Advanced VLSI Design

Prof. D. K. Sharma
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture - 23
Introduction to Verilog

In this series of lectures, we have looked at the general principles of Hardware Description Languages and gone through the details of VHDL which is one of the leading hardware description languages, these days Verilog is also extremely popular indeed in commercial circles perhaps even more popular compared to VHDL. Therefore we shall have a brief review of Verilog as a hardware description language.

The underlying principles remain the same, but there are some differences in how various objects are handled in Verilog as opposed to VHDL, therefore we will begin with first understanding these major differences, otherwise it is largely a matter of syntax and once you understand how to program in one hardware description language it should be possible to adapt to the syntax of another one.

The additional advantage of Verilog is that it syntax is very similar to C indeed as a just like VHDL is based on the programming language ADA, Verilog is based on C which is familiar to a very large number of people, therefore Verilog unlike VHDL is case sensitive just like C, it uses essentially the same comment, markers etc. etc.

So in general it looks and feels very much like C of course as we have often remarked a hardware description language is very different from a programming language and therefore it retains all the characteristics of a hardware description language. So let us first of all see what are the major differences in the objects which are handled using Verilog first of all Verilog makes a distinction between two kinds of signals the position taken by Verilog is the following suppose a particular node is say 1.

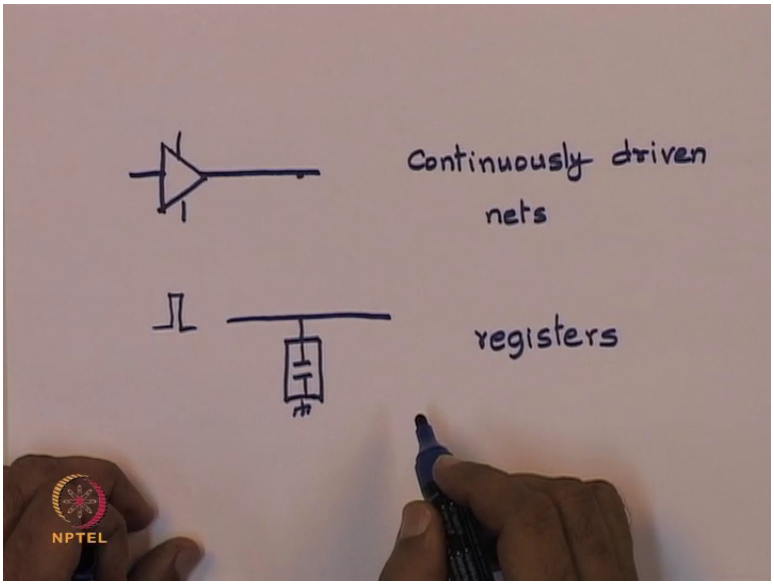
The question is who is keeping at one who is keeping this node at the value 1, there are two possibilities, one there is a driver attached to this node and this node keeps this signal at 1, so it is

the duty of the driver should there be any noise or any loss of charge or whatever to make up this losses and to keep this node at 1 such signals are called continuously driven signals and are in general classified as nets.

There is another class of signals which hold their value simply, because there is an associated memory with them that means the assignment to these nodes is not continuous they are not continuously driven they are given a evaluate at an instant of time and they have the inherent capability of retaining this value, because of the memory associated with them, this class of signals are called registers unfortunately this choice of terminology is not very good.

Hardware designers tend to think of registers as a static latch or a complicated circuit, whereas a particular node might we termed a register simply, because it has a capacitance and once you place a 0 or 1 on this, it is likely to hold this value till a new value is written.

(Refer Slide Time: 05:05)



