent say you have two blocks they share an input called a. So you write two processes and if a changes both will be computed for the same event okay is not that the simulate cannot compute

at the same time need not be but if a changes at 100 nanosecond both these computation will happen for that particular 100 nanosecond event that is the meaning of concurrency.

(Refer Slide Time: 08:33)



So, let us go head and let us come to today's part and here we see a model of description called structural model of VHDL code which enables us to do the hierarchical design or the top down design and we have seen already how we have design the CPU top down. Now you know we put a black box CPU say level1 it was interconnection of various components. So, we will see how that can be done in VHDL using the structural code that is the basic idea.

So, when you have a top level entity which is built up of the black boxes from the level 2 or the bottom level components then what is done is that you the top level entity you give a name as usual you define the inputs and outputs okay. Now, the game is that we are now going to describe the behaviour of the circuit. We are going to say this entity the architecture of this entity is composed of the interconnection of various component. That is what is going to be stated in the architecture statement region.

And, for that we need to say what are the components used okay. So, may be you take these 2 components they may be same components okay. You can have a same component instantiated multiple times in an architecture. So, you have to declare in the architecture declaration region before the beacon. The component it is inputs it is outputs and data time.

Exactly like the entity of that component has to be repeated, the idea is that you are going to put another component and you are going to connect using sub signal. And you know that the VHDL is a strict data type checking language. So, if you are connecting an output to an input, if output of the types standard logic, then input also has to be standard logic and this wire or the signal connecting them should be of standard logic.

So, we have to declare the distinct components we are using in the 4 the inter connection in the architecture declaration region. All of them and we also have to you see this signal. This is not part of the input or output. So, we have to declare the signal it is called signal of which is which has no direction, because signal connect some output to input. One or two inputs, one output to one or two inputs normally unless you use tri state gate.

So, the signal does not have a direction, but it has a data type and everything has to match okay. So, the game is you define a top level entity with the input output port in the architecture declaration region you declare the components essentially it is name, ports, direction, data type everything, various components and what are the internal signal we are going to use that has to be declared. Because, that is not part of the input or output that has to be declared.

What is the name, what is the data type and when you come to architecture statement region you say how these components are interconnected using the signal or you can say we define the necklace of this interconnection okay. We say the component 1, particular output is connected to the component 2 particular input using the description language, that is the basic idea.

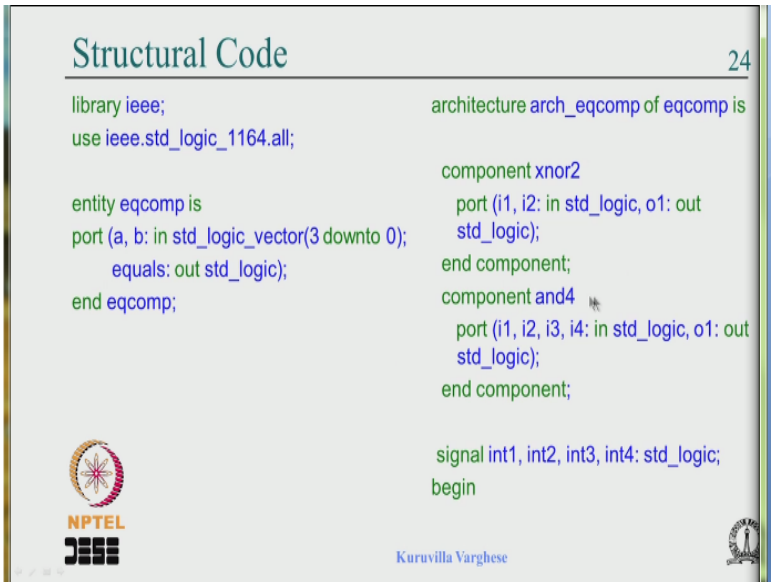
So, to make it clear let us take an example, we will take the same example okay. We will take the quality component we have you know we have describe using the data flow model and behavioural model. So, we will use a same thing and you know it is a four bit equality comparator, so you needs to check the equality of all the pair of four beds a3 equal to b3, a2 equal to b2, a1, a0, b0, and so on okay.

So, this equality is check by exclusive NOR gates and output is going to an AND gate, if all are 1, the output is 1 okay. So, we have 2 distinct components, we have a 2 input exclusive NOR gate, and 4 input AND gate okay. Now we have only 2 components. But, this particular 2 input exclusive NOR gate is instantiated four times. But, we need to declare only once. And, AND gate similarly we have four input AND gate, which is declared once, instantiated one.

But, mind you we have some signals which is connecting the output of XOR gate, to the input of AND gate. So, these are inner signal this need to be declared okay. So, we have 4 signals this 1, 2, 3, 4 that need to be declared. So, we are going to use give a top level entity of the equality comparator. In the architecture declaration region we are going to declare a component exclusive NOR and AND gate.

We are going to declare 4 signals, and in the architecture statement region we are going to show this component will be instantiated 4 times this 1 time and we will show the connections of the wire connection that means will say a3 is connected to the first exclusive NOR gate, first input b3 is connected to the second input of the first exclusive NOR gate. The first exclusive NOR gate output is connected to the first input of the AND gate and so on okay. We will see how that is done. So, let us go to the VHDL code .

(Refer Slide Time: 15:05)



See, you see the entity part is exactly same we have the library risk declaration. The package use statement, entity, exactly similar to the previous one A and B are 4 bit vector equal is a single bit vector. Okay. When it comes to the architecture part you see before the begin there is an architecture declaration, and there we are going to declare components and the signal.

