Advanced VLSI Design

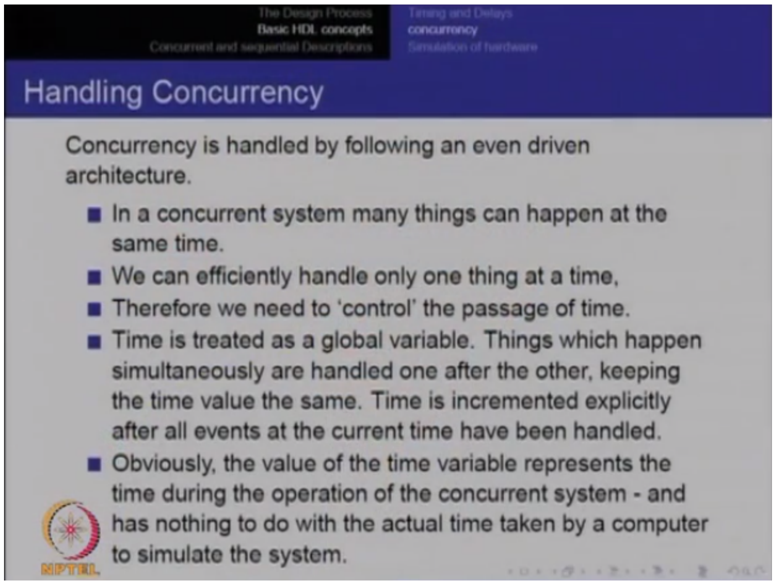
Prof. D. K. Sharma
Department of Electrical Engineering
Indian Institute of Technology - Bombay

Lecture - 18
Managing Concurrency and Time in Hardware Description Languages

In the previous lecture we have looked at hardware description languages in general. We have seen that in many respects these are similar to writing programs but in many very important aspects they are different. The overall structure and syntax of hardware description languages may look like programming languages, but we must always be aware that this is not something which is executing line by line.

In fact, it is describing part by part some piece of hardware. All parts of this hardware might be acting simultaneously and this is very different from the running of a program which executes line by line. This concurrency has to be handled in a special way and we shall quickly revise what we had discussed in the last lecture on how this concurrency is handled.

(Refer Slide Time: 01:26)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Handling Concurrency

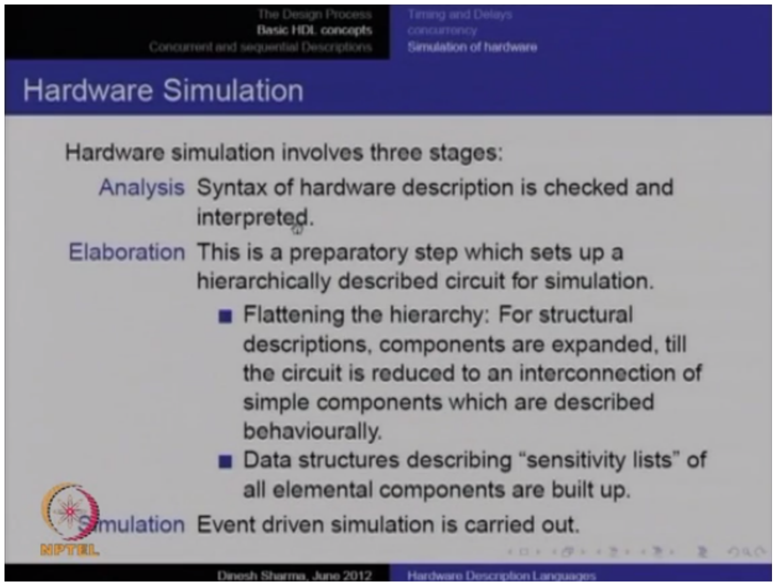
Concurrency is handled by following an even driven architecture.

- In a concurrent system many things can happen at the same time.
- We can efficiently handle only one thing at a time,
- Therefore we need to 'control' the passage of time.
- Time is treated as a global variable. Things which happen simultaneously are handled one after the other, keeping the time value the same. Time is incremented explicitly after all events at the current time have been handled.
- Obviously, the value of the time variable represents the time during the operation of the concurrent system - and has nothing to do with the actual time taken by a computer to simulate the system.

Concurrency is handled by following an event driven architecture. In a concurrent system many things can happen at the same time. We can efficiently handle only one thing at a time, therefore we need to control the passage of time which is then treated as a global variable. Even though during simulation the time is passing but as long as the value of this global variable remains constant in simulation time it appears as if everything is happening at the same time.

The time is incremented explicitly after all events at the current time have been handled and obviously the value of the time variable represents the time during the operation of the concurrent system and has nothing to do with the actual time taken by computer to simulate the system. You can have fast computers, you can have slow computers.

(Refer Slide Time: 02:26)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Hardware Simulation

Hardware simulation involves three stages:

Analysis Syntax of hardware description is checked and interpreted.

Elaboration This is a preparatory step which sets up a hierarchically described circuit for simulation.

- Flattening the hierarchy: For structural descriptions, components are expanded, till the circuit is reduced to an interconnection of simple components which are described behaviourally.
- Data structures describing "sensitivity lists" of all elemental components are built up.

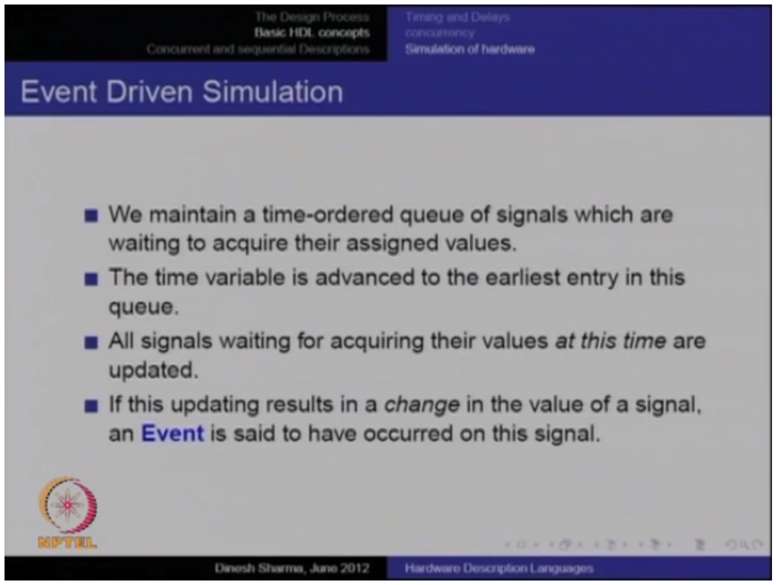
Simulation Event driven simulation is carried out.

Dinesh Sharma, June 2012 Hardware Description Languages

The hardware simulation involves three basic steps, the first step involves the analysis of the description that you have given, this includes checking the syntax etc. of the hardware description the, second part is elaboration. In this according to the description supplied by you the circuit is actually built up, this is a preparatory step and which sets up a hierarchically described circuit for simulation.

So there are two major tasks which are performed during elaboration, you flatten the hierarchy. For structural descriptions components are expanded till the circuit is reduced to an interconnection of simple components which are described behaviorally and data structures describing sensitivity lists of all elements. All elemental components are built up during this time and it is only after these two steps are complete that we begin with the simulation of a circuit.

(Refer Slide Time: 03:41)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

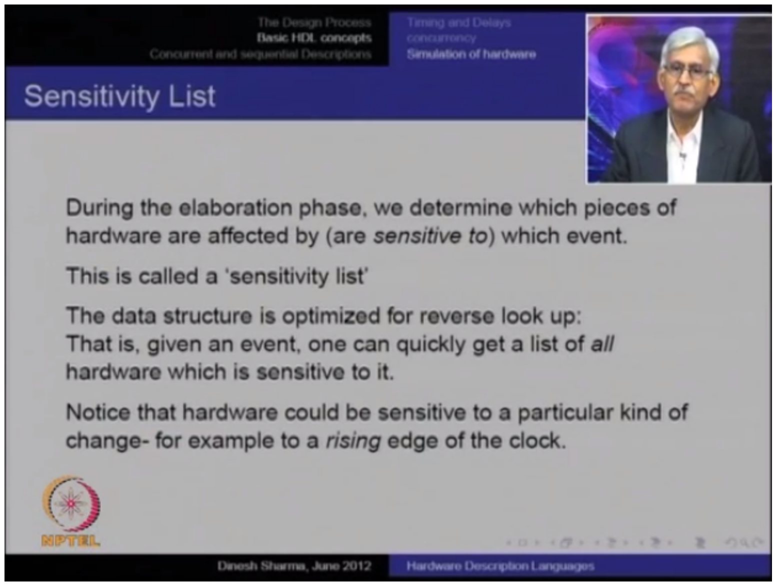
Event Driven Simulation

- We maintain a time-ordered queue of signals which are waiting to acquire their assigned values.
- The time variable is advanced to the earliest entry in this queue.
- All signals waiting for acquiring their values *at this time* are updated.
- If this updating results in a *change* in the value of a signal, an **Event** is said to have occurred on this signal.

Dinesh Sharma, June 2012
Hardware Description Languages

Simulation itself is built up of two separate steps. During simulation we carry out an event driven simulation. We maintain a time ordered queue of signals which are waiting to acquire their assigned values, the time variable is advanced to the earliest entry in this queue and all signals waiting for acquiring new values at this time are then updated. If this updating results in a change in the value of signal an event is said to have occurred on this signal.

(Refer Slide Time: 04:22)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Sensitivity List

During the elaboration phase, we determine which pieces of hardware are affected by (are *sensitive to*) which event.

This is called a 'sensitivity list'

The data structure is optimized for reverse look up:
That is, given an event, one can quickly get a list of *all* hardware which is sensitive to it.

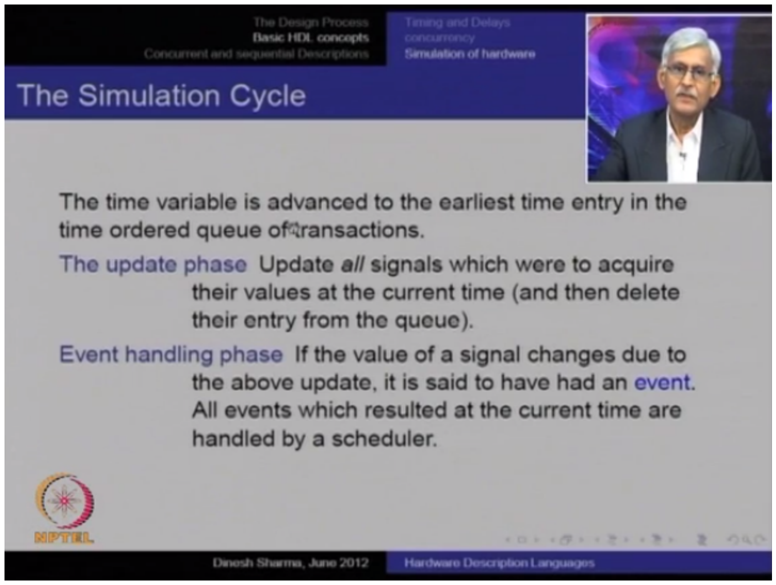
Notice that hardware could be sensitive to a particular kind of change- for example to a *rising* edge of the clock.