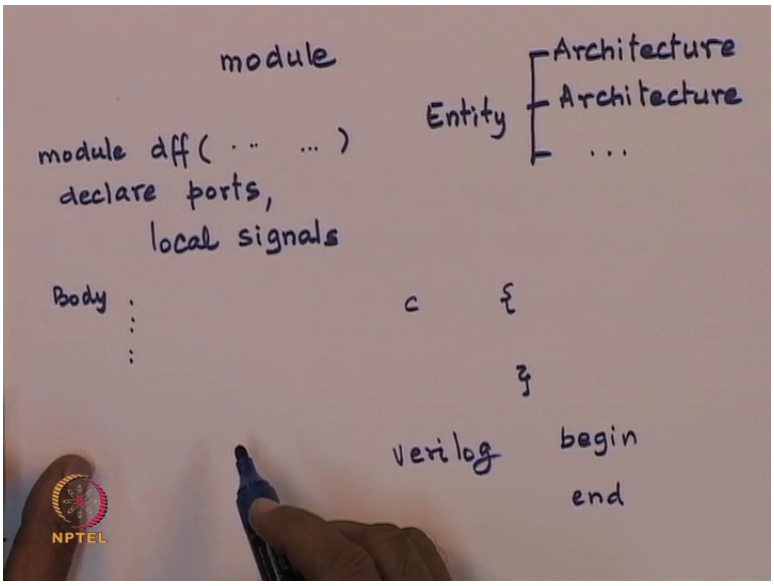
So in short the picture that we take is the following you have a particular node and either you have a driver which keeps this node at the value which is assigned to it in that case this driver is continuously connected to this wire and therefore this node is continuously driven and such nodes are called simply nets, you might have a node which has notionally it may not be a piece of hardware, but you have notionally a memory associated with this node.

And now you can assign it a value at an instant of time you are not continuously driving it to that value, the assignment is made at any instant of time, and now it is the memory associated with this node which holds the value at this point, now this memory could be as simple as just the capacitance associated, so such nodes are called registers.

So all signals are either nets or registers remember if you have a programming language like interface which is what we called a process in VHDL in that case the assignment takes place only at one instant of time and such assignments are permitted only for registers, nets are much more common for the structural kind of description in which you place hardware and then you connect wires to this.

We shall see later what are the sequential and concurrent elements in Verilog and the kind of things that we choose, kind of signals that we choose depends on whether these nodes are inside those sequential elements or outside, so this is the first major difference you have nets and you have registers in Verilog, whereas you just had signals in VHDL.

(Refer Slide Time: 08:23)



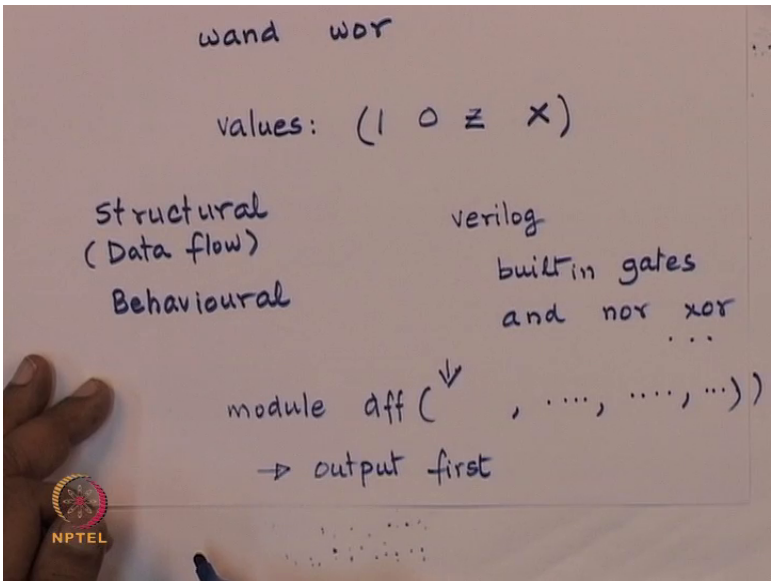
The other difference is the basic hardware object in the language this is now called a module, the module is in fact an amalgamation of the entity architecture combination that we had in VHDL, in VHDL you could have a single entity with possibly many architectures associated with it and that is because entity and architectures were different bodies.

In case of Verilog we do not do this, the entity and architecture are together so there has to be only one architecture and you change the entity all together if you want to change its behaviour, so unlike VHDL you have a module and the module like a C function has a port list associated with it, so you might have for example a module D flip-flop and now you will have the port signals as in the case of an entity declaration.

You will now declare the ports and other local signals and then this is followed by the body of the module which is equivalent to the architecture, now one difference from this syntax of C is, that in C we use curly brackets to begin and end blocks, whereas in Verilog we use the keywords begin and end as in C the statements are terminated by a semicolon, however begin and end are commands and do not take the semicolon, okay.

So this is roughly the structure of the basic description of a piece of hardware. Now let us look at the signals like nets and registers in a little more detail.