And you see the component this component is XNOR2 XNOR gate this one, and port is i1, i2 in standard logic 2 inputs of type standard logic O1 colon out standard logic one output which is standard logic. And you see normally in architecture entity there is an is syntax but for some reason VHDL does not have component name is there is missing so, do not write is the given error.

And, here the AND gate is declared, so component AND for which has 4 inputs and 1 output. So, the port i1, i2, i3, i4 in standard logic O1 is out standard logic. So, normally when you write the description of the AND gate, and the XNOR gate this component declaration is exactly like the entity. So, it is enough if you write the component entity and architecture .you copy, paste the entity here.

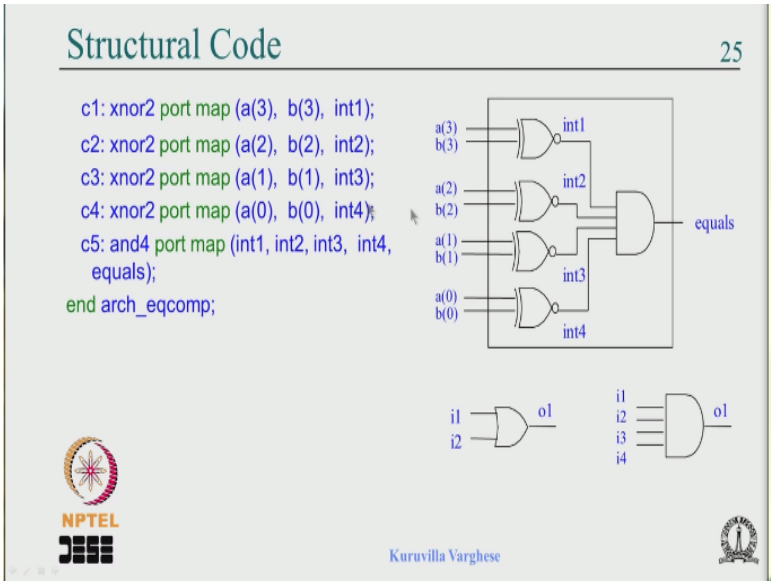
Change the entity and end to the you know entity to the component and this entity name to the component and remove the is then you will get it, and this is similar to a c code when you write a c language program and use a function if the function elsewhere then you write use a function prototype okay. Many times that you will find in the header file which then check for the type argument past and all that. This something similar, because VHDL is a strict data type checking language.

So, ultimately we are going to use instantiate this component and connect some wires to it so, the tool will check whatever the wires we have connecting is of the same data type. So, need to be here. You know that is why this component part is here we will see how to write a component in the library and use it and all so on. But, basic purpose is that and this component need not be as part of this particular the file it can elsewhere or it can be in a library, or it can be elsewhere it does not matter.

Now, before we declare the signal thus syntax signal, name and data type. We have 4 signals which are of the same type we use you know we specify the same statement signal int1, int2, int3, int4 colon standard logic okay. That means that we are naming this as int1, int2, int3, and int4 that is the game. So, we have declared the 2 components, we have declared the signal.

Now, we are going to instantiate this for times, instantiate this one time, and going to interconnect them. That is what is going to be done.

(Refer Slide Time: 18:46)



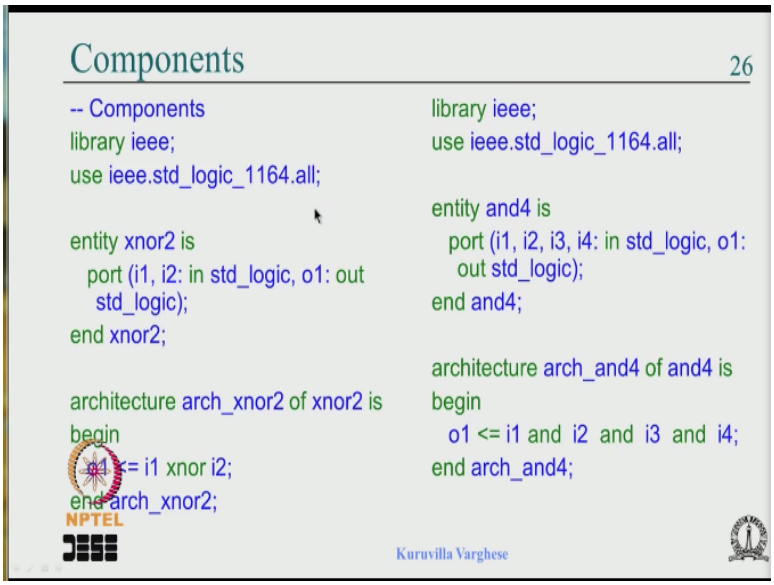
So, this is what is shown here we connect a3, b3 to the first one. The output is int1. a2 b2 int2, a1 b1, int3, a0 b0 int4 and the AND gate get this int1, int2, int3, int4 and the equals. So, the syntax symbol you give a label some label does not matter colon, the component name and we say the port map that means the actual formal ports which we declared as i1 and i2 is map to the actual port. So, we are going to specify a formal to actual mapping that is why it is called port map.

And, we say a of 3, b of 3, and int 1.because, we have declared this as i1, i2, and o1, the same position. So, we are associating a3, b3 and int1, that is this one. Similarly c2, xnor2, port map a2 b2 int2, a1 b1 int3, that is this a0 b0 int4, that is this one, and ultimately this AND gate which gets int1, itn2, int3, int4 and equals int1, int2, int3, int4 equals the same model okay. So, this is called a formal to actual mapping by position or we can say we are associating the actual to the formal you know in a positional way.