Dinesh Sharma, June 2012
Hardware Description Languages

During the elaboration phase we determine which pieces of hardware are effected by or are sensitive to which event, this is called a sensitivity list. The data structure is optimized for reverse lookup when we build the sensitivity list, we go piece of hardware to piece of hardware and find out to which events each piece of hardware is sensitive; however, we arrange this list in such a way that we scan it event by event.

When an event has occurred we can directly go and look up which are the pieces of hardware which are sensitive to the seat. Notice that hardware could be sensitive to a particular kind of change for example a flip-flop could be sensitive to a rising edge and not a falling edge.

(Refer Slide Time: 05:21)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware
NPTEL
Dinesh Sharma, June 2012
Hardware Description Languages

The Simulation Cycle

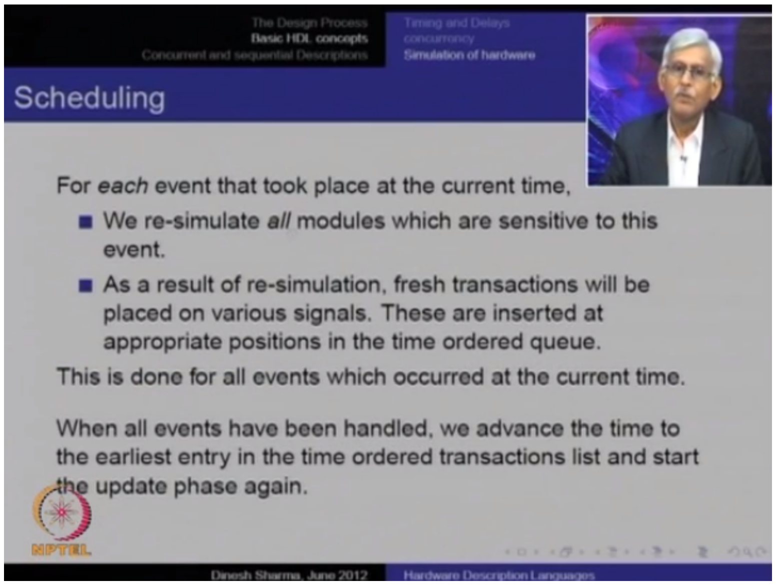
The time variable is advanced to the earliest time entry in the time ordered queue of transactions.

The update phase Update *all* signals which were to acquire their values at the current time (and then delete their entry from the queue).

Event handling phase If the value of a signal changes due to the above update, it is said to have had an event. All events which resulted at the current time are handled by a scheduler.

The simulation proceeds in two cycles, the update phase during which all signals which were to acquire their values at the current time are updated and then these entries are deleted from the time sensitive queue. Only when the update phase is complete do we go to the event handling face. While updating, we make a note of which signals changed and therefore an event occurred on them and then we take event by event and for each event we find out which hardware is sensitive to it and re-simulate only that part of the hardware.

(Refer Slide Time: 06:08)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Scheduling

For *each* event that took place at the current time,

- We re-simulate *all* modules which are sensitive to this event.
- As a result of re-simulation, fresh transactions will be placed on various signals. These are inserted at appropriate positions in the time ordered queue.

This is done for all events which occurred at the current time.

When all events have been handled, we advance the time to the earliest entry in the time ordered transactions list and start the update phase again.

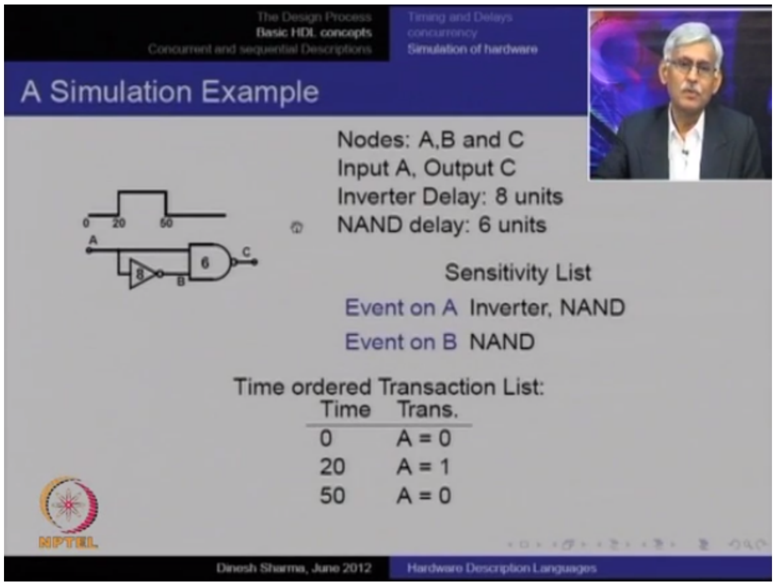
Dinesh Sharma, June 2012 Hardware Description Languages

For each event that took place at the current time, we simulate all modules which are sensitive to this event. As a result of re-simulation fresh transactions will be placed in various signals, because the inputs have changed. On re-simulation these pieces of hardware will say that my output should change and that will of course occur at a future time. So these are inserted at appropriate positions in the time ordered queue and then this is done for all events which occurred at the current time.

Notice that the events are handled at the current time, their results will show up at a future time. If the delay is given explicitly then it will occur at a physically different time but even if zero delay is specified, the results will show up delta times later which will be eventually reported as zero. However, for sequencing purposes they will appear to occur at different time events separated by delta times.

When all events have been handled we advance the time to the earliest entry in the time ordered transactions list and start the update phase again so we keep alternating between update phase and the event handling phase.

(Refer Slide Time: 07:44)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

Nodes: A,B and C
Input A, Output C
Inverter Delay: 8 units
NAND delay: 6 units

Sensitivity List

Event on A	Inverter, NAND
Event on B	NAND

Time ordered Transaction List:

Time	Trans.
0	A = 0
20	A = 1
50	A = 0

NPTEL

Dinesh Sharma, June 2012 Hardware Description Languages

We might want to look at it really using a very simple example. This will help in cementing these processes in our mind, so let us take a very simple circuit, we have a NAND gate which receives a signal as well as its inverse so the input is at A and this input goes directly to a NAND gate, this input is also inverted and the inverted signal appears at B. The NAND of both of these will then appear at the outputs C. We intend to apply an input which is zero at zero time.

At time 20 it will have a transition to the value one and then it will remain at 1 till 50. At 50 units of time it will come down to 0 again and remain zero afterwards. So let us take this very simple circuit, you will recognize this circuit as one which produces a glitch and we want to see whether the mechanism that we have described will be able to correctly predict this glitch. We assume that the inverter introduces 8 units of delay and the NAND introduces 6 units of delay.

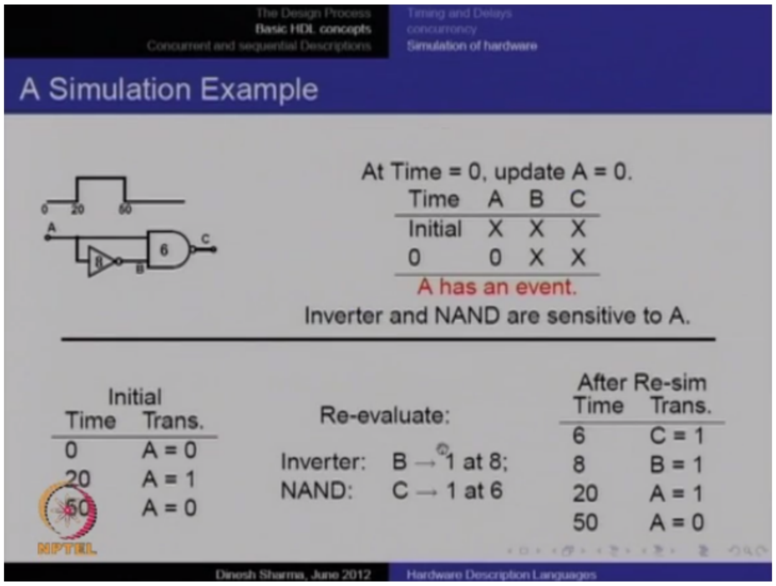
As we build up this circuit we find that this NAND is sensitive to A as well as to B. On the other hand, the inverter is sensitive only to inputs at B at A and B in fact is its output. When we invert this list as I had described earlier we now go event wise rather than hardware wise and then we say that events at A require re-simulation of the inverter as well as a NAND. Whereas events at B require the re-simulation only of the NAND.

And the C as an output it affects neither the inverter nor the event. We initialize this transaction list corresponding to the input being specified by us. We specify that at times 0, A will be 0. At time 20, A will go to 1 and at time 50, A will come back to 0 again. So this is the initial time ordered transactions list. Notice that actually this is the only case in which case this whole list is treated as if it is a transport delay that means no pulse specified at this time will vanish otherwise we will not be able to apply fast pulses to any circuit.

So the transaction list is initialized to a list and in this case it is a simple list and at this time it has only three entries one at 0, the other at 20 and the third at 50. Initially all inputs are supposed to be undefined so given that situation let us go to the least time in this sensitivity list, in this time ordered list. So if you look at the sensitivity level list at this time, what is the least time 0, so we make our time variable equal to 0 and what is supposed to happen at time equal to 0.

Well, A is supposed to become 0, now remember the initial value of A was undefined. From undefined to 0 is a value change, notice that changes need not be only from 0 to 1 or 1 to 0. It was initially undefined; its value has now become 0.

(Refer Slide Time: 12:26)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

At Time = 0, update A = 0.

Time	A	B	C
Initial	X	X	X
0	0	X	X

A has an event.

Inverter and NAND are sensitive to A.

Initial

Time	Trans.
0	A = 0
20	A = 1
50	A = 0

Re-evaluate:

Inverter: B → 1 at 8;
NAND: C → 1 at 6

After Re-sim

Time	Trans.
6	C = 1
8	B = 1
20	A = 1
50	A = 0

Dinesh Sharma, June 2012 Hardware Description Languages

So let us see what happens at zero time and track this format that I have for the development of the circuit. We have the circuit and the waveform at the input here, what action takes place at this time will be here and we shall see the initial state of the time ordered queue here and the final state of the time ordered queue here.

So as we know the initial state of the time ordered queue was these three entries. At 0, A was to become 0; at 20, A was to become 1 and at 50, A is to become 0 again, that is all we know at this stage. Because zero is the minimum time we make the time equal to 0 and therefore we update this, we are in the update phase right now. So the only required is to make A equal to 0 that means initially A was undefined, B was undefined, C was undefined.