(Refer Slide Time: 11:33)



The most common kind of net are the wires and a wire is just that it is like a wire connecting one node to the other, it has to be continuously driven like any net should be. Now, there are other kinds of wires these are called wand and wor, these are things which are wired and wired or combinations and should the wire be multiply driven then the resultant value on the node is the

end or depending on whether it is wire and or wire or correspondingly the resultant value on the net will be the and or or of all the driving signals.

So this permits essentially the open collector and other kinds of circuit in which you can have multiple drivers on the same wire, for example a microprocessor bus can have multiple drivers and the resultant value will be decided by a logical function of all the driving signals, there are also in fact that term tri is also used for wires and tries are signals which can from which the driver can be removed which can be tri stated and just like wire and and wire or you can have that tries with and and or associated with them.

Now a Verilog uses a signal value system by default which is 4 value, now if you recall in VHDL the standard thing defined by the language was bit which was 2 value but the most commonly used signal type is std logic which is invoked by including the IEEE standard library and that was 9 value. Verilog takes a middle of the road approach and permits 4 logical values for each node, these 4 values are you can have 1 0 these are the standard bit values.

But you can also have Z which is a high impedance value that means this node is not being driven currently even though a driver is attached the driver is driving it to Z that means the driver has been tri stated and X which is an unknown, for example a signal simultaneously driven to 1 and 0 will acquire the value X because its value cannot be determined. So these are the various values which can be assigned to signals in case of variables or in case of signals, in case of in Verilog.

So we will use this 4 valued logic throughout in our discussion on Verilog, now just like VHDL we have various styles of describing hardware we have for example structural descriptions there is a data flow kind of description and behavioural, now notice that when you there are various modules which are built in and known to the language this is different from VHDL, so Verilog has built-in gates.

So the gate level descriptions do not require you to define the behaviour of say NAND gate as we did in case of VHDL, so it has built in gates and all the standard gates like AND, NAND,

NOR, XOR, XNOR etc. these are all built in, this immediately brings a kind of problem first of all and which has a solution the AND gate and have a variable number of inputs so you have a variable number of inputs and one output.

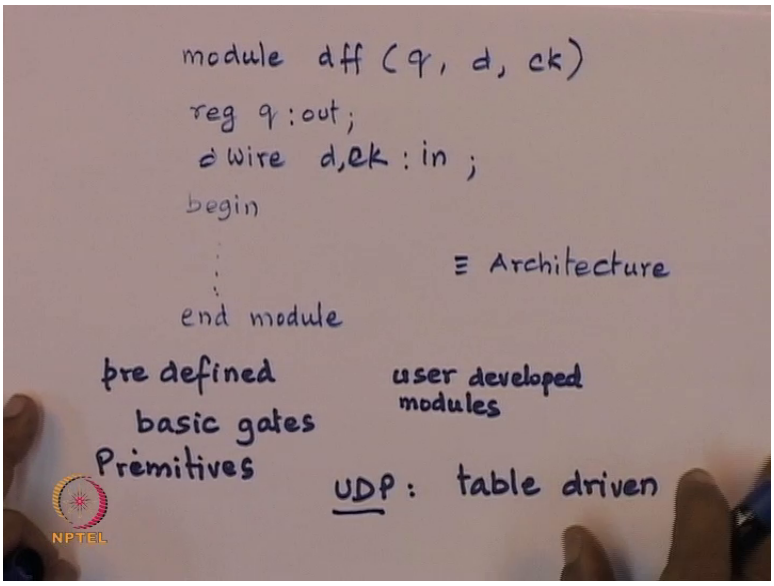
Similarly, NOR, OR or XOR gate could have a multiple number of inputs and a single output this being the case it is difficult to describe the structure which you will be using, the module is pre-defined for you by the language but you have to place the component in your design suppose you want to use a three input AND gate now the AND gate is provided by the language but how do you specify all the inputs and outputs of this piece of hardware.

Now because for most of these there is a unique and single output, therefore in Verilog it is conventional to put in the signal list so you had for example module say D flip-flop and here we have the signal list the port list in the port list you place the output first and put as many inputs as you like subsequently.

