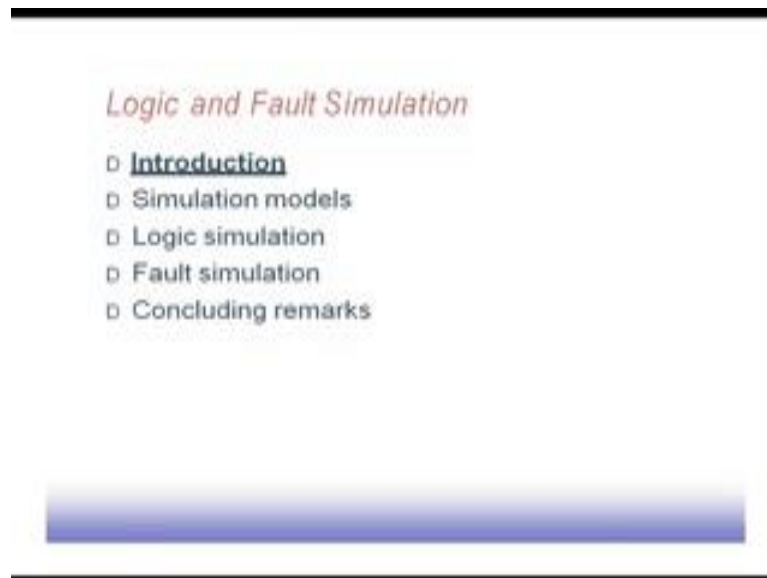


Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Logic and Fault Simulation

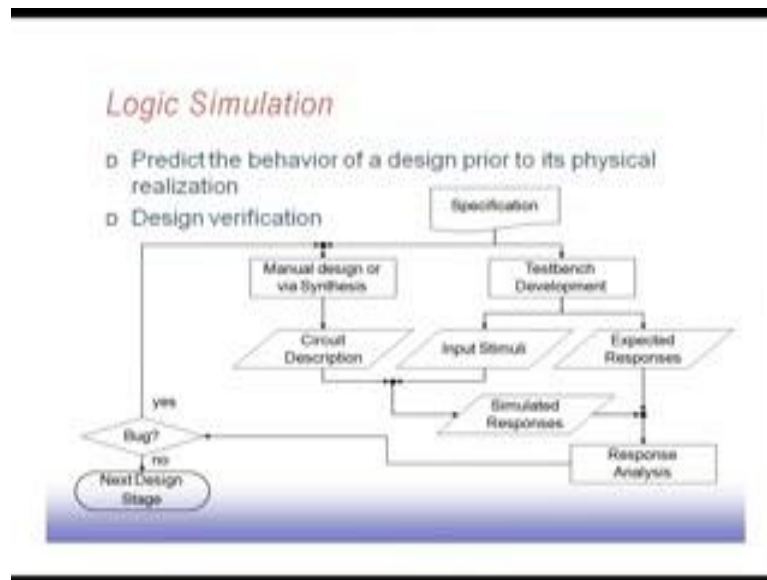
Next, we will start with Logic and Fault Simulation. So, this is basically some software tool that has to be designed for checking the; for getting confidence about the goodness of this module, the test pattern set that we have generated or that we are using for testing some system.

(Refer Slide Time: 00:44)



It will go through this flow; first we will have an introduction, then the simulation models, then logic simulation, fault simulation and the concluding remarks.

(Refer Slide Time: 00:52)



Logic simulation; it predicts the behavior of a design prior to its physical realization. So, what happens is that suppose we have got a circuit to be designed, we have generated some test pattern set for that and now how do I judge the quality of this test pattern set? One possibility is that once the design has been obtained once the design has been fabricated, if the chip has been manufactured, we can apply those test patterns and see whether it is detecting some faults that might have crept into the system and crept into the chip and do I get a confidence, it is able to detect those failures.

If it is true that for all the chips that are detected by this process, they are really faulty then we are happy, our tool, our test pattern generator could generate test patterns for this purpose or it may so happen that for all the defective chips our tool also; our testing process also tells that they are faulty. So that is the other avenue and in which we can say that our test patterns are good, but that is far away and in fact, that is after the chip has been manufactured, but can we do something so that before this manufacturing is done, we can get confidence about the goodness of the test pattern set that we are using.

So, this predicts the behavior, this logic simulation problems; this predicts the behavior of a design prior to its physical realization. So, this is basically the problem of design verification thus specification is there. So, it goes into manual design or via synthesis. So, it may be manual design or may be by synthesis tool. So, it gives the circuit description. When this manual design is going on, side by side it also goes into a