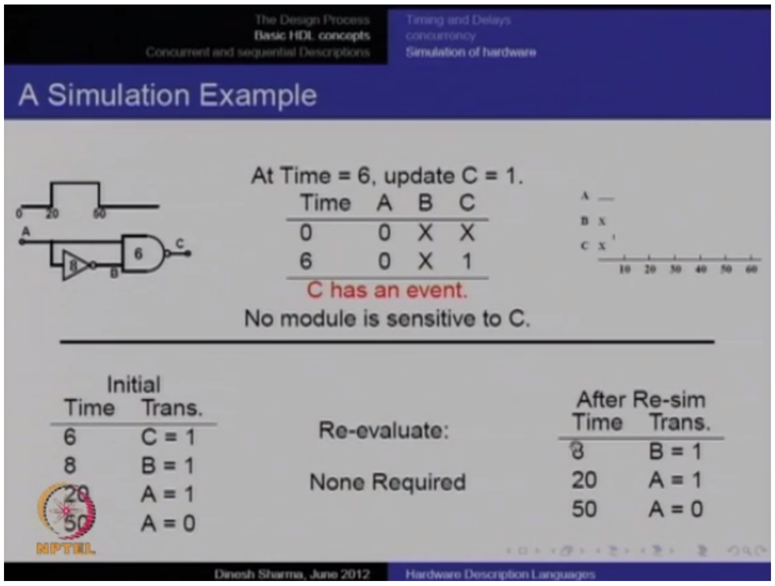
Now at time 0, A has become 0. B and C are still undefined. We do not know what their values are, therefore we find that A has had an event. We go around and find out who is sensitive to A and as you will recall both the inverter as well as the NAND are sensitive to A therefore we must re-evaluate the outputs of the inverter as well as the NAND, so we go to the inverter and tell it that look your input just became zero what should your output be.

The inverter says that please if my input has just become zero and the current time is zero then please make my output B equal to 1 at 8 units of time because its delay is 8. Similarly, we go the NAND and tell it look the current tiny is zero and A just became zero, so the NAND looks at the inputs and says oh if A has become zero please make my outputs C 1 because this is a NAND and a zero at either at 1 input is sufficient to determine its outputs.

So it says please make my output 1 at 6 units of time, that means these two new transactions must be entered in a time ordered queue and now because we have handled this event at zero this should be removed so when you remove this and add these 2, our time ordered queue looks somewhat like this. At 6 units of time makes equal to 1 that is this transaction and because this is at an earlier time it will be the first entry.

Then at 8 make B equal to 1 and the remaining two entries at 20 and 50 remain as such we have not handled it. Now what is the earliest entry in our time ordered queue, the earliest entry is for a time equal to 6 units of time. So we advance the time variable to 6 units, skip straight from 0 to 6 because nothing happens in between therefore we go to a time which is 6.

(Refer Slide Time: 16:08)



A Simulation Example

At Time = 6, update C = 1.

Time	A	B	C
0	0	X	X
6	0	X	1

C has an event.

No module is sensitive to C.

Initial	
Time	Trans.
6	C = 1
8	B = 1
20	A = 1
50	A = 0

Re-evaluate:

None Required

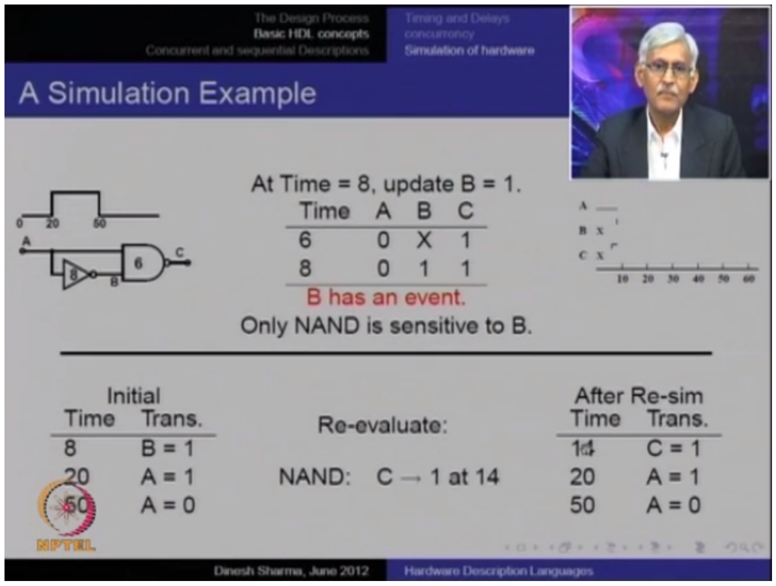
After Re-sim	
Time	Trans.
8	B = 1
20	A = 1
50	A = 0

So we make the time equal to 6. What is waiting to happen at 6, well at 6, C should become 1, so we make C equal to 1, remember after the update at equal to 0, A was 0, but B and C were unknown. Now the time is 6, A remains 0, but C should become 1 and therefore C has had an event, its value has changed, nothing else has changed.

C has had an event but nobody is sensitive to C, C is just the output, so all we do is that we make C equal to 1 and do not have to do any re-simulations, so therefore we make C equal to 1 and remove this entry from the time ordered queue, no new entries are entered at this time. Therefore, at the end of it our time ordered queue looks like this, the entry for 6 has vanished, we still have an entry for 8 and we have the old entries at 20 and 50.

I am making graft, a diagram of the evolution of A, B and C here so A is now at zero and C has become 1 from X at 6 units of time. Since 6 units of time has now been handled we have removed this entry at 6. This is the remainder of the time ordered queue, the least time on this time ordered queue is eight units of time, so we advance the time variable to 8 units of time and let us see what happens when we go to 8 units of time.

(Refer Slide Time: 18:15)



A Simulation Example

At Time = 8, update B = 1.

Time	A	B	C
6	0	X	1
8	0	1	1

B has an event.
Only NAND is sensitive to B.

Initial

Time	Trans.
8	B = 1
20	A = 1
50	A = 0

Re-evaluate:

NAND: C → 1 at 14

After Re-sim

Time	Trans.
14	C = 1
20	A = 1
50	A = 0

The earlier time was 6 and at the end of the transaction A was 0 and C had become 1, B was still undefined. At 8 units of time what are we waiting for, we want to make B equal to 1, therefore we make B equal to 1 and we notice that B has had an event, it was earlier X, it has now become 1. So who is sensitive to B because B has had an event we must go around looking for those pieces of hardware which care that the value of B has changed.

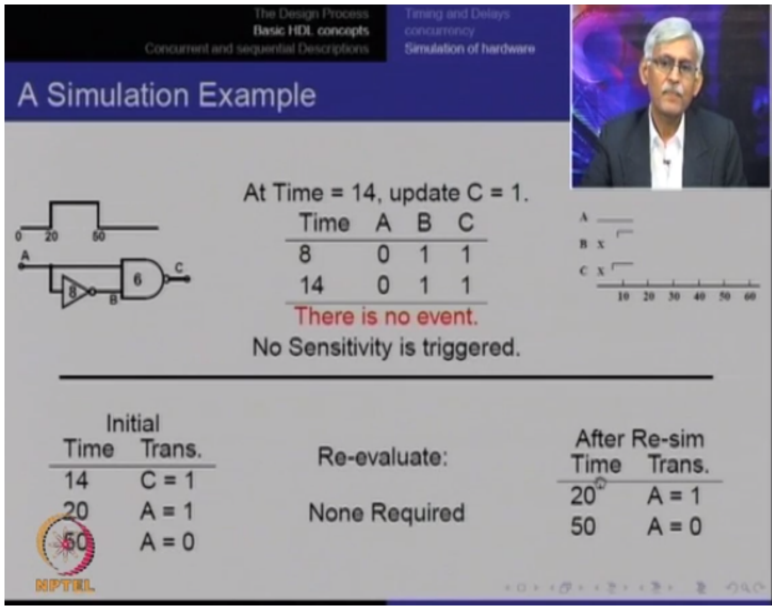
And we find the only NAND is sensitive to B, the inverter is not sensitive to B, the inverter has its output at B. So therefore we have to re-evaluate only the NAND. So we go and tell the NAND that look the value of the just became 1 at the current time which is 8. So this is the story that we report to the NAND. The NAND knows that its other input is 0, so it computes its output with these new inputs and it says oh A is 0 and B has just become 1.

Then please make my value C, my outputs C 1 at 14 units of time. Notice that the value of C is already 1, but that does not matter, it will place a transaction on C now at 14 and say that my value should be 1 at 14 units of time because its delays 6 and the current time is 8. So therefore the output will appear at 14 units of time and the output which will appear will correspond to A equal to 0, B equal to 1, therefore C should be 1.

So this is the transaction that we enter in a time ordered queue, this is a new transaction which has gone into our time ordered queue and this transaction is now discarded because it has been handles. So, therefore the new entry is at 14, makes C equal to 1. We now handled the time equal to 8, nothing else remains to be done at 8 and therefore we advance the time variable to the least time in this time ordered queue.

This happens to be 14, so we make the time equal to 14 and let us see what happens at T equal to 14.

(Refer Slide Time: 21:05)



A Simulation Example

At Time = 14, update C = 1.

Time	A	B	C
8	0	1	1
14	0	1	1

There is no event.

No Sensitivity is triggered.

Initial

Time	Trans.
14	C = 1
20	A = 1
50	A = 0

Re-evaluate:

None Required

After Re-sim

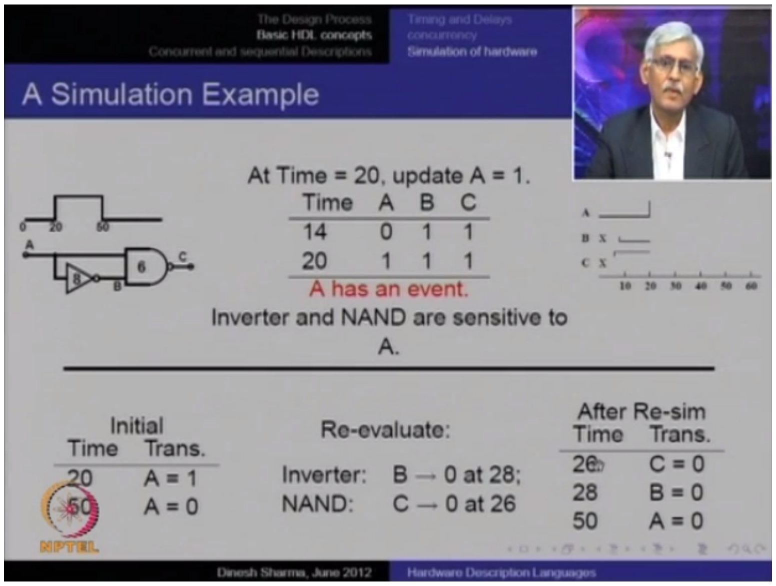
Time	Trans.
20	A = 1
50	A = 0

The previous time was 8. At that time A was zero, B was 1 and C was 1. At 14 we have a transaction which says please make C equal to 1 but C was already 1, so therefore there is no event, even though the transaction has fructified it has placed a value 1 at C but the old value of C itself was 1, therefore even though there is an update there is no change in value and therefore there is no event.

No sensitivity is triggered and therefore no re-evaluation is required, so essentially we say that alright we know that we were supposed to make C equal to 1, C is already 1, no changes involved, no re-evaluation are required and therefore we just removed this transaction. Now we are left with only 2 transactions, 1 at 20, the other at 50. Watch this place because that is where we are plotting the evolution.