Therefore since the first element is always there the output is identified and depending on how many additional parameters are provided the language knows how many inputs does this gate have of course this is a user-defined component D flip-flop is not a part of the language, but even for user-defined components we tend to follow the same convention, so that we are consistent and essentially what it means is that this signal list is output first.

The declaration of these signals can be in line just like C functions the type of these things can be in line right here or it they can be declared immediately afterwards.

(Refer Slide Time: 19:29)



So to complete the example that we started with the module D flip-flop could have let say a signal q as the output and d and clock as the inputs, now because you are going to assign to q inside this module and perhaps only once and this will retain its value, therefore it should be a register, so the main type of register is called reg and therefore you will say reg q which is an output signal, similarly d and clock will be declared as wires this completes the declaration of these signals.

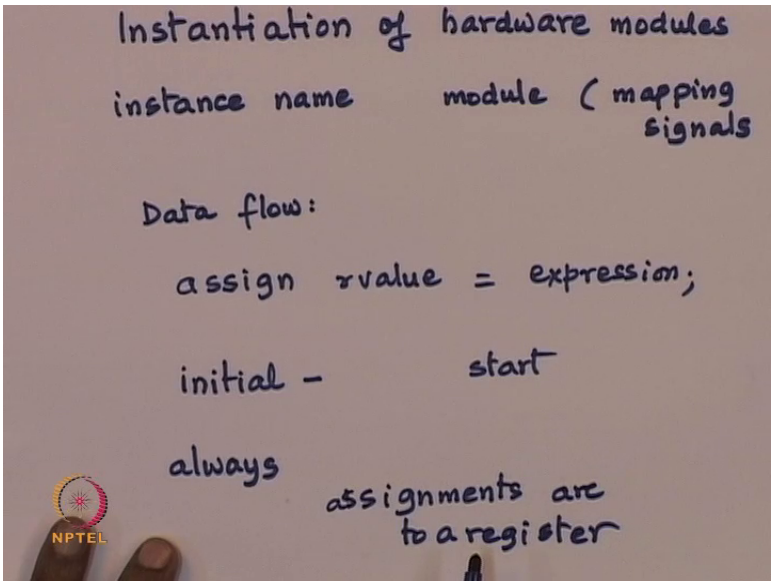
This will be followed by a begin and end module and here will be all the logic the equivalent of architecture, so this is the general structure of describing a hardware element in Verilog having got this, so now we have essentially a combination of various hardware modules some of this these are pre-defined these are all the basic gates and then there are some which you as a programmer as a hardware description language writer will described.

So these are user developed, for example you have this particular D flip-flop and now you would like to put all this together and interconnect them, in fact there is a user-defined primitive or a UDP which is defined in Verilog, the user defined primitive becomes equivalent to the basic gates defined in the language but the user defines this primitive as a table driven architecture, where the output is a function of all the inputs and all possible combinations of inputs are given in a table.

These combinations include values as well as transitions, so therefore the behaviour of a UDP is nothing more than a table lookup, so the inputs are presented to the UDP and the output is just read from the table from which our description matches, this is a unique feature of Verilog and no such thing existed in VHDL.

So now you have a combination of the basic gates or the primitives the user-defined primitives and the user developed modules, these are primitives you have user-defined primitives and then user developed modules like this. Now in structural Verilog you will put these all together and then the structural Verilog is very similar to structural VHDL.

(Refer Slide Time: 24:19)



And in this case it is just a matter of instantiation of hardware modules and then you simply instantiate them by giving the instance name, what kind of module you are instantiating and what kind of signals you are mapping, remember we did this using a port map in VHDL, once you have described all this using signals which have been pre-declared you have described a interconnected set of known components and that is the style of structural description.

We also have a style which is data flow in which we use a continuously assigned signal that means you have the keyword assign followed by r value this is the recipient of the value to be assigned equal to some expression, so this expression is evaluated this expression could be a

logical function and that value is then assigned to this signal this is the target of the assignment and this is a continuously assigned signal.

Notice a continuously assigned signal will then forever drive this node with this value, this expression will be reevaluated whenever it is sensitivity is struck that means whatever signals take part in this expression if there is an event on any one of those then it will be evaluated and that will then be assigned to this r value but the module remains the same that you have a driver with this value which is permanently keeping this net at the value of this expression.