So, it is called positional association, so you have to remember the order like you know the AND gate, inputs can be in any order. But, then at least you should where the input is where the output is. Suppose, if we add specify in the declaration equals you know at the beginning then we have to you know bring the equals so, here you know, if the O1 was at the beginning then equals has to be written at the beginning.

So, this is kind of done this is all the structural description happens. Instantiate the components and formal signals are map to the actual signal by position. It is very simple there nothing you know great about it and.

(Refer Slide Time: 21:17)



But, ultimately now we say that this can come from a library or it can come from a different file, ultimately there has to be description of the xnor gate and AND gate, for it to happen may in the library already complied may be in the project in an another file but it has to be written some times by somebody okay. So, I am for completion sake I am showing that so, we have the component xnor2 library close. Entity port i1, i2, and o1and the architecture we have very simple statement o1 gets i1, xnor i2.

And, that is the end of the architecture. Similarly you have an AND gate which is for input. I1, i2, i3, i4 in standard logic, O1 is out standard logic. In the architecture declaration statement

region, we have o1 gets i1 and i2 and i3 and i4 so, simple as it is. So, if you write this and you know you compile it this will be same as the behavioural code or the data flow, model and so on. at this point somebody might ask is there a difference between the earlier code.

And, this code which gives a better circuit. At least for the symbol very simple cases like this does not matter at all. But when you write a very complex circuit using behavioural the data flow and the structural code you can get different circuit it depends how you understand how you describe and things like that. So, we cannot say but, if you follow strict practices what I am going you know teach you.

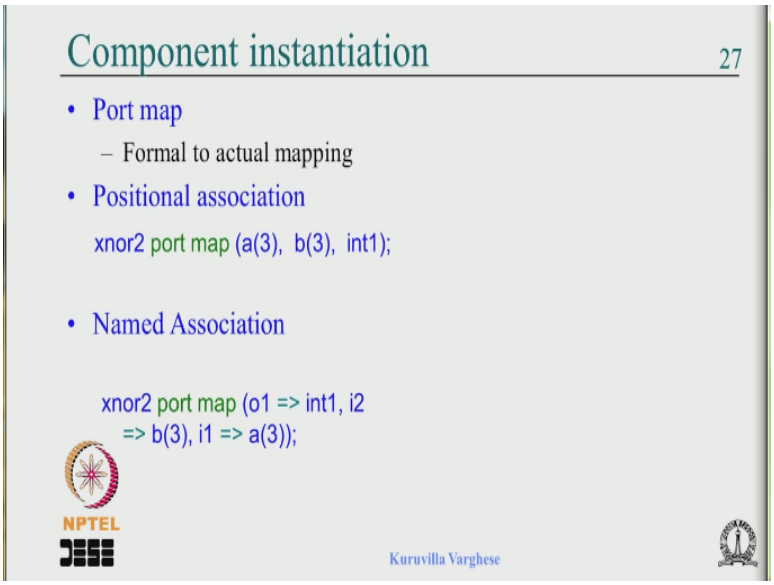
Then you can be sure that you will get the kind of the expected circuit with maybe some kind of optimisation in all 3 cases you need not worry. At least in simple case you close your eyes and you can choose there is no need to frankly if you are making an equality comparator, there is absolutely no need to write a structural code. Just to illustrate because if I take an complex example in runs into pages.

And, just we illustrate I use a very simple code but, you do not do that. You know point in write you know ripple order or a equality comparator, using a structural code. When you have complex design, which is partition into multiple pieces. Then those partitions are combined in a structural code you know, top level of the level, second level and so on. But, when you come down you can still use the behavioural and data flow description.

When you really describe the known high level functions lastly discussed. Like marks, demarks, encoder, decoder, adder, subtractor. All that can be described that behaviourally or in data flow model and when we come up when we put it together use a structural coding. That will be easy to kind of keep track of. Then you can come by in a behavioural way.

But, then it is it can be complicated to track it is easy to use a structural coding at the top level, when you connect the complex blocks together okay. So, let us move on, let us see some kind of let us look at his component instantiation.

(Refer Slide Time: 25:07)



The way we have done the positional association is like xnor2 port map, then i1 is map to a3, i2 is map to b3, and o1 is map to int1 positional association. But, when you this is okay, if you are writing the code done, the code is the same file. Suppose you have written xnor2 code that is in the same file you can scroll down and check the order of the formal ports then you can map it correctly.

But, when somebody has written a library component in a library, it is very difficult to keep track of the order**.** But, they can given input output table where the input signal name and the data type is described, and the function is described. So, it will be convenient if we can forget about the order of the order of the ports we can say what is the formal port name and what is actual port name.

Association can be specified expressively that is named association. Which is the most useful way of you know during the port map so, that is like this you can say for the previous case, this case you can say xnor2 port map of o1 is map to int1, i2 is map to b3, i1 is map to a3, exactly similar to this. You don't have to keep any order. You correctly say which formal port is map to which actual part that is it okay.