A became 0 at 0. The current time is 14 and from 0 to 14, A has remained at 0 that is what this line shows. B became 1 at 8 units of time so from unknown it went to 1 and has remained 1 till this time. C became 1 at 6 units of time and at 14, we have decided that it will remain at 1, no event has taken place so we just advanced the time to the next entry in a time ordered queue, this entry is 20.

(Refer Slide Time: 23:09)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

At Time = 20, update A = 1.

Time	A	B	C
14	0	1	1
20	1	1	1

A has an event.
Inverter and NAND are sensitive to A.

Initial

Time	Trans.
20	A = 1
50	A = 0

Re-evaluate:
Inverter: B → 0 at 28;
NAND: C → 0 at 26

After Re-sim

Time	Trans.
26	C = 0
28	B = 0
50	A = 0

Dinesh Sharma, June 2012
Hardware Description Languages

So we advance the time to 20, the previous values of A were 0, 1 and 1 as we have just seen. Now the time has become 20 and what happens at 20 well the value of A at 20 goes from 0 to 1, the other 2 values are the same, but A has had an event, its value has changed from 0 to 1 so we go around asking as before who cares that A has had an event and because A is the input of both the inverter as well as the NAND.

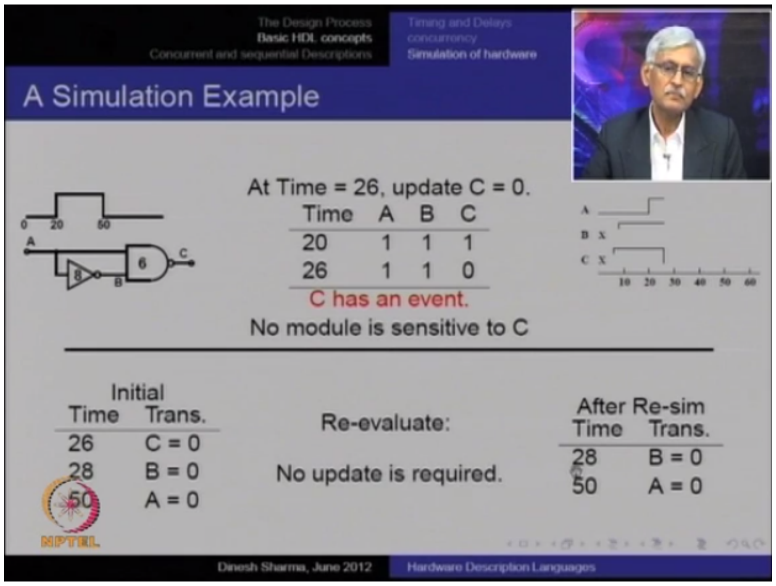
Therefore, both the inverter and the NAND must be re-simulated, both are sensitive to events at A. So at 20, we made equal to 1, this has resulted in an event on A, both the inverter and the NAND are sensitive to A so we simulate both; so we go to the inverter and say the current time is 20, A just became 1, what is you output. So the inverter says the current time is 20 and my input is 1.

Then, please make my output B 0 at 28 units of time because it has a delay of 8, the current time is 20. You go to the NAND and say that A just became 1, the current time is 20, what should your output B. The NAND looks at A as well as B, both being 1 and says please make my outputs C 0 at 26 units of time. So these two transactions are inserted in the time ordered queue, they are ordered by time.

And therefore this entry at 26 is the first entry which says please make C equal to 0, it came from the NAND. And then the next entry is at 28, it came from the inverter and it says please make B equal to 0 at this time and with that we have taken care of the event at 20, A equal to 1, so we delete that and now we are left with a time ordered queue of transactions which looks like this and the earliest entry is at a time equal to 26.

So we say that please make the time variable equal to 26 now. So we advance the time variable to 26 and let us see what happens at 26 units of time.

(Refer Slide Time: 25:55)



A Simulation Example

At Time = 26, update C = 0.

Time	A	B	C
20	1	1	1
26	1	1	0

C has an event.

No module is sensitive to C

Initial

Time	Trans.
26	C = 0
28	B = 0
50	A = 0

Re-evaluate:

No update is required.

After Re-sim

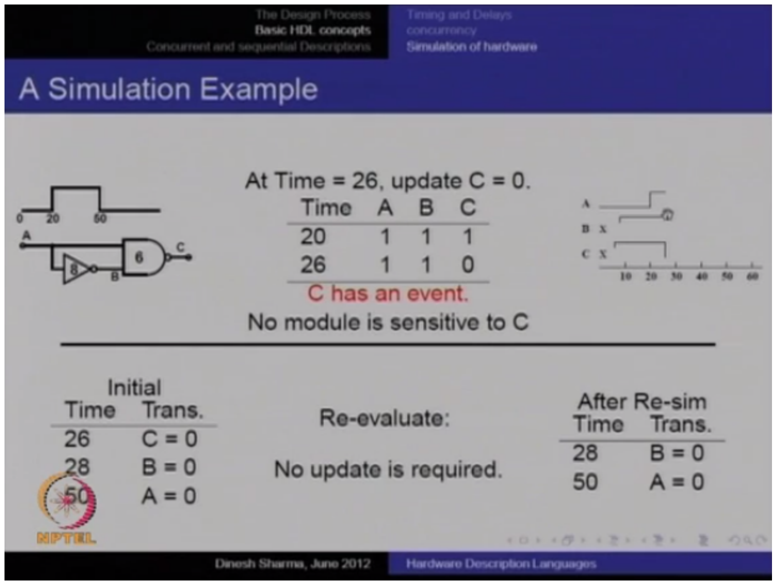
Time	Trans.
28	B = 0
50	A = 0

At 26 units of time, remember at 20, A had become 1, B was still at 1, it had not acquired its value of 0, there is a transaction waiting for it to become 0 but B was still at 1, C was also at 1, and the waiting transaction at 26 says makes C equal to 0, so the old value of C was 1, the new value of C is 0, therefore C has had an event; however, no module is sensitive to C, it is not the input of anybody.

Therefore, no re-evaluation is required, no update is required, therefore we just make C equal to 0 and remove this entry. So now A has become 1 at this time at 20, we are currently 26 and 26 C has become 0, B is still awaiting to be updated because the delay of the inverter was 8 and it is scheduled to become 0 only at 28. No update is required anymore because nobody is sensitive to C and therefore we simply remove this having made C equal to 0.

Therefore, we are left with a time ordered queue which looks like this, says at 28 make B equal to 0, at 50 make A equal to 0. So we advance the time to the lowest entry in this time ordered queue and now we go to the next time which is 28.

(Refer Slide Time: 27:34)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

At Time = 26, update C = 0.

Time	A	B	C
20	1	1	1
26	1	1	0

C has an event.
No module is sensitive to C

Initial

Time	Trans.
26	C = 0
28	B = 0
50	A = 0

Re-evaluate:

No update is required.

After Re-sim

Time	Trans.
28	B = 0
50	A = 0

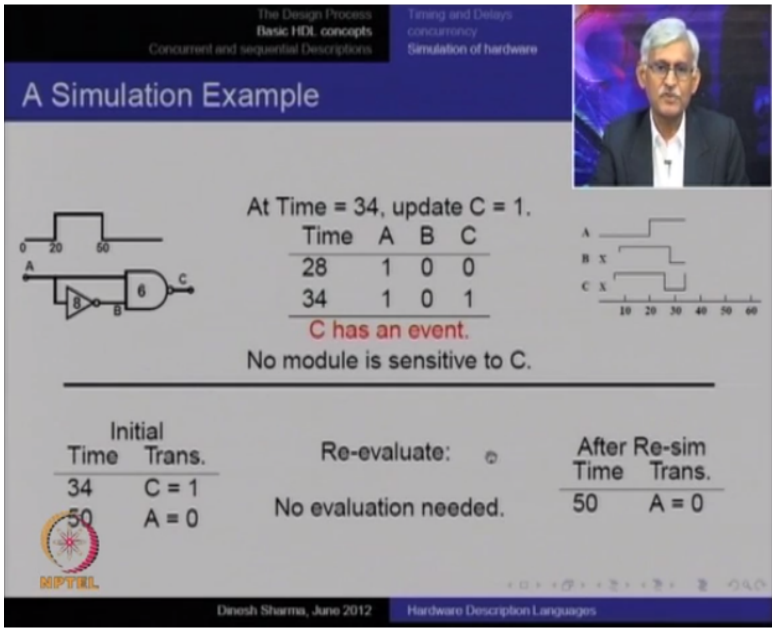
Dinesh Sharma, June 2012
Hardware Description Languages

So we have advanced the time from 26 to 28. A has remained at 1 and at this time B acquired its new value, it goes from 1 to 0, C remains at 0, so therefore the waiting transaction at 28 is make B equal to 0 and this results in a value change for B, B goes from 1 to 0 and therefore has an event so this is the event that I am talking about. So B has had an event, who is only sensitive to B, only the NAND is sensitive to B.

Therefore, we must re-simulate the NAND, so we go to the NAND and say that B just became 0 at a time of 28. The NAND has a delay of 6 so it says if B became zero at 28, please make my output 1 at 34 units of time. The current time is 28, B just became 0, the output was recompleted to be 1, but this value should occur at 28 plus 6 equal to 34, so this event is then inserted in our time ordered queue and this entry having been handled is removed.

So therefore this entry goes away and this entry comes in and our time ordered queue now looks like this. At 34 makes C equal to 1, at 50 make A equal to 0, so therefore this is the least entry in our time ordered queue and we advance the time to 34.

(Refer Slide Time: 29:27)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

At Time = 34, update C = 1.

Time	A	B	C
28	1	0	0
34	1	0	1

C has an event.
No module is sensitive to C.

Initial

Time	Trans.
34	C = 1
50	A = 0

Re-evaluate:
No evaluation needed.

After Re-sim

Time	Trans.
50	A = 0

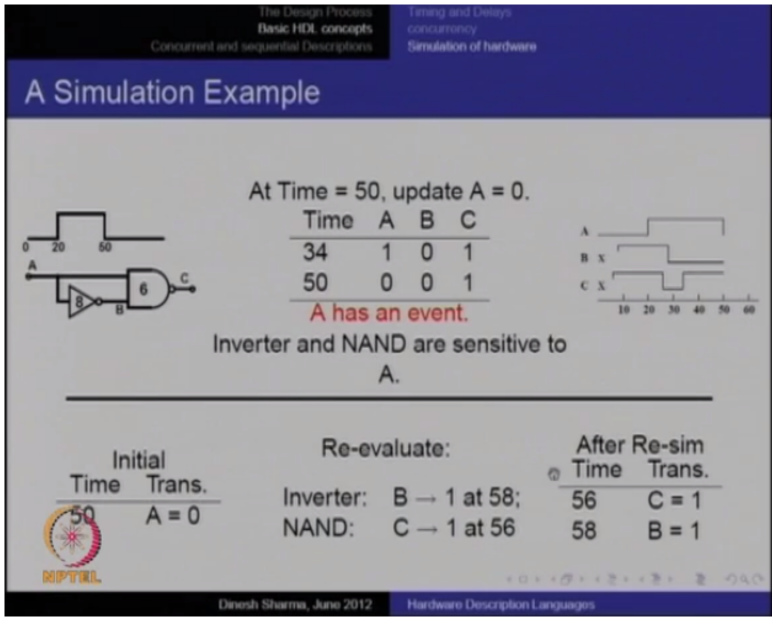
Dinesh Sharma, June 2012
Hardware Description Languages

So this is what happens at 28, A was 1; B was 0; C was zero. At 34, we have a waiting transaction which says please make C equal to 1, so we will make C equal to 1. We noticed that C value has changed, it has had an event but no module is sensitive to C, it is not the input of any module, therefore no re-evaluation is needed and therefore nothing is inserted into the time ordered queue.