So it is a continuously driven signal. on the other hand there are two kinds of processes in C and these are called initial and always, initial and always are otherwise equivalent but initial is run at the start and it is run exactly once, so this process is started it is triggered at T = 0 that is the trigger point for starting it and then where it terminates it is never run again, another thing to remember is that the contents of initial are ignored by most synthesizers.

And therefore the contents are not supposed to be synthesized into the circuit that you are describing therefore the purpose of the initial block used to setup the input signals and so on and not really to describe hardware, in general initial is widely used in test benches which then apply a sequence of inputs to the hardware that you are describing.

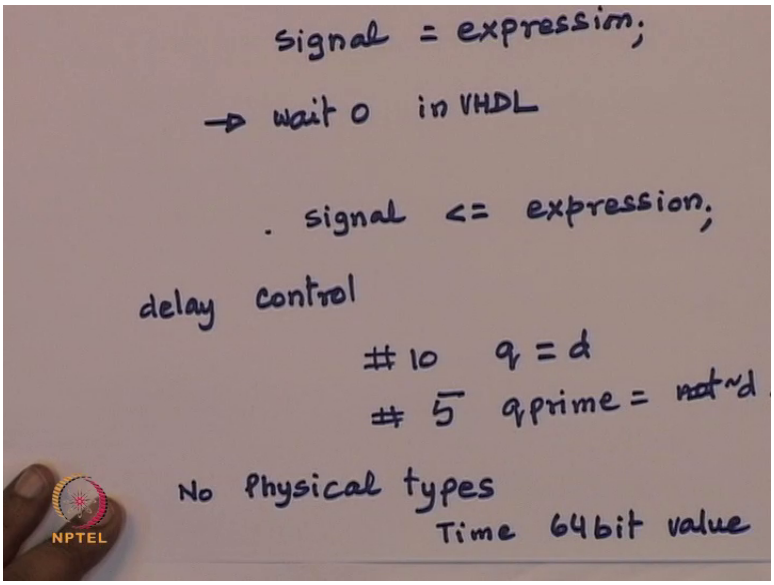
Always, on the other hand is the standard and the most widely used structure for in Verilog for describing sequentially the behaviour and the innards of always are like innards of process that means they look very much like a programming language and most of the programming constructs of C are available as programming constructs in always blocks, so therefore it is really convenient to use this.

Now notice that in a sequential block the assignment is made at a particular instant, so what is this instant at which this assignment is made and because the assignment is instantaneous therefore the assignments in an always block and indeed in the initial block also, they should always be to a register so assignments or to a register, it is for this reason that in the D flip-flop that we described we declared q to be registered.

Because we will be depending on whether clock has had an event or not, we would assign to q a value inside a logical block which should be an always and therefore q had to be a register, in general there is an implied conversion from a register to a wire at the output of a module so then you can connect a wire to the output. There are some very strong notional differences between VHDL and Verilog and we must be aware of those.

One of these is blocking and unblocking assignments, these make a very big difference and we must understand blocking and unblocking assignments in Verilog in order to make full use of its facilities.

(Refer Slide Time: 31:09)



The ordinary procedural assignment which is essentially just signal equal to expression, this implies that the next statement will be taken up only after signal has acquired the value of expression that simply means that the next expression can assume that signal has already been updated to its value, this is equivalent to inserting a wait 0 in VHDL, these are essentially blocking statements they block until the L value the signal on the left has acquired its value.

There are also non-block statements and the syntax for that replaces this equal to sign by an assign to sign, so for example you can have assign to signal this expression, this kind of assignment remember all of these will occur inside an always or an initial block, this kind of

assignment is the standard VHDL kind of assignment that we discussed that means only a transaction is placed for making the signal equal to this value.