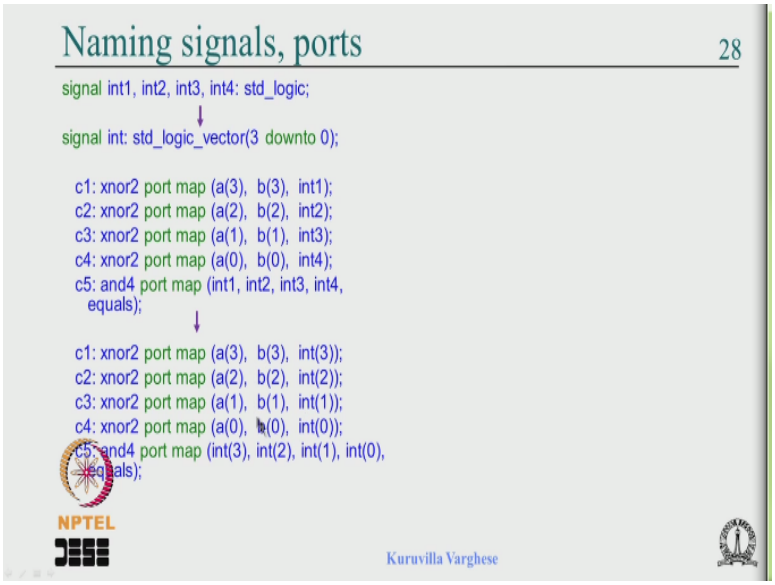
So, that is the named association, the most useful way of you know describing the component instantiation so, let us move on, you know that when you write some software code like if you

are a computer science student, or you written some serious software one important thing before you write the code is data structure okay. So, algorithm we have the proper data structure if you define.

Then the code will be very easy. You can have a elegant the c code if the data structure is correct where you are suppose to use an array, if you use a kind of a distinct variables, set of distinct variables the code will be very clumpsy. So, similar to that in VHDL if you define signals in a proper way, the can be very elegant and concise. So, I am going to show the same structural code we have written.

I will introduce a slight chain and show you how you can simplify, the code how it can make elegant you know make the code elegant. So, let us move on.

(Refer Slide Time: 28:06)

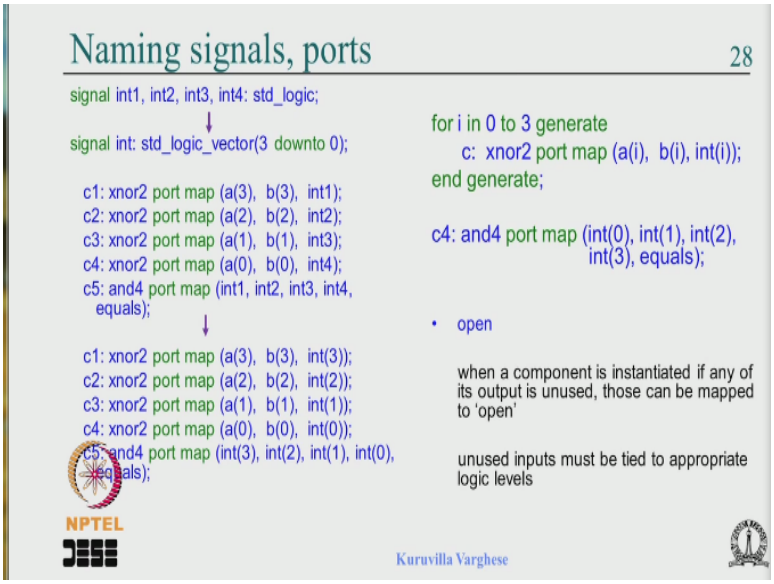


We have declared earlier, the internal signals as 4 distinct signals called int1, int2, int3, and int4 okay. you remember that suppose insert of that I am going declare this signal as I four bit vector okay. Or four bit array, whatever four bit boss okay. So, I am calling int1, int2, int3, int4 so, I am calling the signal int okay. Which is standard logic vector 3 down 0 that means you have an int of 3 int of 2 int of 1 and int of 0.

The moment I write like that, then what I do is that earlier code was written like this. A3, b3 and int1, so I could write now instead of that I have change the declaration to a boss, which is 4 bed. Now, I am going to change you know description which say xnor2 port map a of 3, b of 3, and int of 3. A2, b2 int2, a1, b1, int1, a0 bo int0, in the 4 you know the component instantiation of the xnor gate and the inputs are int3 int2, int1, int0 in any order and the equals okay.

Now, you see here this comes needly you know you have 3, 3, 3, 2, 2, 2. So, you can write something called for this 4 statements you can write a loop call generate loop. So, that is written shown here.

(Refer Slide Time: 29:51)



For i in o to 3 generate c xnor2 port map a of i, b of i, int of i, that means it goes from a0 b0 int of 0, a1 b1 int1, a2 b2 int2, a3 b3 int3. You might ask what happened to the label, we are giving c of i , you do not need to give, you just give c, you will automatically get c0, c1, c2, c3 okay don't worry about the label and ultimately you have to write to write the instantiate the AND gate , now mind you this will not a loop in a conventional sense of the sequential language.

Because, in the sequential language you know something is kind of executed 1 after the other. But, this is just a short form of writing it expressively. When the compiler tools heath us. It will replicate it 4 times like you know a0 b0 int0 up to a3 b3. So, it is a very concise way of you

know generating this statement. So, if you take care when you write the code when you have something replicated with some cemetery, it has to be symmetrical

And, the a proper definition of the signal can help you in getting a very simple concise code and which is easy to kind of understand otherwise you have to suppose you have something replicated you know 12 times than you have this issue, so you can think of doing this for suppose you have a ripple order where the full order is replicated it time heat right the generate you you might need to to do find the signal that carry input, that carry output you know.