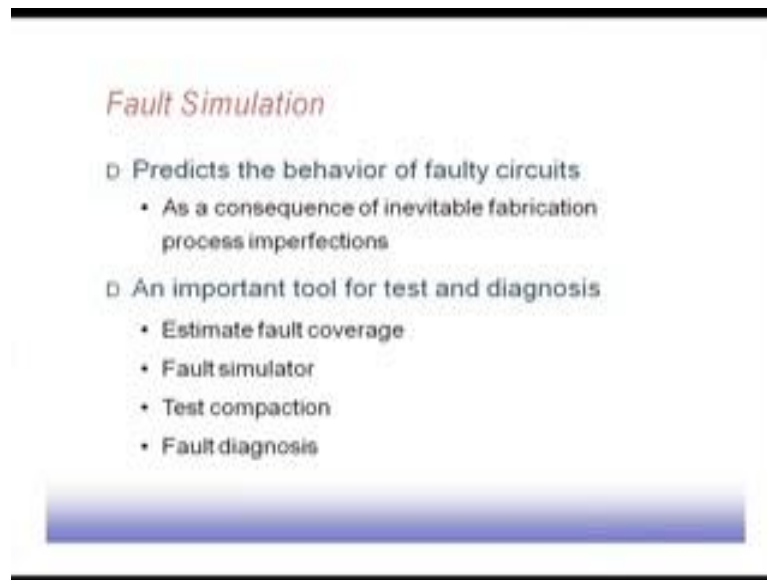
Testbench development. So, in the Testbench development, it will figure out like what are the; what may be the possible input patterns that that will be applied to this system and what are their correct responses?

They are actually forming this input stimuli and expected responses. So, this Testbench development process comes up with these 2 output input stimuli and Testbench responses. Now after this circuit has been synthesized or this circuit, the system has been synthesized, you have the circuit description. So, this circuit description, we simulate with respect to this input stimuli. So, till now I do not have the actual chip manufacture. So, I have got only the description of it after the synthesis has been done. So, the description is simulated with that input stimuli and accordingly we get the simulated response.

Now this simulated response we compare with the expected response and if these 2; from these 2, we do a response analysis. Now if these 2 responses are not matching; that means there is a bug. So, if there is a bug that so we need to correct. So, what do we need to correct? We need to correct this design process and that way again this it will go through the modification, it will do the synthesis and it will go on like this; however, if there is no bug then of course, we can go to the next design stage that may be the actual implementation of the circuit and all that, but what about the quality of the Testbench that we have developed.

With respect to a given set of Testbench, we are getting the confidence that there is no bug in the system, but it is actually the system does not contain any bug which could have been detected by this particular Testbench, only this much is the confidence level. So, we have a better confidence the Testbench development should also be judged. So, they need to be judged like how good or bad is this Testbench.

(Refer Slide Time: 05:00)



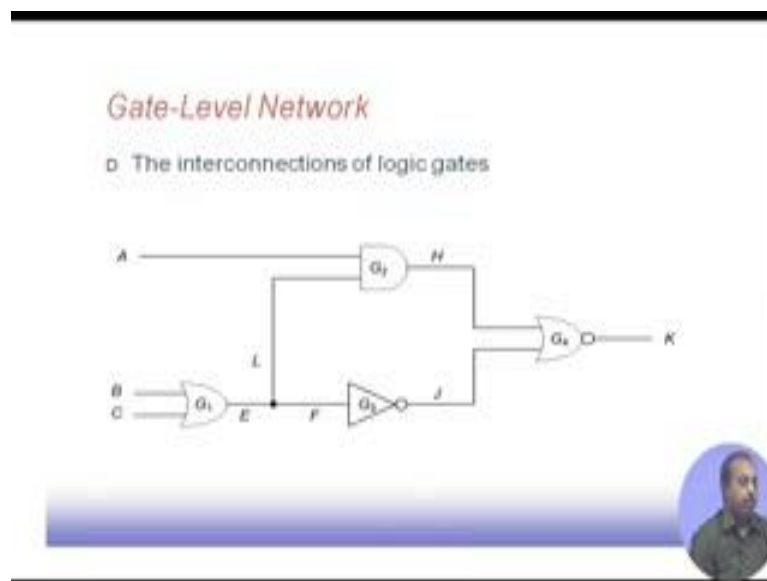
This fault simulation; it predicts the behavior of faulty circuit as a consequence of inevitable fabrication process imperfections. So, it tells like if this particular fault has occurred in the circuit then if you apply this particular pattern, you will get this type of output. So, that is the fault simulation. So, this is a very important tool for test and diagnosis purpose because it can tell us like what is the fault coverage like if I apply this test patterns set how much is the confidence that my circuit does not have any of those modeled faults as we have said that if there are n lines and you are considering single stuck at fault model then there are 2^n possible faults. Out of this 2^n possible faults how many of them are getting detected by my test pattern set?

That may be the type of confidence that gives us. So, that is the fault coverage and also it gives us some way to do to diagnose the problem like it may so happen that some test; some test has failed, in the sense that we applied some pattern to a circuit for simulation or at the manufacturing stage. So, we apply the pattern and we see that the fabricated chip it has failed that test, but why has it failed? So, from the faulty response we should be able to trace back and see where exactly is the fault? So, that is actually the problem of diagnosis. So, at the fault simulation, you can tell us like if you get this type of behavior then possibly the fault is somewhere here, if you get for this pattern so if you get some other response then possibly the fault is somewhere else. So, that helps us in pinpointing the fault location, locating the faults.

They help us in estimating the fault coverage; this fault simulation it helps us to tell give us the confidence how many faults are come? What is the percentage of faults covered then fault simulator like it is simulating the faults like what type of faults we can simulate? Single stuck at fault, multiple stuck at fault, Beijing fault, etcetera, it can also help us in test compaction because we know that for the one particular fault may be detected by a number of patterns. So, it may be true that I do not need to apply all those test patterns for testing a particular fault.