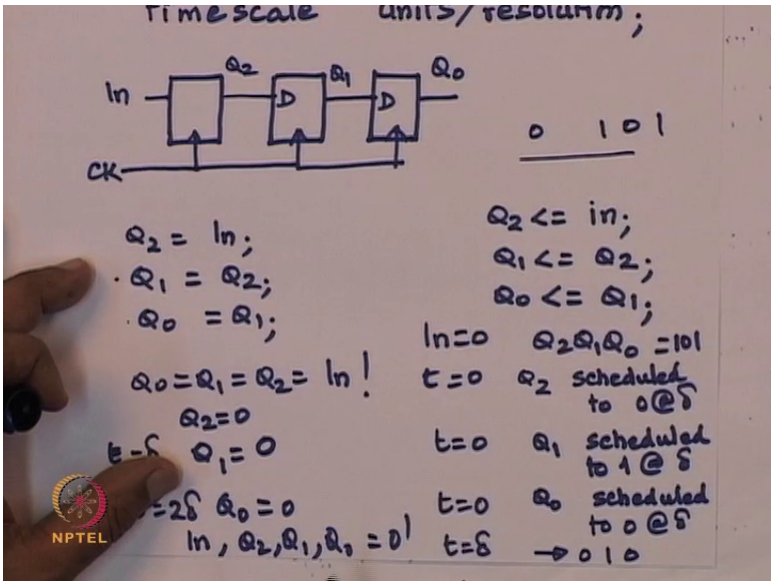
And then you move on before the signal has acquired this value and usual timing analysis that we had done in the tutorials that we did a few lectures ago that follows that applies only to the non-blocking kind of assignment, the blocking kind of assignment you place a transaction and this transaction has to fructify before you move on to the next, what this means is that now you have to be aware when this or this assignment is made and this is done through two kinds of controls.

One is a delay control that means the assign assignment is made after a certain amount of delay has passed, so for example you might have something like this it is simply says wait for 10 units of time and then assign the value of d to q, this has to acquire the new value only then you will go to the next statement and therefore if there are delays associated with various statements, okay.

So the not operator is actually the tilde operator so that means this will be attempted at 15 units of time that means delays or cumulative in a Verilog always or initial block that is because this unit will first wait for 10 assign the value of q to q this assignment is complete then you will wait for 5 units of time and then assign the value of not d to q prime, so therefore by the time both q and q prime are assigned 15 units of time have passed.

Notice by the way that there are no physical types in Verilog, so time is just like a 64 bit value and the implied units have to be declared right in the beginning by a statement which is time scale.

(Refer Slide Time: 36:19)



So therefore all design modules begin with a time scale declaration and this gives the time in units resolution format, the implied unit are this and these units are maintained with this resolution, for example you might say that the units are 1 nanosecond and the resolution is 100 picoseconds, so the time will be quantized to picoseconds and the values will be reported in nanoseconds, you can choose your units.

Now this particular declaration causes all 64 bit values of time to be scaled according to the time scale value given and interesting combination occurs when different modules which are being combined use a different time scale, in that case internal conversion takes place of all those times and the lowest unit is the unit which is internally used, however there is no inconsistency you can freely mix different time scales and resolution and the reasonable thing will be done.

There are other features of the language which are different but one particular side effect of blocking and non-blocking assignments must be well understood, otherwise you may land into trouble while using Verilog and this I illustrate by the case of a Shift register. So let us say that you have a shift register and the output of this becomes the d of this, so this could be Q2, Q1 and Q0 this is the input all of these are clocked together by the same clock signals.

Now we suppose in an always block we say that at posedge of clock by the way that even control will come to in a little bit, but let us understand this concept first so when there is a positive edge

on clock we are going to say that Q2 = In, Q1 = Q2 and Q0 = Q1 this seems a reasonably good description of this hardware saying whenever the clock has a positive edge then make Q2 = input make Q1 = Q2 and make Q0 = Q1.

However this can land you in trouble, notice that this is a non this is a blocking statement therefore Q2 will acquire the value of In before the next statement is executed, therefore Q1 will not be assigned the value of old value of Q2 it will in fact the assigned the value of In, because by the time you came here Q2 has already acquired the value of In, similarly Q0 will also be acquired will also be assigned the value of In.

Because Q1 has completed acquiring the value of Q2 before the next instruction executes, as a result essentially you have Q0 = Q1 = Q2 = In in one clock cycle that is equivalent to same that you are describing not this circuit, but a circuit in which Q0 Q1 Q2 and In are all shorted, this of course will not do what you want to do is to describe a shift register and the answer is that in this case we must use non-blocking statements which will follow the kind of logic that we had developed during the tutorial.

So in that case you assign Q2 like so assign Q1 and Q0 by these, now what happens well a transaction is placed on Q2 to acquire the value of In in the next time cycle here of course the delay is 0 so therefore in next delta, however currently Q2 keeps its old value and because this does not wait for Q2 to acquire the transactions value the next instruction is executed before Q2 value changes.