That need to be taken care, but that can easily taken care, so that is one example of this generate loop, you can definitely write a generate loop for an (()) (31:58) ripple order using the component as a full order but, suppose if you use a carry looker order that is going to be little tough. You know because in a carry looker order do not symmetric. When you move from 0 to 7 it becomes exponentially bigger and you can do that.

You can write a loop but probably not a generate loop, maybe we will see what to be done when you go air and one more thing about instantiation when suppose you are instantiating a component and one of the output is not connected okay. Then when you instantiate suppose see we are using a 4 input AND gate or you say equality comparator with 3 output 1=1, 1 is greater than, 1 is less than.

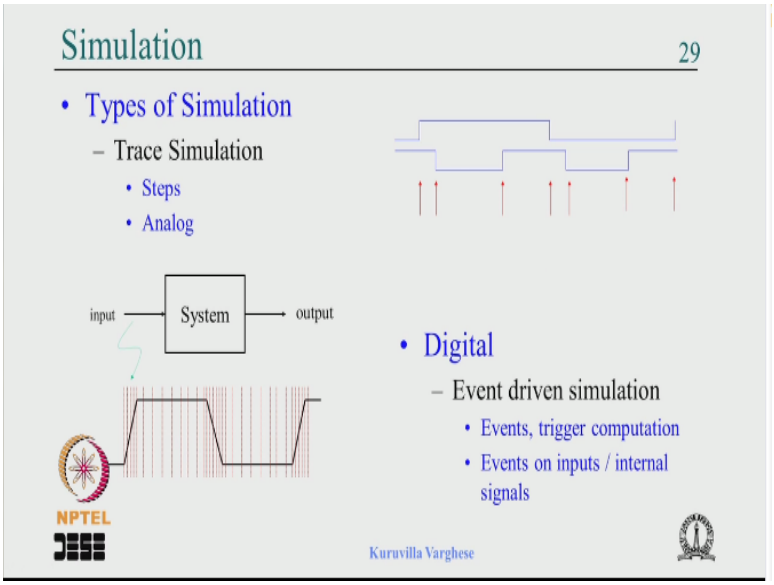
And, we are using only equal to in the instantiation then when you come to the greater output less output. When you instantiate you have to say since it is not use you have to say open and similarly some inputs like you using an in some a s we are using only the 3 of an AND gate. Then one of the input has to be tight to you know 0. The inactive level depending on whether the function is AND or OR okay.

So, you have to properly tie to the appropriate logic levels when an input is not used, otherwise things are work, because we are describing the hardware. So, that need to be taken care. So, let us that is about the different models of description the structural coding is what we have looked at basically we instantiate, we declare components, declare in order signalling instantiated.

And, interconnect them by describing the interconnection using the internal signals and ports and if you elegantly define signal, you can you know use a kind of concise elegant generate loops for a you know kind of when there is quite a good cemetery you can work it out it becomes concise and, if an output is not used we have to say open, input is not used and you have to pull it out for pull it down depending on the function of the circuit to make it an active**.**

So, let us move on now that is the 3 different models of description. Now let us look at the simulation okay.

(Refer Slide Time: 34:36)



I do not know whether you have done any digital simulation may be in your undergraduate you have done the analogue using spice and in a spice simulation, suppose you have a system. May be a filter let us take an example of LOPA filter, and you give some input. So, maybe some trapezoid wave form and you will definitely the LOPA filter depending on the cut of frequency is going to smooth it out, you will get a triangle wave for side wave depending on what how the filter is designed.

But, in a you know that in a spice simulation it will going to divide the whole period into 2 minute intervals okay, need not to be uniform okay. because, you know in a trapezoidal wave maximum information is in this kind of slope all the harmonics the frequency content comes

because of this kind of rate of change. So, there will be lot of division, lot of steps there, when it is stable maybe less steps.

So, this spice is going to divide into interval and there will be computation of the input with the system transfer function generate the output okay. So, that is how the analogue simulation or the spice simulation is done. So, let us see how it is different for a digital circuit okay. Now, look at a digital circuit so, you see here I have shown some like you can imagine like this is a digital system now.

And, we give some input to the system okay. Now, you see here this input is changing like that. And, this one changing like this okay. Now, suppose initially at the first step the simulator take this value 0 and 1 and compute output. But, you know the moment in this computed there no point in dividing into this into minute intervals and computing.

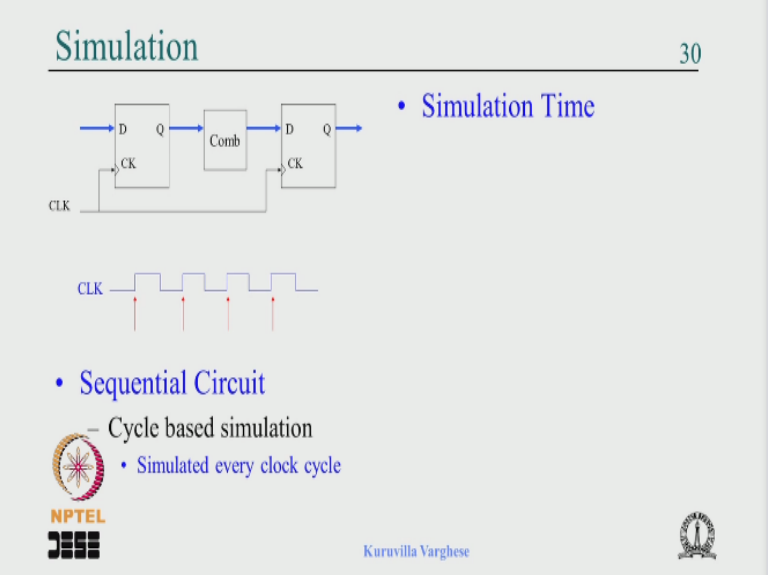
Because, it is going to be stable, because digital system is binary. But, here you see when analogue signal, analogue signal changes the value moves from all the values are important. It may be going from maybe 0 to +15 volt, each voltage is important. But, in our case either 0 then 1, so there no need to compute 1, 01 is computed the next computation need to have only here.