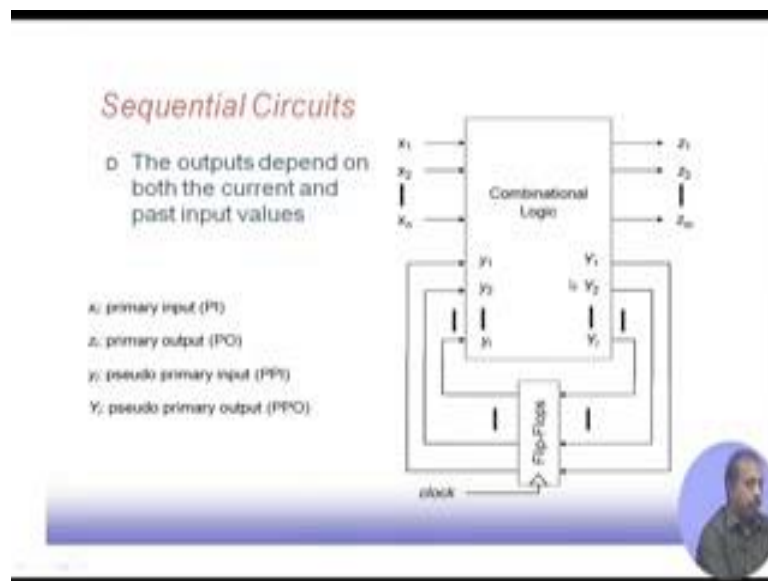
Can we select a subset of test patterns from a given set so that all faults are covered, but we do not apply all the patterns? So, that is actually the process of compaction. So, that can be done to reduce the test application time and test data volume and fault diagnosis that we have already discussed. So, if the manufacturing process if there is some problem then this fault simulation it can tell us like where exactly the fault may be located.

(Refer Slide Time: 08:02)



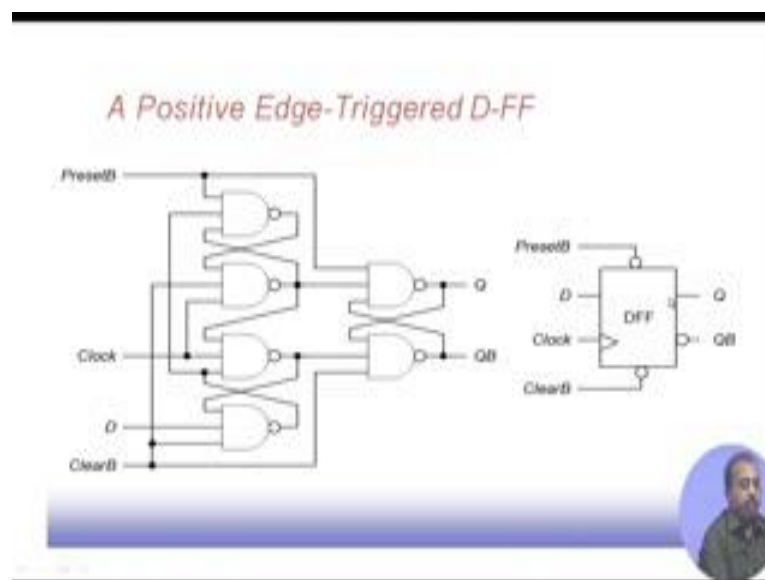
Simulation models; the first one is the log gate level network. So, we can we can think that my circuit is at gate level. So, it interconnection of logic gates, like this here we have got this circuit with 4 gates. So, that maybe gate level network.

(Refer Slide Time: 08:19)



For sequential circuits, output will depend both on current and past in past value. So, this may be the model. So, we have got this primary input, primary output, pseudo primary input through this Y_i lines and pseudo primary output to this capital Y lines.

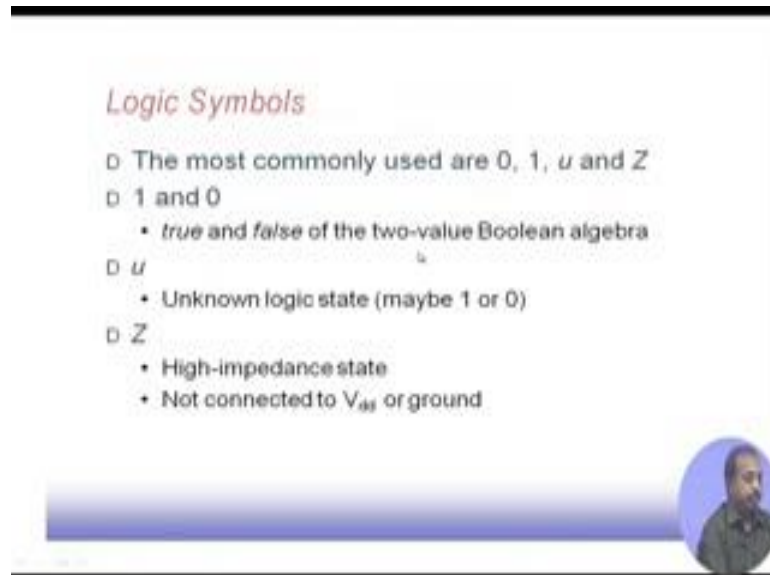
(Refer Slide Time: 08:40)



This maybe the model for the sequential circuit a positive edge triggered D flip flop. So, it may be the actual circuit may be like this, but in terms of block diagram, you may look at it like this that if the clear line is high, clear line is low then this Q will become low

this preset line is low then this Q will become high . So, this logic gate circuit actually is modeling that.