As a result the correct value is then scheduled to be given to Q1 that means the transaction is placed on Q1 for the value Q2 and the same thing for Q0 and this will have the correct shift register behaviour just to give you an idea let us say In is 0 and Q2, Q1, Q0 are 1 0 1 at t = 0 and then these instructions are executed, in the two cases we will see what happens in the first case Q2 is made 0 and indeed acquires the value 0 before we move to the next instruction.

So at t = delta Q2 is already 0 and this instruction is executed only in the next delta, therefore Q1 becomes 0 because Q2 is 0 and indeed now we must wait till Q1 acquires this value that means

we must wait for the entire delta and now Q1 becomes 0 and finally at t = 2 delta, Q0 also becomes 0, because Q0 = Q1, Q1 is 0 and therefore at 3 delta all values are 0 in this case so it is not a shift register at all.

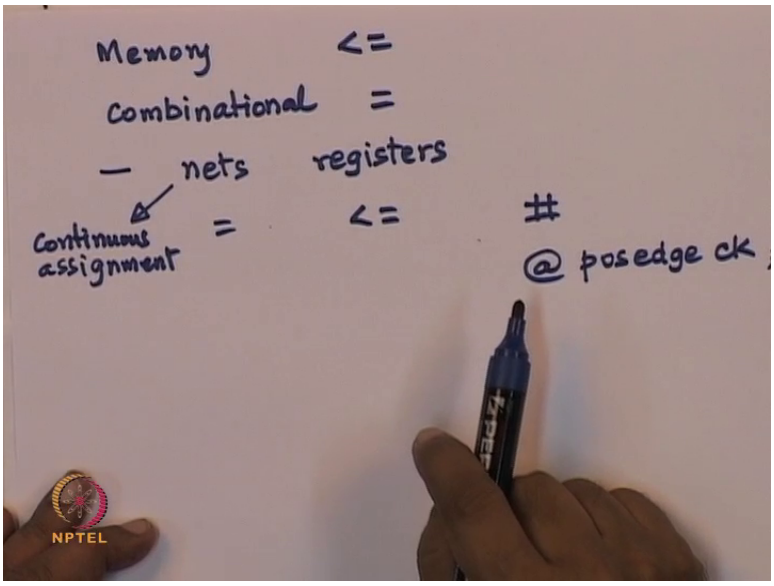
However in this case let us say that the first statement occurs at t = 0 in that case Q2 is scheduled to receive the value of In which is 0 at delta, however the time is still 0 and we do not wait for Q2 to acquire its new value we go to next statement in the current delta itself, therefore the next statement occurs at t = 0 itself not at t = delta, now when you say assign the current value of Q2 to Q1 the current value of Q2 is in fact 1 that means Q1 is scheduled to receive 1 at delta.

And then we proceed we place this transaction and we proceed we do not wait for Q1 to acquire this value and therefore the next one also takes place at t = 0 and now Q0 is scheduled to acquire the old value of Q1 which is 0 and then at delta all these values are assigned and therefore Q2 becomes 0, Q1 becomes 1, Q0 becomes 0, so essentially you have 0 1 0, Q2, Q1, Q0 acquire the new values at delta and that is correct.

Because you had 0 at the input and 1 0 1 in the shift register when you shift then you will get 0 1 0 and that is what we get, now this is all very well this is a very simple case how do we know when to use blocking and non-blocking assignments, so because the effects can be disastrous if you do not understand what the impact of this is and when it is appropriate to use blocking assignments or non-blocking assignments.

This whole business comes about because the blocking assignment which is traditionally used is actually not physical in the sense that hardware does acquire time and therefore if we are describing everything together what Verilog does is to shift the responsibility of properly interpreting this description to this synthesizer and therefore the simulation can sometimes lead you to difficulties, so now in general there is a thumb rule which is fine.

(Refer Slide Time: 48:11)



So essentially if there is memory involved then use non-blocking if there is combinational logic then you may use equal to, so this now introduces the new complication in Verilog which did not exist in VHDL, in VHDL we use the same kind of assignments throughout and in Verilog now we must determine whether the assignment is being made to an element with memory.