Because, the input signal is change from 0 to 1. So, now you compute here at this point one. The moment you compute till any change happens, then no need to compute. Because, it is going to remain stationary. So, as for as digital circuit is simulated digital simulation is concern. Whenever there is a change in the input, at that point the simulator need to compute okay. That means whenever we say, whenever there is an event on in an any of the signal at that time the computation should happen okay.

So, in digital simulation is an event driven simulation. Whenever there is an event on any of the signal, than the simulator has to compute, otherwise nothing to do okay. So, events trigger the computation, an events can be on the inputs maybe event happen like you give the inputs to various input signal then you have to compute. But, because of the input something can change internally.

At the time also that should trigger a computation. So, in digital circuit unlike the analogue circuit, we use event driven simulation like you take a combinational circuit, then we have to simulate by you know it is an event driven simulation. But, let us come to the sequential circuit.

(Refer Slide Time: 38:52)



Now, when you have sequential circuit, suppose input is changing quite a bit, suppose we give up clock which is say of frequency say 100 mega hertz, then the duration is 10 nanosecond. But, maybe the all inputs coming to this register, say we have a register, combinational circuit and a register. Here the input is changing every 1 nanosecond. But, we know that this flip flop is going to be lacks every 10 nanosecond.

So, there is no any point in computing the simulator the compute every 1 nanosecond okay. so, in principle for a sequential circuit you can compute on the active clock ends and you can get the value that is called cycle based simulation. Every clock cycle active clock edge you can do the computation, definitely you will miss lot of things, because of maybe that there could be lot of changes happening here.

And, unless that is all computed the in between things can be last button. This is the very quick way of simulating. You don't simulate you know every time the input changes. When the clock comes just before the kind of that in whatever is in stable input, that I only need to compute but,

you might you know you might not be able estimate the power displacing and things like that in the simulation of something will be missing.

But, does not matter for sequential circuit. Because, updation happens only on the clock edge. So, the cycle base simulator, simulation can be use in sequential circuit. So, that is the different types of simulation. And, in simulation there is a notion or a terminology we use simulation time okay. Now, you should have some clarity on that**.** First up all it is not real time okay.

It not that it is the time of the day or some kind of clock which is running which indicates the real time okay. That is not the simulation time, so real time is not simulation time. It is not even the time taken by the simulator compute something, you say you have given a complex circuit for timing simulation. It takes 3 hours to simulate whatever you have given.

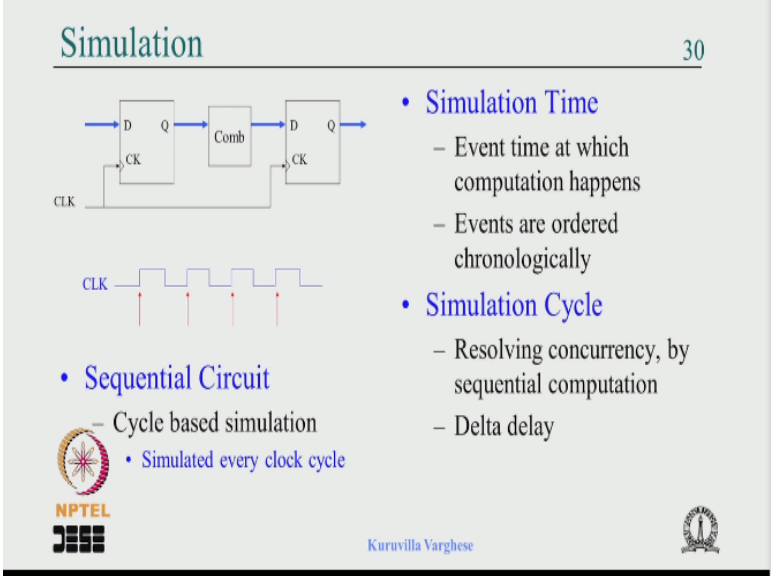
That is not the the time taken by the simulator is not the simulation time. Simulation time basically is a time at which the events happen okay. So, when you say at 100 nanosecond an event an input a changes. Then we say the simulation time is 100 nanosecond okay. Then this simulator will start some computation, which might take any amount of time. Like at 100 nanosecond, and event happens and simulate a compute maybe will compute in few nanosecond or microsecond or millisecond or hours or day it does not matter.

It might take a day for the computation for the 100 nanosecond event. If the circuit is complex okay. Now, after the day everything is done if the next event in the Q is 200 nanosecond, the simulation time now goes from 100 nanosecond to 200 nanosecond okay. As well as digital simulation is concerned. Because it is event base, in the next event in the que in the order is 200 nanosecond.

The simulation time goes from 100 nanosecond to the 200 nanosecond, so that is this simulation time do not confuse. Now, I am giving a very kind of useful explanation for you to understand the concept. I am not going to be very precise, I am going to avoid all the terminologies. You can go back to some textbooks, there are all lot of you know the names the terminology which is used put regard to simulation.