(Refer Slide Time: 09:04)



Logic Symbols

- The most commonly used are 0, 1, *u* and Z
- 1 and 0
 - *true* and *false* of the two-value Boolean algebra
- *u*
 - Unknown logic state (maybe 1 or 0)
- Z
 - High-impedance state
 - Not connected to V_{dd} or ground

09:04



Most of the time will be using these symbols 0 1 u and Z. So, 1 and 0, they are known from our binary logic. So, they are true and false of that Boolean algebra then u; u is unknown at when we are trying to see like what may be the value of some input, what the value of some point in the logic circuit? So, that may be unknown for example, say if we consider this circuit and all on a sudden, we ask what is the value of H without telling explicitly what are the values of A, B and C and simulate and trying to figure out the value of L and E. So, at that point of time the value of H is unknown, that unknown is sometimes used to represent that to be the unknown logic state it maybe 1 or it may be 0 and there is another logic state which is Z high impedance. So, they are not neither connected to V_{dd} nor connected to ground. So, they are neither 1 nor 0, but u means it is either of them, but it is unknown Z means it is high impedance, but it is not connected to either of them V_{dd} or ground.

(Refer Slide Time: 10:15)


Ternary Logic

Three logic symbols: 0, 1, and u

AND	0	1	u
0	0	0	0
1	0	1	u
u	0	u	u

OR	0	1	u
0	0	1	u
1	1	1	1
u	u	1	u

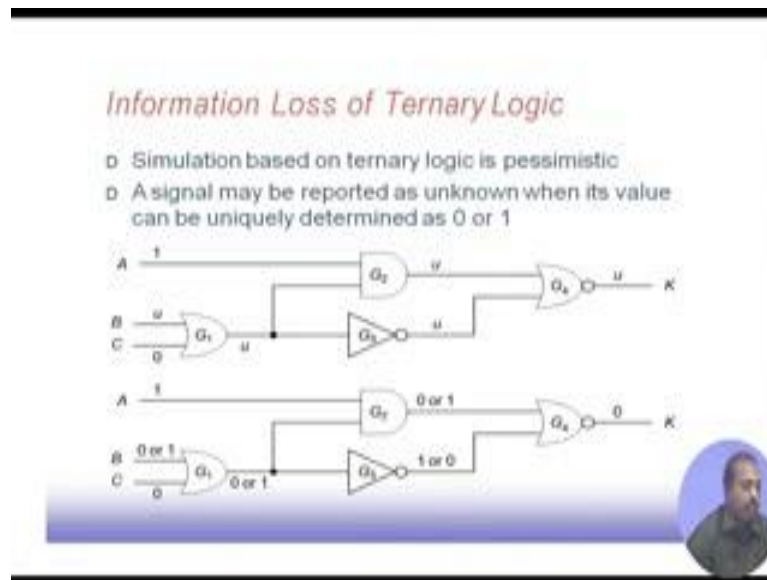
NOT	0	1	u
	1	0	u



In terms of ternary logic, we can define this AND OR and NOT functions like say in a AND gate; 2 input AND gate, if one of the input is 0 then the output is definitely 0. So, one of the input being 0 output is 0, similarly one input being 0 output is 0.

If one input is u then the output depends on the other input. So, if other input 0 it is u , if it is 1 or u then it is u , similarly here also it is like this. Similarly for the OR gate, we can make the corresponding to a logic table. So, 0 and u is u , 1 and u is 1 because whatever be the value of u , it will become 1. So, like that we can do it, similarly not 0 converts to 1, 1 converts to 0, but u remains u only.

(Refer Slide Time: 11:05)

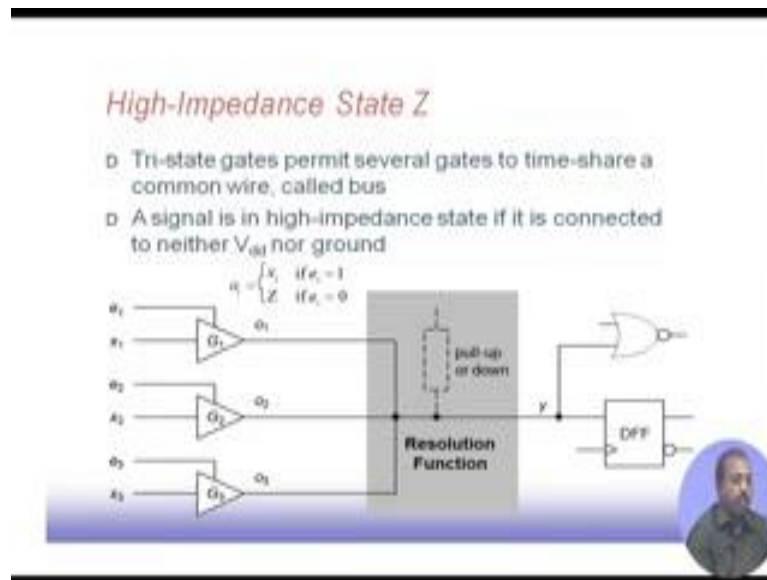


Now, simulation based on ternary logic is pessimistic why a signal value a signal maybe reported as unknown when it is value can be uniquely determined as 0 or 1.