Now noticed that this memory is of different kind this is the proper register kind of memory, the latch kind of memory, whereas of course every register had a memory but assigning to register is not always with non-blocking, so there is this complication in Verilog, I think it is difficult to spend the entire language in one lecture, however as I said before the real understanding of the language will come when you practice it.

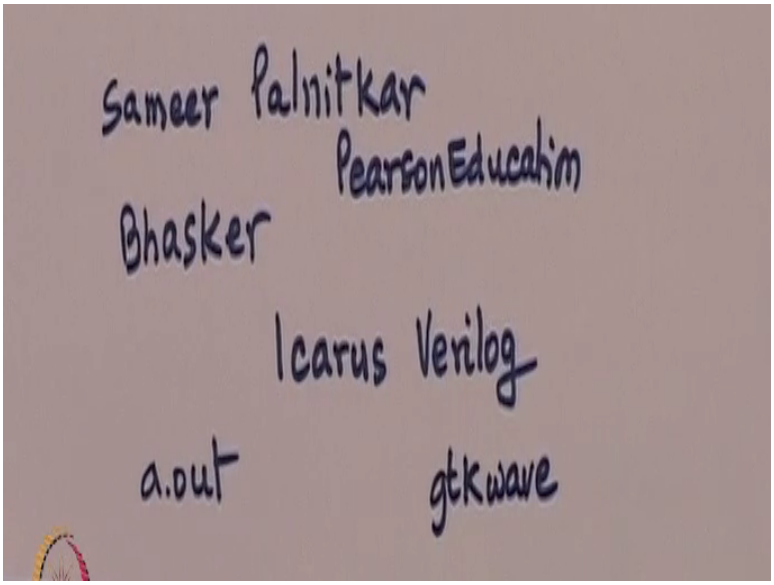
However these things are a bit hard to comprehend at the beginning and therefore I have picked out those things which require special understanding, so essentially what we had what you need to keep in mind is first of all the difference between nets and registers and then blocking and non-blocking assignments, that nets need continuous assignments, that registers assigned at a particular instant and that instant is determined either by the delay control if you do not give a delay then 0 is assumed or what I did not describe here in detail is an event control.

So for example you might say @ posedge clock this is called event control, so there is an implied instance at which every assignment is to be made in a sequential block in Verilog and that

implied instant could be if it is not given could be just as # 0 but # anything is essentially wait for this much time or wait till this event become true and then make the assignment.

Because the assignments are made at a sharp instant of time and then the node is supposed to hold that value for future, therefore that has to be a register time so it could be typically a reg. As I said you need to practice it and I would like to finish this lecture by giving you a kind of list of resources which you may find useful, again for Verilog there is a very large number of good books.

(Refer Slide Time: 51:32)



Just to name a few there is one by Sameer Palnitkar and another by A. Bhasker, I think the Sameer Palnitkar books is published by Pearson Education, Bhasker's book is called a Verilog Primer, in addition to that there is a public domain program which is excellent which is called Icarus Verilog its versions are available for Linux as well as Windows though it is largely meant for Linux kind of systems.

It is a public domain implementation if you do not have access to professional VLSI design software you can download and install it on any Linux installation, Icarus Verilog is a fairly complete implementation of the language and it works like GHDL in the sense that it produces an executable by default a dot out as is the common practice in UNIX, so it produces an

executable a dot out and when you execute this executable file then the simulation outputs are reported to you.

Icarus Verilog is then coupled with the same waveform viewers that we had discussed earlier and those waveform viewers are essentially VCD in VCD format which is the value dump format that Verilog has and there are many public domain programs for viewing this waveform a prominent one is the gtk wave I think with this we shall bring this lecture to a finish. We just had a general introduction to Verilog.

I would not call it a sufficient for you to even start writing modules in C, however excellent resources are available in C by the way an excellent learning module on Verilog is available from CEERI Pilani and this learning module was developed for the SMDP program a special manpower development program for VLSI design.

But it might be available in the public domain and this is an excellent step by step tutorial on how to learn Verilog by yourself in a lab that assumes the availability of let say cadence or other kind of VLSI design, synopsis, Verilog software, however it will work equally well with the Icarus Verilog and gtk wave combinations. What I suggest is that you acquire this learning aids an actually program using either GHDL in case of VHDL or Icarus Verilog in case of Verilog.

And then you will appreciate the concepts that we have discussed in these lectures well, we bring this lecture to an end here.