We simply removed this transaction, C has become 1 now, noticed that this is the evolving wave shape. A became from 0 to 1 at 20, it resulted in B becoming 1 to 0 at 28, it resulted in C becoming 1 to 0 at 26 and now at 28 this change in B has caused this change in C, so the sea at 34 has now become 1, but this change in C does not require anybody to be re-evaluated and we are left with just one entry in a time ordered queue which says please make A equal to 0.

So we advance our time finally to 50 and this is what happens.

(Refer Slide Time: 30:48)



A Simulation Example

At Time = 50, update A = 0.

Time	A	B	C
34	1	0	1
50	0	0	1

A has an event.

Inverter and NAND are sensitive to A.

Initial

Time	Trans.
50	A = 0

Re-evaluate:

Inverter: B → 1 at 58;

NAND: C → 1 at 56

After Re-sim

Time	Trans.
56	C = 1
58	B = 1

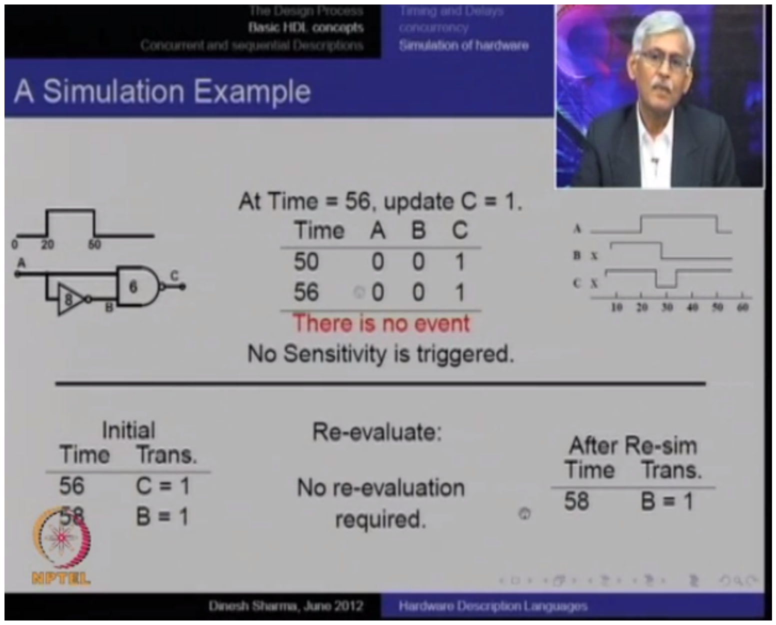
Dinesh Sharma, June 2012 Hardware Description Languages

At 50 we say please make equal to 0 so noticed that at 50 A will return to 0, the old values of A, B and C were 1, 0 and 1 and now A has become 0, so A has had an event. Since A is in input both to the inverter and the NAND, we must we re-simulate both the inverter and the NAND. So we go to the inverter and say that the current time is 50, A just became 0, what should we do.

The inverter says the current time is 50 then please make my output B 1 at 58 units of time. This is an inverter the input just became 0, the output must become 1 and because of the delay it must become 1 at 58 units of time. We also go to the NAND and tell it that look A just became 0 and the current time is 50, the NAND says that please make my output 1 because one of the inputs has become 0.

So please make my output 1 but at 56 units of time, so we insert both these transactions into the waiting transaction queue in time ordered. The first entry is at 56 and it says please make C equal to 1, the next entry is at 58 saying please make B equal to 1 and we have already handled this entry at 50 equal to 0 so this is deleted and therefore our time ordered queue looks like this. At 50 units of time the earliest entry is at 56 so we just advance the time variable to 56.

(Refer Slide Time: 32:42)



A Simulation Example

At Time = 56, update C = 1.

Time	A	B	C
50	0	0	1
56	0	0	1

There is no event

No Sensitivity is triggered.

Initial

Time	Trans.
56	C = 1
58	B = 1

Re-evaluate:

No re-evaluation required.

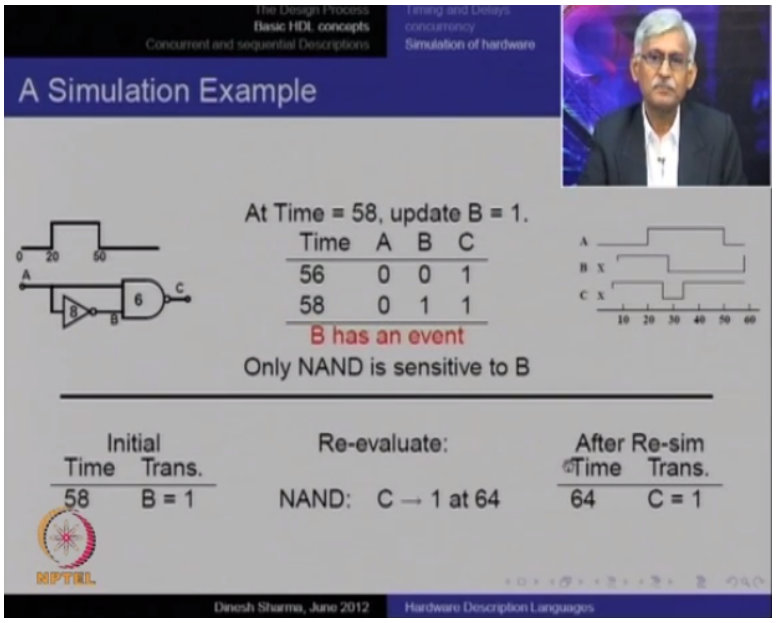
After Re-sim

Time	Trans.
58	B = 1

So what happens at 56, well at 50 both A and B were 0 and C was 1. The waiting event at 56 makes C equal to 1, but C is in fact already at 1, therefore there is no event and therefore no sensitivity is triggered, no re-evaluation is required. We just note that C was 1 and remains 1 and advance the time that means we can now drop this entry which says at 56 makes C equal to 1, C is already 1.

No re-evaluation is done and therefore we just dropped that entry and this is all that is left in our time ordered queue. At 58 B is to be made 1. The current time remember is 56. Since nothing else needs to be done till 58 we just advance the time variable to 58 and this is what happens.

(Refer Slide Time: 33:38)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

A Simulation Example

At Time = 58, update B = 1.

Time	A	B	C
56	0	0	1
58	0	1	1

B has an event

Only NAND is sensitive to B

Initial

Time	Trans.
58	B = 1

Re-evaluate:

NAND: C → 1 at 64

After Re-sim

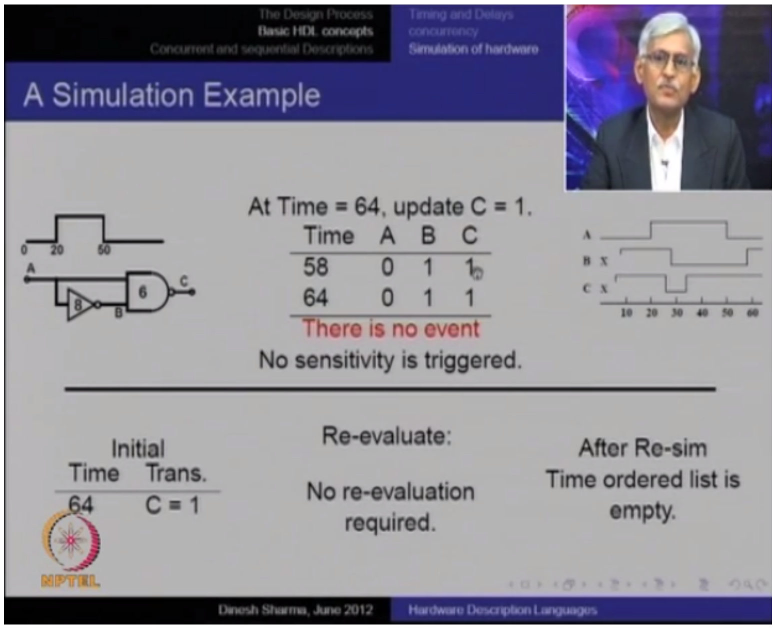
Time	Trans.
64	C = 1

Dinesh Sharma, June 2012 Hardware Description Languages

The previous state was at 56 and A, B and C were 0, 0 and 1. The waiting transaction says B equal to 1 at 58, the time has advanced to 58 and we are in the update phase. In update phase we actually make B equal to 1. We noticed that the value of B has changed, it has had an event, who cares if there is an event on B, only the NAND cares whose input Be is. Therefore, we go to the NAND and we say that the current time is 58 and B just became 1.

Notice that the value of A is 0. So NAND says oh if the current time is 58 and A is 0, B is 1, please make my output 1 at 64. Notice that C is already 1 but still replaces transaction, so we place a transaction at 64 saying please make C equal to 1 and then we drop this transaction. As a result, we are still left with one transaction to do in the time ordered queue. Therefore, we advance the time to 64 and make the value of C equal to 1 and let see what happens at this time.

(Refer Slide Time: 35:08)



A Simulation Example

At Time = 64, update C = 1.

Time	A	B	C
58	0	1	1
64	0	1	1

There is no event

No sensitivity is triggered.

Initial	
Time	Trans.
64	C = 1

Re-evaluate:

No re-evaluation required.

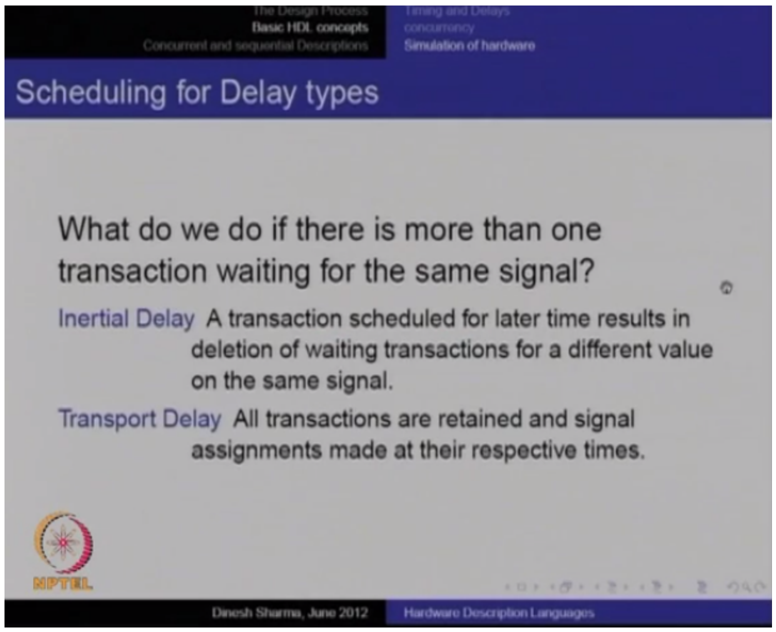
After Re-sim

Time ordered list is empty.