Here it is say this 1; say if we follow this circuit and suppose the value is 1 u and 0. So, this is unknown as a result by the OR logic, since this is unknown, I have to make it unknown and in this AND gate, one input is 1 another as unknown. So, this is also unknown since this is as an inverter and one input is unknown. So, output is also unknown and this NOR gate. So, that is also unknown. So, if we follow a pure ternary logic based simulation of this circuit then this is the situation that many of the points are remaining unknown, now you see that unknown is basically 0 or 1.

What we can say, instead of u if you write 0 or 1, so at this G 1, I cannot say what it is? So, it is it remains at 0 or 1 that is unknown; similarly at this point, one of the input is one. So, whatever it is 0 or one. So, that 0 or one comes here similarly at this point one or 0 comes. So, if this happens to be 0 then this is 1 and this if this happens to be 1, this is 0. So, from this I can find out that these 2 values are never same. So, as a result, this NOR gate output will always be 0. So, that way, we say that the ternary logic simulation is a bit pessimistic, but accepting that. So, it helps in many cases where the values of logic different signal lines are not known.

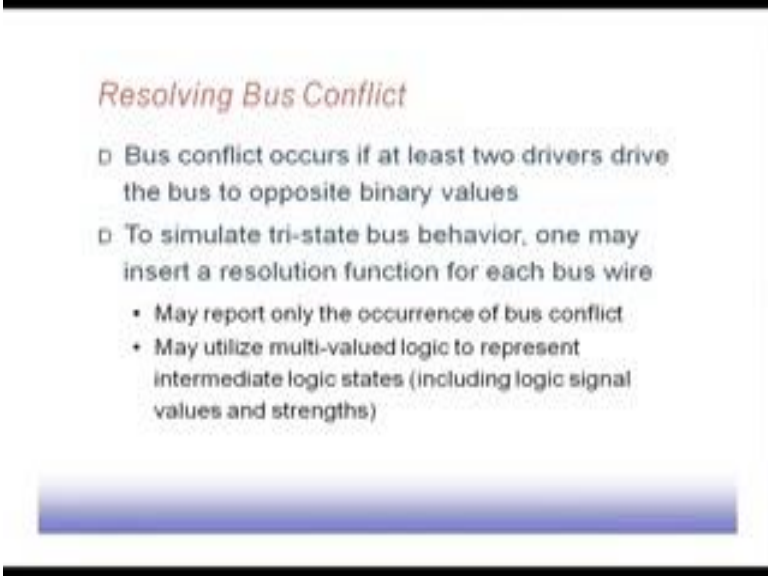
(Refer Slide Time: 12:55)



Doing a simulation at that point of time, we have to go for ternary logic, another is the high impedance state. So, tri-states gates, they permit several gates to time share a common wire like this. So we have got say this G_1 , G_2 , G_3 . So, these are all tri-state gates. So, this if e_1 is high then this x_1 will be coming to the output o_1 and this o_1 , o_2 , o_3 , they are actually connected in a fashion. Now o_i equal to x_i if e_i equal to 1 and o_i is Z , if e_i equal to 0. So, that is the thing now this whole operation, there may be, this is connected here, now when more than one of these devices are on more; more than one of these buffers are on. So, what happens at this point? So, that is defined by the resolution function if all of them are high, if all of them are disabled. So, e_1 , e_2 , e_3 , all of them are 0 then we may have a pull up or pull down mechanism by which this bus will be high or low. So, pull up will make it high, pull down will make it low then it may be fade to some d flip flop or it may be fade to some other combinational element. So, anything can happen. So, it will go like this. So, the sequential it may go to some other circuit input.

This pull up or pull down elements may be there, now resolution function will tell us like how do you get the logic at this point if multiple drivers are active. So, there are wired or wired and we have seen many such models. So, any of those models can be followed and it depends on the technology that we are using like what will be the resolution function.

(Refer Slide Time: 14:43)