And, avoiding everything you can refer to the reference book and learn all that. But, it could be confusing for a beginner to go through that. I am telling you that cracks of the matter in a very used way so, it cannot be very kind of precise part I am trying to explain in concept in a clear way. Hopefully you can understand and very important.

(Refer Slide Time: 43:37)



As use that events are ordered in the chronological order or in the simulation time. What I mean is that suppose you are trying to simulate a circuit and you are given say your AND gate you are trying to simulate, and inputs are there. Say, you give some wave form with changes of 100 nanosecond, 200 nanosecond, 250 nanosecond and 300 nanosecond. The simulator will you know order the events like 100, 200, 300 like that, you know in the order.

But, suppose is AND gate is driving another OR gate, the AND gate output is driving another OR gate. Now, because of change in the input of the AND gate 100 nanosecond, output changes maybe after say 101 nanosecond okay. So, when the simulator computes, earlier simulator had say 3 events at the input of AND gate 100, 200, and 300. But, when it simulate it find that the output of the AND gate because of the 100 nanosecond change as changed that 103 nanosecond.

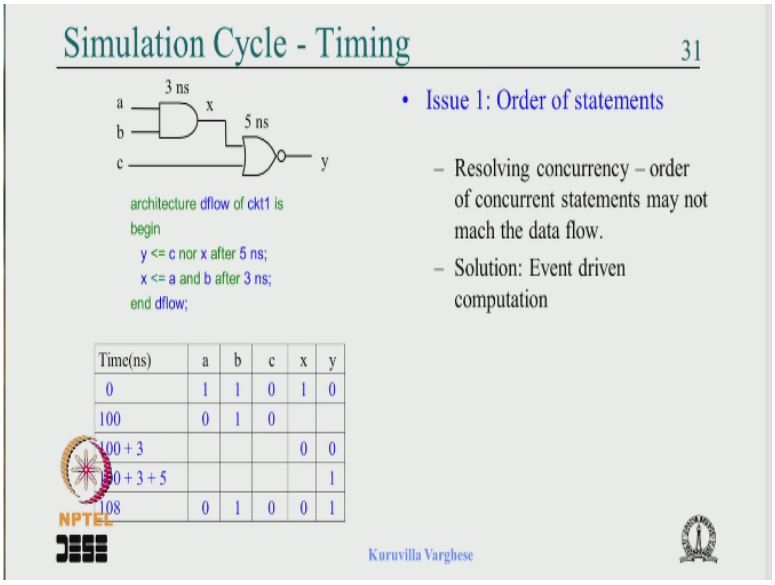
So, now it has to odd and even between 100 and 200 at 103. Then only the next OR gate behaviour can be simulate properly. So, that is what I mean the events as it happens like when

the simulation happen, maybe there are you give some inputs, some events and it is ordered properly. But, the simulator as it simulate new events happen in the internal signal that is pushed in at the appropriate place in the correct order.

And, one by one the input is taken and simulated, so that should be understood, the best thing is that you draw a some 2 or 3 level circuit, imagine some input. and you try to compute and order it in the proper way. Then you will understand it is better you yourself work it out, I could have shown a picture. Maybe as we go a long, some pictures will be shown which is related from there you can grasp.

So, that is the idea simulation time, so do not get confuse and there is some the most textbook will talk about a simulation cycle. Basically, it shows a resolving the concurrency by sequential computation okay. And, they will call you know describe something called delta delay. We will see that in a moment okay. So, let us move on, let us see how the simulator simulates the sequentially written code or the concurrent behaviour is simulated properly.

(Refer Slide Time: 46:32)



So, please have a look at the circuit which is shown here. So, here we have a very simple circuit an AND gate and NOR gate and a, b is going to an AND gate. And, the output of the AND gate is X, X is X and C primary input is going to a NOR gate which is output is a Y. And, the AND gate

has a delay of 3 nanosecond, and NOR gate has a delay of 5 nanosecond, and we know that the AND gate one is one.

For the NOR gate if any input is 1, the output is 0, because it is opposite of OR gate. Now, we have not learned about how to specify the delay in VHDL. But, this is the way to specify, we are writing the description, but we are writing the statement, concurrent statement in the order of the data flow. Data is flowing from a b to X, X and c to Y but we are going to write a this part first and this part second.

Say Y kept c nor x after 5 nanosecond, that is this one. And, this is way to add a delay the model the delay in VHDL, only for simulation, just like if you write a statement like after 5 nanosecond, definitely you are not going to get a NOR gate with 5 nanosecond delay. If you give to a synthesis tool that is going to generate circuit like this ignoring this, it is not going to give any delay you know or it gives if you go to FPGA map some circuit.

Whatever may be the delay of the circuit you get it okay. There is this is not for synthesis, it is only for simulation. But if you simulate this circuit it will definitely show the output will come after 5 nanosecond delay okay. Now x, a and b after 3 nanosecond. Now, you see that the Y is written first and X is written after that. So, the order of the statement is opposite to the order the way data is flowing.

Now, normally as in sequential language simulation, if the simulator looks at the like there is an event on a, it computes say as then x then nothing is going to happen now, the y says c nor x after 5 nanosecond. Whatever was the old value of x is use for y computation. But, x is computed and where the new value of a is used. And, x is correctly computed, y is not correct.

You can imagine what happens if you have a five level circuit, and everything is jumbled up okay. So, this is a little bit hopeless situation, we have to make the simulator has to make some sense out of it. Now, we cannot that look at the ordering which the data is flowing and rearrange the statement and so on. This one work definitely you can have feedback okay.