As we have seen the value of C was already 1, when we obtained the value of C we updated to the same value 1, therefore there is no event, no sensitivity is triggered, nothing needs to be re-evaluated and now our time ordered list is empty, nothing remains to be done. C remains at 1. As a result, we noticed that we have correctly simulated this glitch which appears in C and which occurs at the rising edge of A.

At the falling edge of B nothing happens to C. This is indeed what you would expect in a real hardware and our system of update re-simulate and place transactions has successfully recreated the scenario in simulation, so therefore this shows that this is how we proceed in our simulations.

(Refer Slide Time: 36:17)



The Design Process | Timing and Delays
Basic HDL concepts | concurrency
Concurrent and sequential Descriptions | Simulation of hardware

Scheduling for Delay types

What do we do if there is more than one transaction waiting for the same signal?

Inertial Delay A transaction scheduled for later time results in deletion of waiting transactions for a different value on the same signal.

Transport Delay All transactions are retained and signal assignments made at their respective times.

Dinesh Sharma, June 2012 | Hardware Description Languages

This was a particularly simple example in which there were very few transactions which were waiting at a particular time but what happens if for one signal more than one transaction is pending. In the current example the delays were carefully adjusted such that this complication did not occur. So it might happen that your simulation is such that more than one transaction is waiting to occur for the same signal.

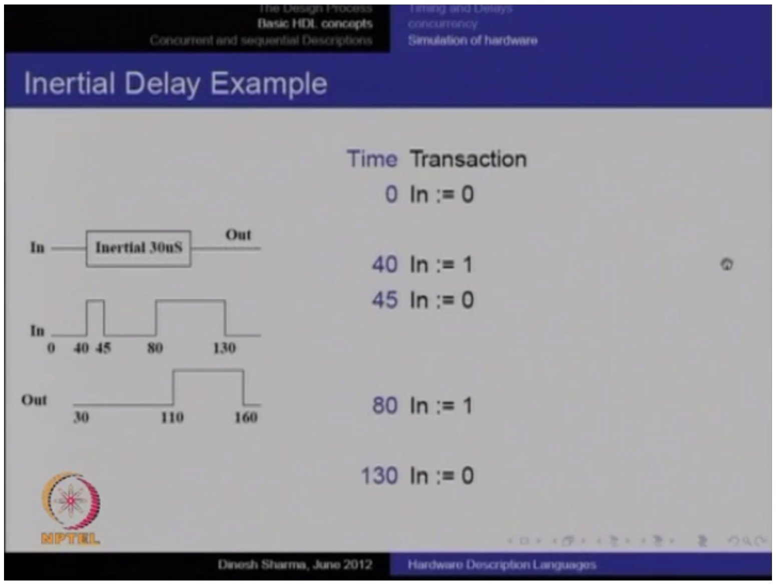
Let us look at this example and see how we handle this particular behavior in our simulations. In fact, we take our old example of delay of 30 microseconds which we had done in the last lecture and then this is a rule that we follow. If the delay is inertial, a transaction schedule for later time and for a different value results in deletion of waiting transactions which were for a different value.

If the delay is transport, then all pending transactions for the same signal are retained and signal assignments are made at their respective times as we did in the previous example. So therefore what we do with waiting transactions when multiple transactions are waiting for the same signal depends on what kind of delays were involved. If inertial delays are involved and the transactions are of 4 different values then all earlier transactions which are for a later time they are deleted.

In case of transport delay or transactions are retained okay so the later time transaction whenever it comes it results in deletion of waiting transactions which are for a different value on the same thing. This is the rule that we follow and let us apply this rule and see whether it stands true for the example that we had done in case of the RC delay and the fibre optical kind of system delay.

So we take that same example and look at the kind of delays that we have and see whether the narrow pulse will indeed vanish in case of inertial delay. The transport delay keeps all transactions therefore it is clear that all pulses will remain at the output. The interesting case is for the inertial delay and notice whether it successfully deletes the narrow pulse and keeps the white pulse. So let us look at this example.

(Refer Slide Time: 39:27)



Inertial Delay Example

Time	Transaction
0	In := 0
40	In := 1
45	In := 0
80	In := 1
130	In := 0

This is the circuit that we had discussed earlier if you recall. We have a module which has an inertial delay of 30 microseconds. We apply an input which is much smaller than 30 microseconds and one which is much longer than 30 microseconds. Here we have taken times where the narrow pulse is 5 microseconds wide whereas the wide pulse is 50 microseconds wide. Notice that from our earlier arguments, we expect the narrow pulse to vanish.

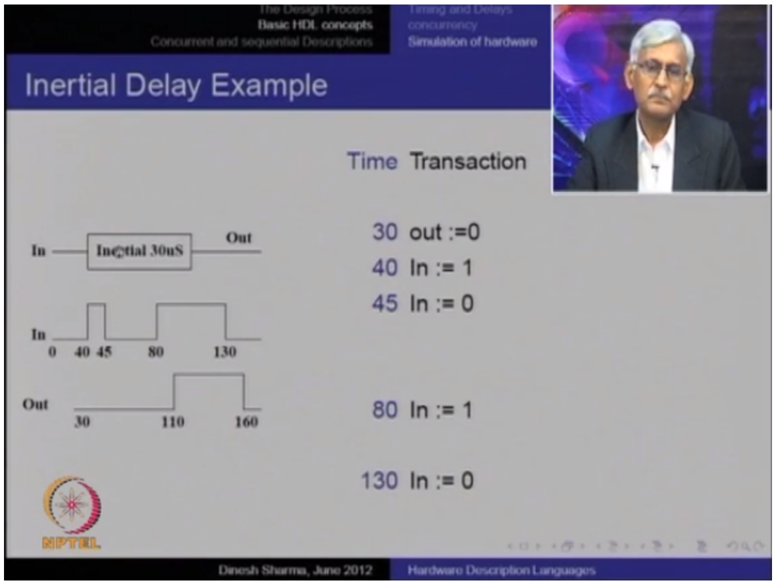
And we expect the wider pulse to appear delayed by 30 microseconds that means the pulse which lasted from 80 to 130 should now appear from 110 to 160 and the narrow pulse should vanish. Let us carry out the process exactly what we did earlier. In this case the circuit is even simpler, there is only 1 unit and there is only one input signal so the unit is sensitive to the input. We do not have to decide whether it is sensitive or not or whatever there is only one unit to simulate.

And every time the input changes it must be re-simulated, so we can go a little quicker over this simulation. So how is the time ordered queue populated initially. Well if you look at this input waveform you will notice that these are the major points from unknown it became 0 at 0. At 40 units of time it goes from 0 to 1. At 45 units of time it goes from 1 to 0. Then it remains at 0 till 80. At 80 it goes to 1, remains at 1 till 130 and at 130 it comes down to 0 and then remains at 0.

Therefore, all changes in the value of the signal must be inserted in our time ordered queue, so the first is from unknown to 0 so we say at 0 make input equal to 0. Next we say at 40 make input equal to 1, next we say at 45 make input equal to 0, next we say at 80 make input equal to 1, next at 130 we say make input equal to 0, so this describes the input waveform and that constitutes the initial time ordered queue of transactions.

So we start with this transaction list and proceed exactly as we did in the example before. We proceed to a time which is earliest in our time ordered queue. So we make our time variable 0 and make input equal to 0. There is an event on input, therefore we ask our delay module what should its output be.

(Refer Slide Time: 43:03)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

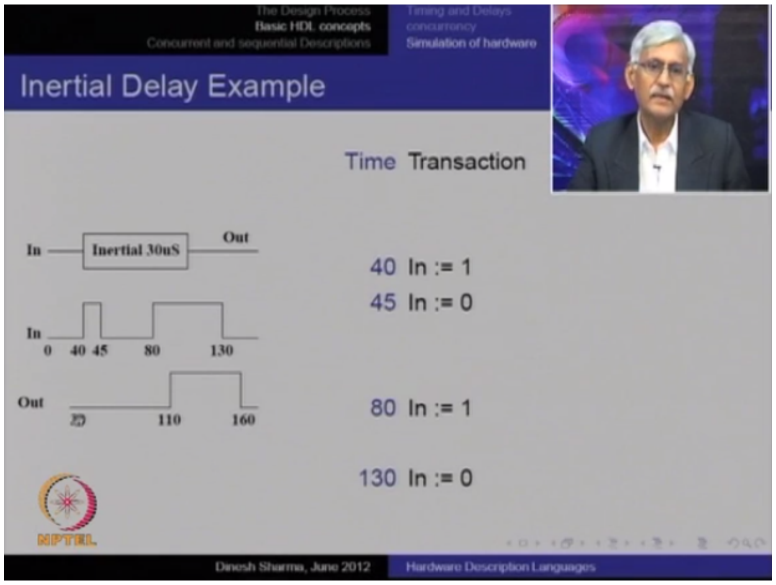
Inertial Delay Example

Time	Transaction
30	out :=0
40	In := 1
45	In := 0
80	In := 1
130	In := 0

Dinesh Sharma, June 2012 Hardware Description Languages

So the delay module inserts an output staying at 30 make output equal to 0, remember it is just a delay module, it just replicates its input at the output 30 units of time later, so at 0 the input became 0, the module says at 30 make my output equal to 0. The earliest time in this time ordered queue is 30 so therefore this is the output and it has said at 30 make my output equal to 0. So we advance the time to 30 and make the output equal to 0 that is all that happens at 30 units of time and then we delete that entry.

(Refer Slide Time: 43:50)

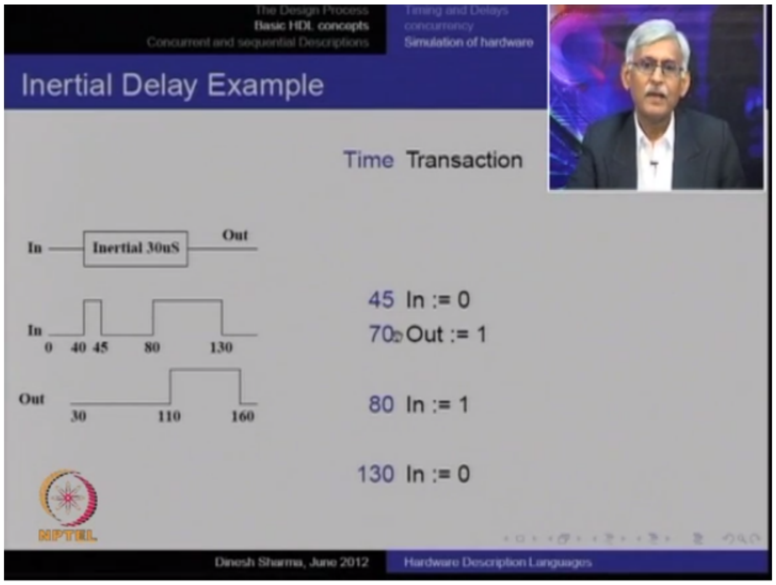


Inertial Delay Example

Time	Transaction
40	In := 1
45	In := 0
80	In := 1
130	In := 0

The next entry is at 40 when something happens so we advance to 40 and input has become 1, input has had an event therefore we must ask our delay module what happens, the delay module says, current time is 40, the input has become one therefore please make my output one at 70 units of time.