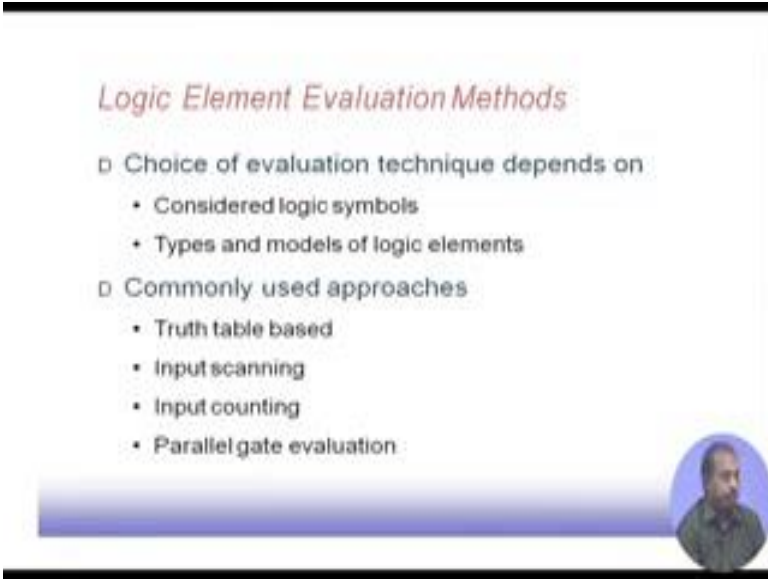


Resolving Bus Conflict

- Bus conflict occurs if at least two drivers drive the bus to opposite binary values
- To simulate tri-state bus behavior, one may insert a resolution function for each bus wire
 - May report only the occurrence of bus conflict
 - May utilize multi-valued logic to represent intermediate logic states (including logic signal values and strengths)

Bus conflict occurs if at least 2 drivers drive the bus to opposite binary values. So, this is the bus conflict to simulate this tri state bus behavior one may insert a resolution function for each bus wire. So, we can say that this wire will be resolved using wired and function other some other place you may say that it this will be resolved by using wired or function depending upon the type of implementation that will finally have.

(Refer Slide Time: 15:38)



Logic Element Evaluation Methods

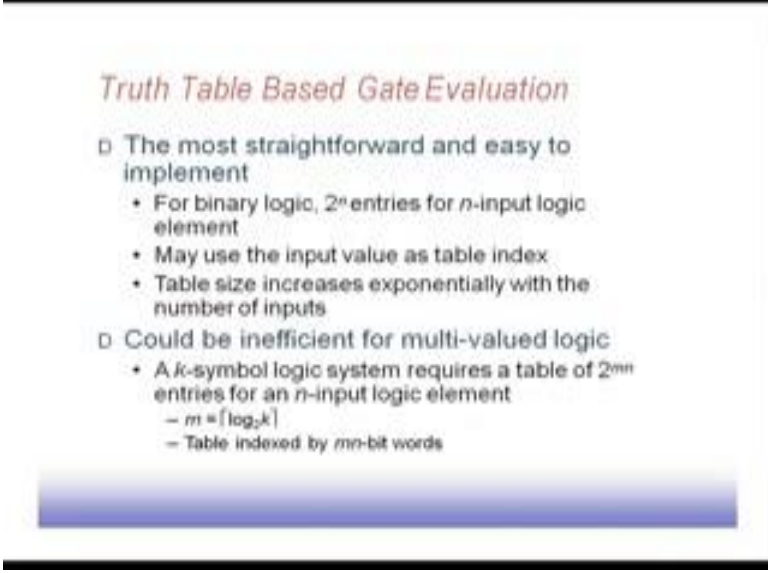
- Choice of evaluation technique depends on
 - Considered logic symbols
 - Types and models of logic elements
- Commonly used approaches
 - Truth table based
 - Input scanning
 - Input counting
 - Parallel gate evaluation

This simulation; it may report only occurrence of bus conflict and they may utilize multi valued logic to represent intermediate logic states as well including logic signal values

and strengths. So, that can also be done like if we instead of having this 2 level logic if you have got multi level logic then we can have a simulation with this intermediate logic states.

Now, this logic element evaluation method like how do you evaluate a particular logic element by for example, a particular logic gate or flip flop like that. So, it depends that the which is evaluation technique that we will use it depends on the type of logic symbols the logic symbol that we have considering and the type of logic elements that we have in our case like if all are um say logic gates and say and flip flops they maybe will be following some we can we maybe following some simple evaluation strategy like say using AND or NAND nor functions like that if it is driven by some other bus resolution function at some point of time maybe we have to use that. So, commonly used techniques are this one the truth table based approach input scanning approach input counting approach and parallel gate evaluation approach. So, you will see them one by one.

(Refer Slide Time: 16:45)

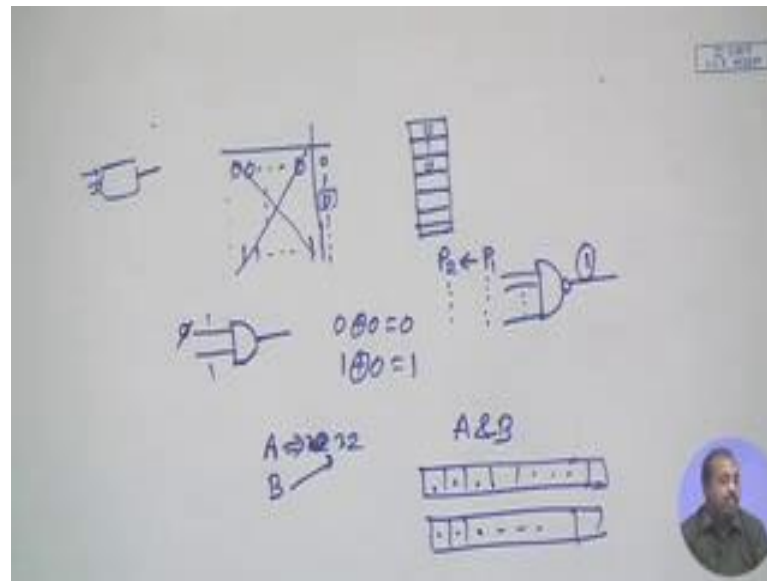


Truth Table Based Gate Evaluation

- The most straightforward and easy to implement
 - For binary logic, 2^n entries for n -input logic element
 - May use the input value as table index
 - Table size increases exponentially with the number of inputs
- Could be inefficient for multi-valued logic
 - A k -symbol logic system requires a table of 2^{mn} entries for an n -input logic element
 - $m = \lceil \log_2 k \rceil$
 - Table indexed by mn -bit words

Truth table based gate evaluation; this is the most straight forward and easy way to implement. So, for binary logic we have got 2 power n entries for n input logic element. So, we can check whether a particular logic value has occurred.