So, when like suppose you have something one more level and output is going back to the input, then what is the order when there is a feedback,, there is no order, it is like you go to a stadium where a running race is in the progress okay. you go you know people have a running a long distance 10,000 meter running in a 400 meter track and you go in the middle, and if look and you might guess saying that somebody is front maybe the you will imagine the tail is the topper top end like that you know.

So, similar thing in a when you have a cycle you cannot decide which is the first. So, ordering it in a proper way is impossible. So, but if you think it is very easy. Because, you know that the computation start because of an event. If an event happens on a, then first thing is do is instead of going from top to bottom, look at the right hand side of the assignment.

Wherever there is a you compute that, so that is a game and there could be multiple statement with the a does not matter suppose the simulation time is 100 nanosecond, at 100 nanosecond you compute x and there may be an with a you compute z really does not matter okay. Whatever I am going to show you is a simple case but, can be kind of interpolated to a more complex case only you need to work it out okay.

So, the rule of resolving concurrency is very simple look at the right hand side of the assignment, wherever there is an event and compute that, okay now you take this simple case, let us assume that a is changing at 100 nanosecond, so suppose initially a was 1, the x is 1. So, a is changing from 1 to 0. Then look at the simulator will look at the right hand side, that is an event an a.

And, you look at the right hand side here, there is nothing, so this is need not be computed. So, a change at 100 nanosecond. So, it will compute new value of x and it will assign 100 + 3 nanosecond. 1 or 3 nanosecond. The moment that happen now some signal as changed. Now, again the simulator will look where the x is being at 1 or 3 nanosecond, so what the simulator does is that current simulation time is 100 nanosecond.

And, there is a 100 nanosecond event is computed some signal you know some assignment for 1 or 3 nanosecond. If nothing else need to be computed for 100 nanosecond, it will keep the event

in order. So, suppose next event on x was 200 nanosecond. Now, since there is a sorry, the a is 200 nanosecond. Since, there is an event on x at 1 or 3 nanosecond. It will look at the right simulation time is forwarded to 1 or 3 nanosecond.

It will look where x happens, and you know compute the y, and that will happen after 103 + 5 nanosecond, again it looks if there is anything to be remaining to computed for 103 nanosecond. And proceed for the next simulation time 108 nanosecond and so on okay. That is how the simulator resolve the concurrency, it does not matter, however many statements are there.

Whether there is feedback, in whichever order and instead of the AND gate there are very complex circuit, this game works, you know anything can be easily the concurrency is resolved by very simple method of looking at the right hand side of assignment. Where the event happens, that is computed. And, new event happens that is ordered properly, and it proceed.

So, that is what is shown here so we have at the time 0, a and b are1, so x is 1, c is 0. So, because of x 1 y0, now at the 100 nanosecond, it remains there a changes from 1 to 0. So, the moment that happens this a appears of the right hand side, x need to be computed, so x is assigned, while you because a is going from 1 to 0. X will go from 1 to 0, but that happens at 103 nanosecond, and y remains same.

Because, at 103 the y is you know there is no change in the x at this point at 100 nanoseconds, Y does not change. Now, since there nothing else here compute at 100 nanosecond, the simulation time moves from 100 to 103, and there is an event now on x, that will you look at the right hand side that will trigger a computation for y, and that you know that is basically 0 nor 0 which is nothing but 1.

So, the y change from 0 to 1 which is assign at 103 + 5 nanosecond, 108 nanosecond, since there is nothing remain to be computed at 103. the simulation time move to 108 nanosecond, and everything is stabilise, so if you put a wave form this comes very cleanly. So, basically the resolving, the game is about resolving the concurrency the issue is order of the statement will not match the data flow.

The order may not may be different and solution is an event driven computation, and you look at the right hand side of assignment, and one after the other you compute and that might trigger the further event and you order them properly. And, you know then you treat them one by one. Everything solve properly, if you have doubt you can really you know I have definitely shown some very simple case.

You can introduce say like event on a at 100 nanosecond, at 1 or 110 nanosecond, 120 nanosecond, you can think of a scenario where a is staining at 100 nanosecond, and also at 102 nanosecond, before even the x changes okay. So, you think really what happens, you know you try to work out such a situation, that will bring clarity to your thinking.

And, so this shows that essentially what I told is that how a simulator resolve the concurrency from the sequential written statement, might is a very idea it is not a very complicated idea, but simulator may have to compute quite a lot. A lot of computation need to be done, but idea is very simple whenever there is an event look at the right hand side compute those statement which as an event.

Assign the output order the event sequentially properly and then when avail computation at the particular simulation time stabilises go to the next event next simulation time, and keep computing, that is a basic idea. So, with that I try to complete today's lecture, the second part of lecture shown we have discuss the simulation how an spice simulation happen for a long circuit by you know kind of discrediting at the signal at various intervals.

At depending on need of computation but in digital circuit you no need to kind of you know introduce intervals, wherever whenever there is a change in the inputs that need to be recomputed. So, the events whenever that is event driven simulation for sequential circuit, basically you can do whenever on the cycle bas simulation on the active clock end. That will reduce a computation.

We have seen a simulation cycle how the simulator resolve the concurrency in the case of timing simulation okay. We will in the next lecture we will see how that happens in a logic simulation where there is no delay. That will definitely bring more clarity to the game. So, I stop here please go back and revise whatever is done refer to the textbook to get hold of the terminology use because I have avoided to explain the concept in a very simple and used way. So, please revise refer to the textbook thank you and I wish you all the best.