(Refer Slide Time: 44:29)

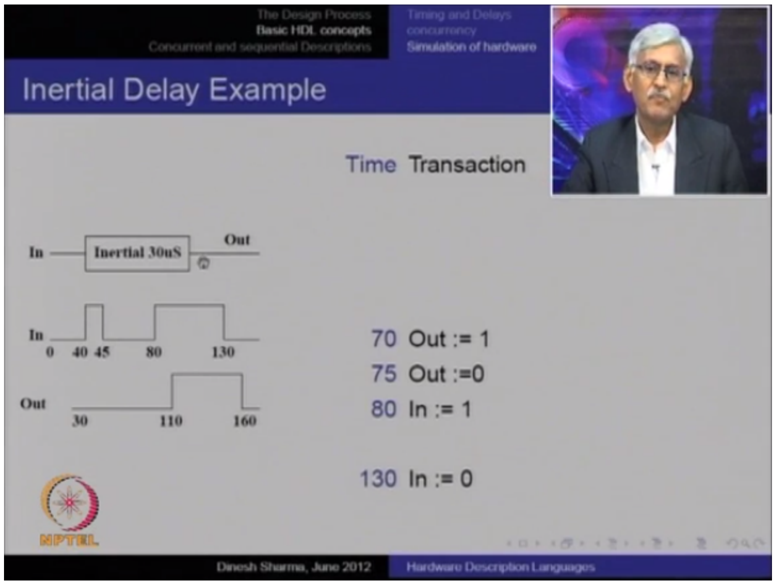


Inertial Delay Example

Time	Transaction
45	In := 0
70	Out := 1
80	In := 1
130	In := 0

And then we delete the input change so we have inserted a transaction which says at 70 make the output equal to 1. We advance the time to 45 and what happens at 45 the input goes to 0 therefore we must wake up our delay circuit because the input is its input and it says okay if the current time is 45 and the input just became 0, please make my output zero at 75.

(Refer Slide Time: 45:15)



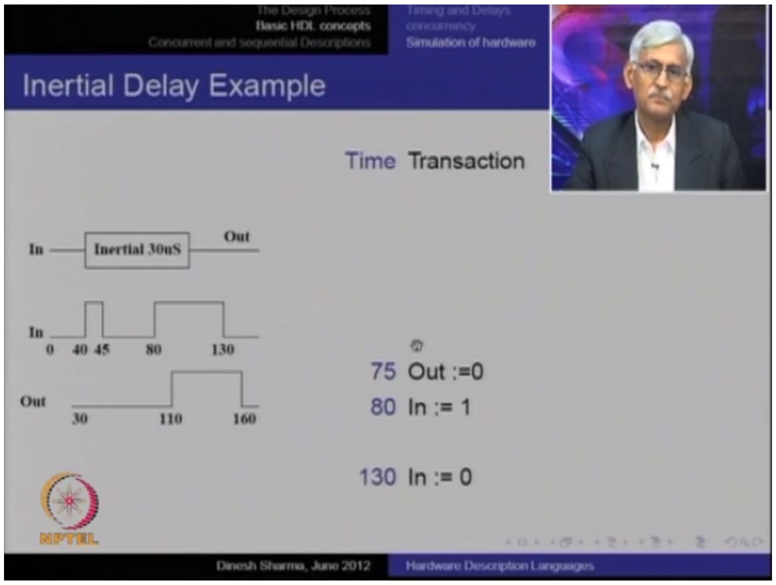
Inertial Delay Example

Time	Transaction
70	Out := 1
75	Out :=0
80	In := 1
130	In := 0

So we go ahead and see what happens if we have an entry of a transaction at 75. Now we have two transactions waiting for different values for the same signal output. And our rule was that if the delay is inertial then this later entry should wipe out the early entry completely, we should take no notice of this, therefore we just delete this without any action. Notice the output will never be made 1 because this transaction will remain the current value of out is 0.

This transaction might have taken it to 1 but because this transaction has come for 0, which is a different value the later transaction in case of inertial delay will simply wipe out this transaction without making it fructified without it having any effect, so therefore we just delete that transaction.

(Refer Slide Time: 46:25)



Inertial Delay Example

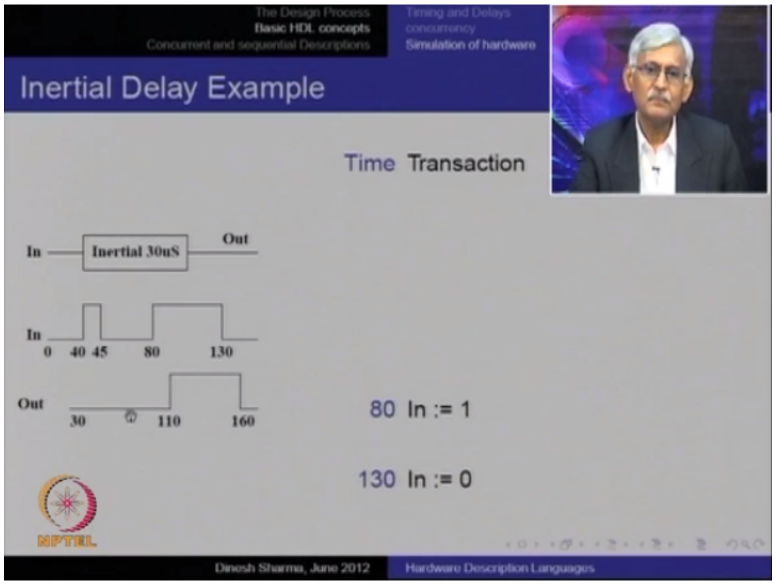
The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
Timing and Delays
concurrency
Simulation of hardware

Time	Transaction
75	Out :=0
80	In := 1
130	In := 0

Dinesh Sharma, June 2012
Hardware Description Languages

Now the earliest transaction is for a time equal to 75. We never visited that transaction which was for 70 because it got wiped out. So the earliest time is 75 and what is the waiting transaction, make output equal to 0, output is already 0, so therefore at 75 we say alright be happy we will make the output 0. No event occurs in any case nobody is sensitive to output so we just delete this output.

(Refer Slide Time: 47:00)

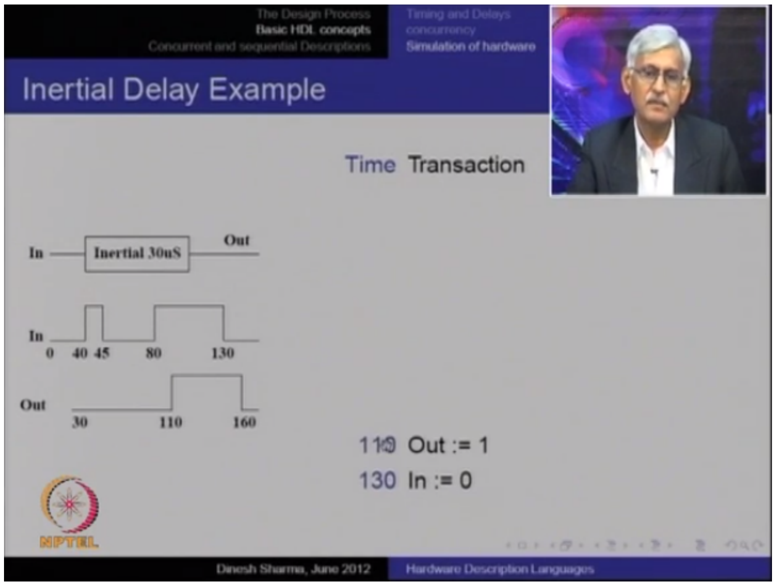


Inertial Delay Example

Time	Transaction
80	In := 1
130	In := 0

The next waiting transaction is at 80 and it says make input equal to 1. Notice at 80 the input should become 1, so the delay element says if the input became 1 at 80, please make my output 1 at 80 plus 30 which is 110.

(Refer Slide Time: 47:23)

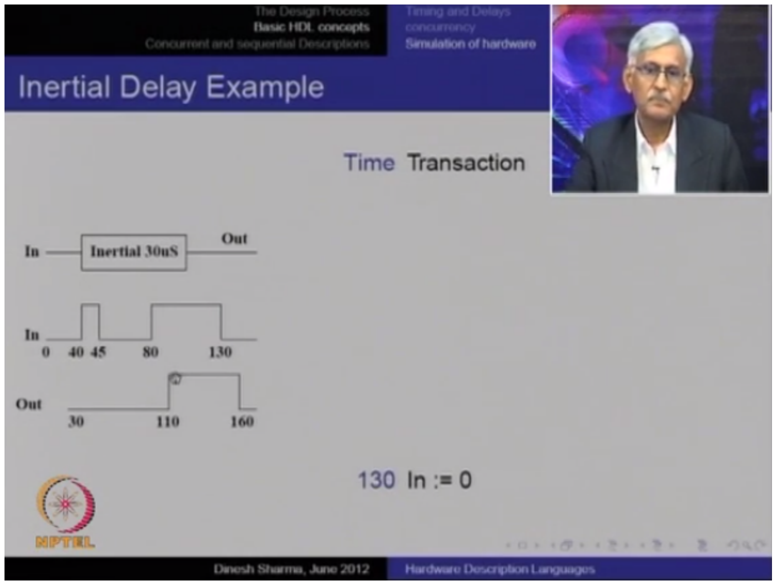


Inertial Delay Example

Time	Transaction
110	Out := 1
130	In := 0

So it inserts a transaction at 110 making output equal to 1. Now the earliest entry in the time ordered queue is at 110, output is to be made one so we do that at 110 we make the output equal to 1, nobody sensitive to the output therefore no re-simulation is required so we make the output equal to 1 and then delete this transaction.

(Refer Slide Time: 47:50)



Inertial Delay Example

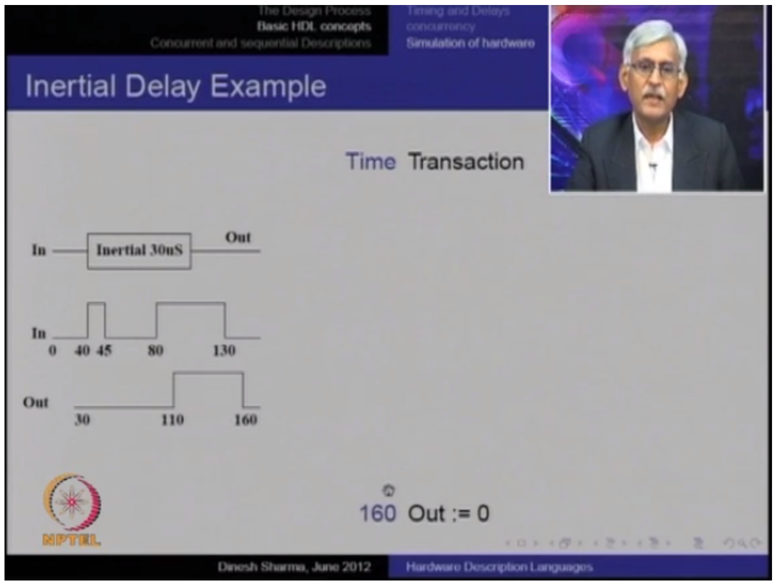
Time Transaction

130 In := 0

Dinesh Sharma, June 2012 Hardware Description Languages

Now the waiting transaction is at 130 when input becomes 0, the delay circuit is sensitive to the input and it says if the current time is 130 please make the output 0 at 160.