(Refer Slide Time: 17:05)



If I have got some n input function then I may have this all 0 to all 1 cases and for each of them I may have these values noted here. So, what is the output there now for a party? So, if this is the logic block for which we are trying to for which this is the truth table. So, we can check what is the input and try to match with one of them and whichever matches we can take the corresponding value as the output and. In fact, storing this part may not be essential this input part may not be essential we can we can say that as if this is a this is a table where the I have stored only the output column only the output column is stored here and this input part. So, this is used as index of the table.

This is the thing that is for we can have 2^n entries of n input logic element now you may use input value as table index. So, that can directly be used as table index and then from there we can find out what is the value, but the problem with this Boolean table this truth table based approach is that the table size it increases exponentially with number of inputs as number of inputs increases table size also increases significantly and particularly when we are going for multi valued logic the situation is becomes words because if I have got k symbol logic system that will require a table of 2^m entries for an n input logic element or m is equal to \log of k . So, for binary system, k is equal to 2. So, our table has got 2^n entries, but in case of in case of k symbols if there are k different symbols then each of these elements they it can take up each of the inputs can take up any of those k values.

As a result my truth table size will become $2^{2^m n}$ where m is the log of k . So, that way the table in table has to be the index of the table has to be $m n$ bit word. So, that it becomes more complex multi valued logic representation in truth table format will be more complex.

(Refer Slide Time: 19:24)


Input Scanning

The gate output can be determined by the types of inputs

- If any of the inputs is the controlling value, the gate output is $c \oplus i$
- Otherwise, if any of the inputs is u , the gate output is u
- Otherwise, the gate output is $c \oplus i$

Table 3.2: The c (controlling) and i (inversion) values of basic gates

	c	i
AND	0	0
OR	1	0
NAND	0	1
NOR	1	1

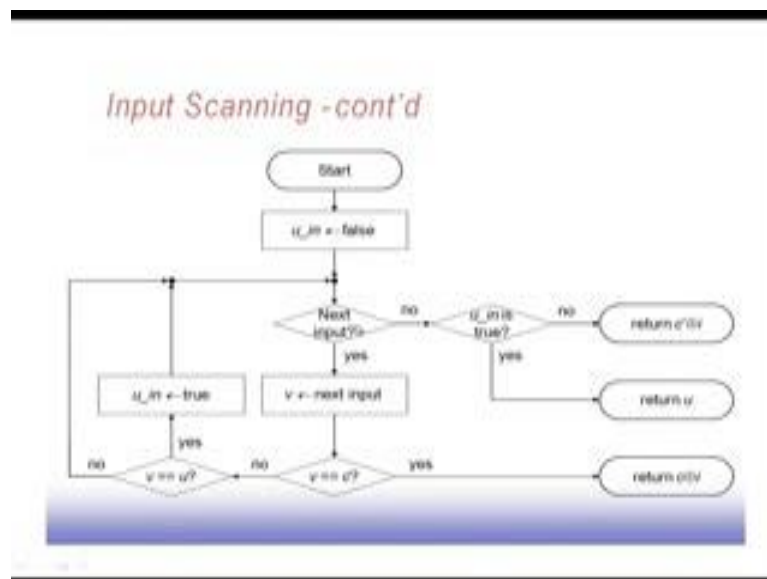


Another way of looking into this logic evaluation is via input scanning method. So, gate output can be determined by types of inputs if any of the inputs is the controlling value the gate output is $c \oplus i$ and otherwise it is the if any of the inputs is u then the gate output is u otherwise the gate output is $c \oplus i$. So, for different types of gate this controlling input and the inversion values are like this for and gate controlling input is 0 and the inversion value is also 0; that means, if I have got an and gate to be simulated it has got 2 inputs. So, to see what is the output? What will be the output of this and gate will see, what are the inputs?

Now, if this happens to be 0; that means, we know that this is this is this has got a controlling value. So, when it has got a controlling value with the output is told to be $c \oplus i$, where i is the inversion value. So, inversion value is also 0. So, $0 \oplus 0$ so that is equal to 0 so whenever this any of the input is equal to 0 output is equal to 0 on the other hand if this is. So, suppose both the inputs are one if both the inputs are one in that case none of them are controlling values. So, as a result it will be coming to this part it is $c \oplus i$. So, c is 1 \oplus 0. So, that is equal to 1. So, in this way the logic

functions can also be captured by means of this c and I value, this AND, OR, NAND, NOR. So, these gates; it is these are shown they are actually capturing the logic symbols the logic values by means of this controlling and inversion values. So, I do not need to store the truth table. So, for every type of gate I need to store the controlling value and the inversion value from there the by applying these 3 rules. So, we can by applying these 3 rules we can find out what is the corresponding value.