(Refer Slide Time: 48:12)



Inertial Delay Example

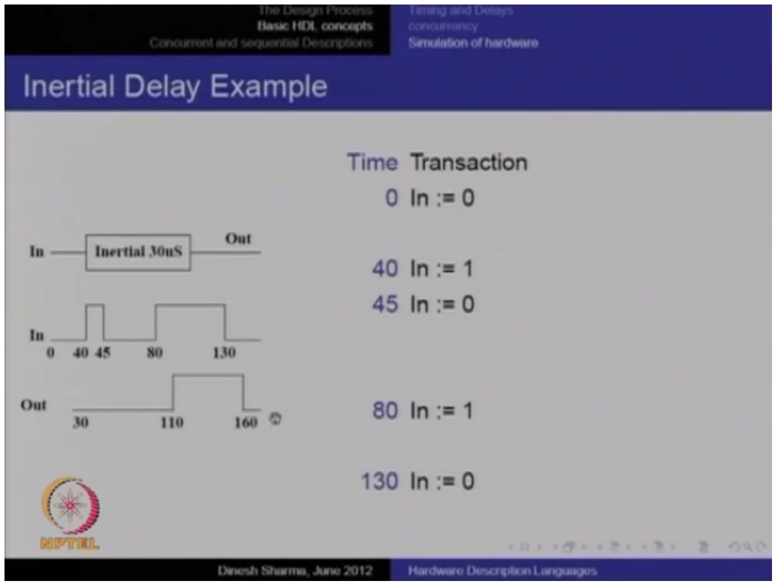
Time Transaction

160 Out := 0

Dinesh Sharma, June 2012 Hardware Description Languages

So that is the transaction that we add that at 160 make the output equal to 0. This is the only transaction left in the time ordered queue and as a result we make the output equal to 0 at 160. Beyond this nobody is sensitive to the output, nothing else happens and we have emptied out our transaction queue.

(Refer Slide Time: 48:37)



Inertial Delay Example

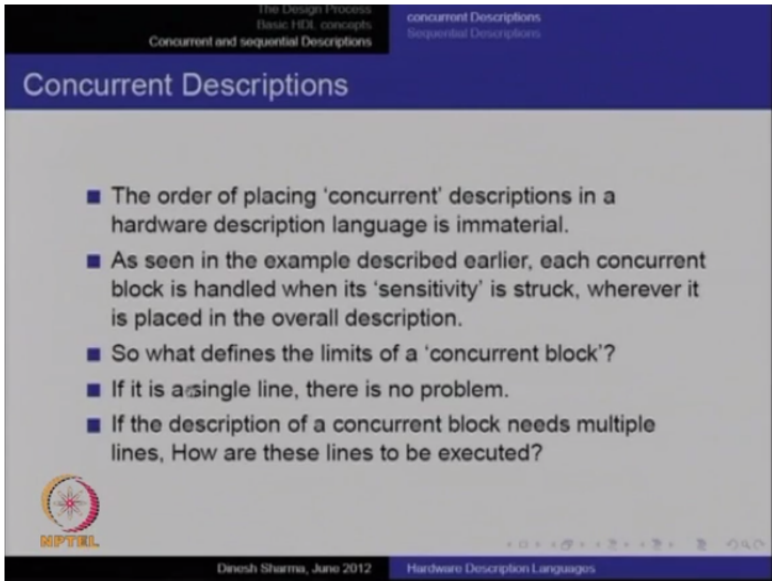
Time	Transaction
0	In := 0
40	In := 1
45	In := 0
80	In := 1
130	In := 0

Let us look at it a little quickly so that it becomes clear how we have gone about this. Remember the delays inertial we have a narrow pulse between 40 to 45 and a white was between 80 to 130 this was our initial time transaction, the input becoming 0 resulted in the output becoming 0 at 30. We make the output 30 and delete that. At 40 the input becomes 1, this results in placing of transaction at 70 units of time for output to become 1.

And we delete the entry at 40, the next event is 45 input equal to 0, the result of that is that we must insert a transaction at 75 saying output equal to 0, this would result in two waiting transaction for output for different values. Because the delay is inertial, the earlier transaction to a different value will be deleted so this transaction at 70 is simply deleted. As a result, we are left with only the transaction at 75 which says make the output equal to 0, output is already 0.

So therefore it has no effect, the next transaction is at 80 when the input becomes 1, the result of that delay therefore is that the output should become one at 110 that is the transaction placed by it. We advance the time 210 and make the output equal to 1. The next transaction is at 130, we advanced the transaction to 130, input becomes 0, the result is that then the output becomes 0. So the expected thing happens, the narrow pulse vanishes, the wide pulse arrives and the wide pulse is delayed by 30 units of time.

(Refer Slide Time: 50:53)



The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions

concurrent Descriptions
Sequential Descriptions

Concurrent Descriptions

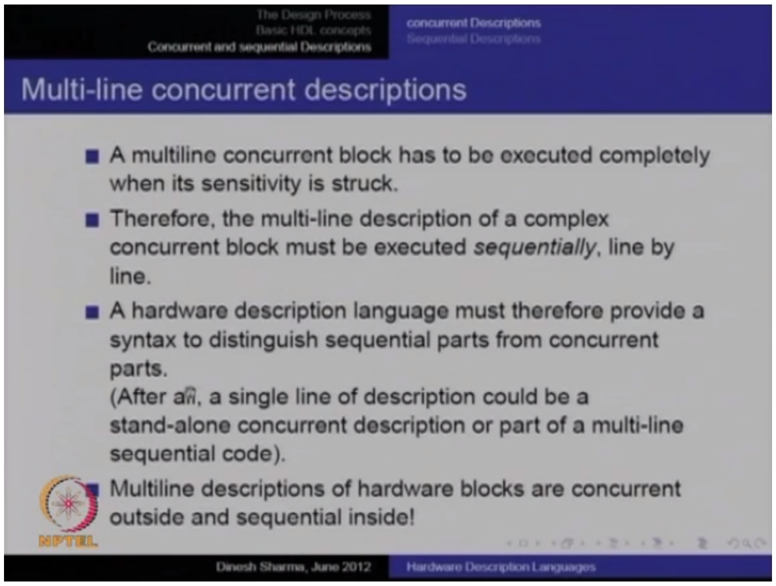
- The order of placing 'concurrent' descriptions in a hardware description language is immaterial.
- As seen in the example described earlier, each concurrent block is handled when its 'sensitivity' is struck, wherever it is placed in the overall description.
- So what defines the limits of a 'concurrent block'?
- If it is a single line, there is no problem.
- If the description of a concurrent block needs multiple lines, How are these lines to be executed?

Dinesh Sharma, June 2012 Hardware Description Languages

So that is a proper way of implementing our delays which could be of transport or of inertial kind. We shall quickly look at concurrent descriptions, the order of placing concurrent descriptions in a hardware descriptions language is immaterial. As seen in the example described earlier each concurrent block is handled when its sensitivities struck.

Wherever it is placed in the overall description we did not care, where that line existed that line was executed when its sensitivity was struck.So then what defines the limits of a concurrent block after all we are going to write it in some file. If it is a single line, there is no problem. When its sensitivities struck we will pick up that single line, but if it is a concurrent block and it needs multiple lines how are these lines to be executed, where do we begin, where do we stop.

(Refer Slide Time: 51:57)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
concurrent Descriptions
Sequential Descriptions

Multi-line concurrent descriptions

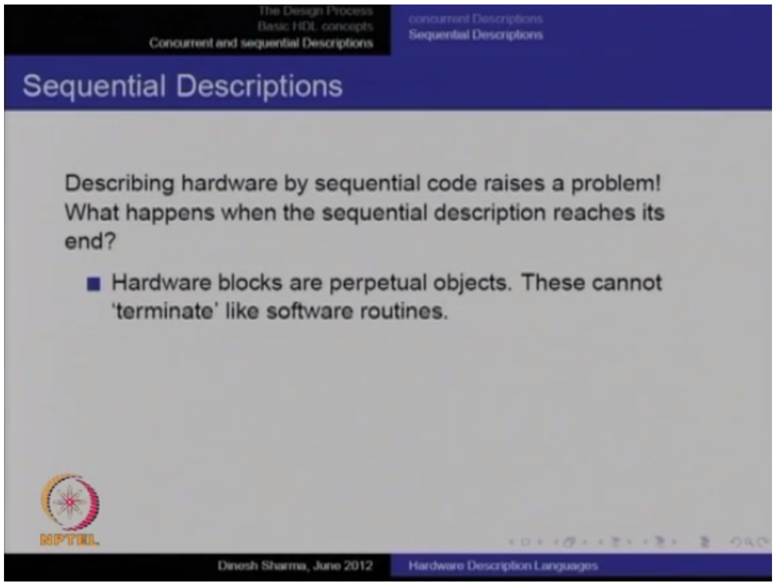
- A multiline concurrent block has to be executed completely when its sensitivity is struck.
- Therefore, the multi-line description of a complex concurrent block must be executed *sequentially*, line by line.
- A hardware description language must therefore provide a syntax to distinguish sequential parts from concurrent parts.
(After all, a single line of description could be a stand-alone concurrent description or part of a multi-line sequential code).
- Multiline descriptions of hardware blocks are concurrent outside and sequential inside!

Dinesh Sharma, June 2012 Hardware Description Languages

So a multiple concurrent block has to be executed completely when its sensitivities is struck all the lines have to be executed and therefore the multi-line description of a complex concurrent block must be executed sequentially, line by line, just like a program. A hardware destruction language must therefore provide a syntax to distinguish sequential parts from concurrent parts after all a single line of description could be a standalone concurrent description or part of a multi-line sequential code.

So multi-line descriptions of hardware blocks are concurrent outside and sequential inside.

(Refer Slide Time: 52:40)


The Design Process
Basic HDL concepts
Concurrent and sequential Descriptions
concurrent Descriptions
Sequential Descriptions

Sequential Descriptions

Describing hardware by sequential code raises a problem! What happens when the sequential description reaches its end?

- Hardware blocks are perpetual objects. These cannot 'terminate' like software routines.

Dinesh Sharma, June 2012 Hardware Description Languages

We shall look at the problems brought by this dichotomy and that we shall take up in our next lecture. We shall begin with discussion of this particular problem and then go on to learn about a specific hardware description language which is VHDL. We end our discussion of the general underlying principles of hardware description languages at this point. In the next lecture we will look at this problem of concurrent descriptions being multi-line and look up a specific hardware description language which will be VHDL. We end this lecture here.