(Refer Slide Time: 21:41)

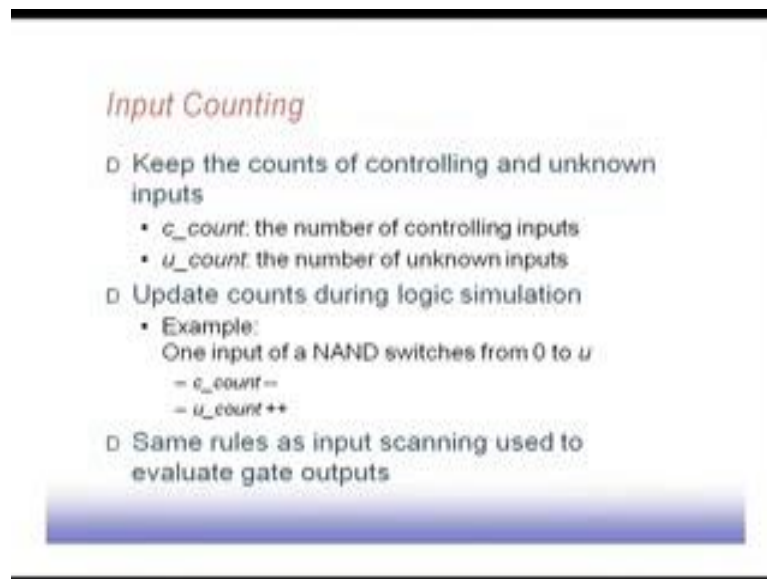


The input scanning method; the algorithm for evaluation is like this, first this u input is equal made equal to false that is we have not seen any input which is equal to u . So, then we take the next input. So, if there, if next input does exist then suppose this is equal to v , if v is the controlling value then we directly return $c \text{ XOR } i$, if v is not a controlling value then we check whether v is equal to u or not, if v is equal to u then this u we have seen one unknown value. So, that u in is made equal to true and if it is not. So, then also it checks whether the next input is there or not. So, if any of the input is equal to u then this u in will become equal to 1.

Now if we have not seen any controlling value then what will happen this input list will exhaust at some point of time. So, it will come out of this no side then it will check with u in is true. So, if u in if none of the inputs are unknown input then it will go to return $c \text{ bar XOR } i$. So, that is that is the value if it is yes then it will return to.

So, it returns 3 possible values $c \text{ XOR } i$, if it c is a controlling value return $\bar{c} \text{ XOR } i$ if it has not seen any controlling value at the same time it has not seen any u and it returns u if it is seen any of the u in the system in the as the input. So, this way this input scanning method. So, this can compute the values of different logic elements in the circuit

(Refer Slide Time: 23:23)



Input Counting

- Keep the counts of controlling and unknown inputs
 - c_count : the number of controlling inputs
 - u_count : the number of unknown inputs
- Update counts during logic simulation
 - Example:
One input of a NAND switches from 0 to u
 - $c_count--$
 - $u_count++$
- Same rules as input scanning used to evaluate gate outputs

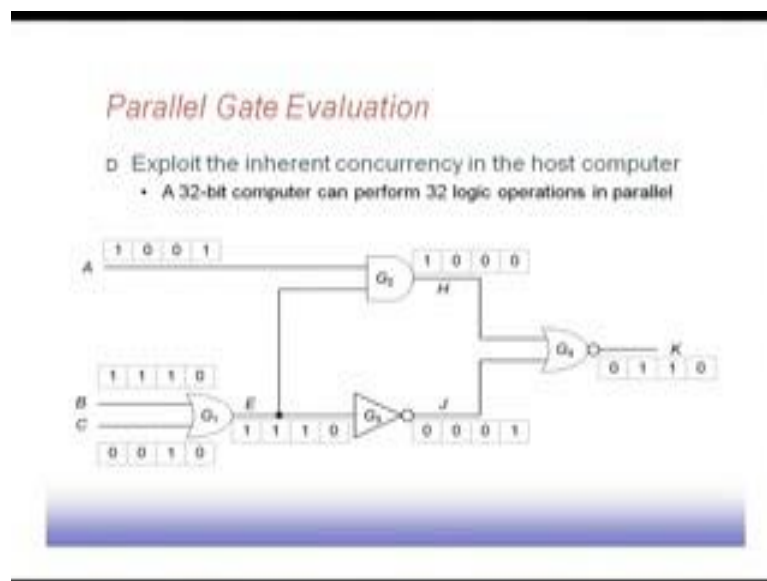
Other approach which is similar to this input scanning method is known as the input counting method. So, what it does it keeps count on the controlling and unknown input. So, c count is the number of controlling input and u count is the number of unknown input. So, it will update counts during the logic simulation for example, if one input of a NAND gate switches from 0 to u , the c count will be decremented and u count will be incremented. So, that way it is, otherwise the rule remains same for evaluating the gate, gate output.

The rules that we have here will be applicable in this one also; however, the point is that it does not do a complete simulation. So, what it will do is that suppose this NAND gate was simulated previously this NAND gate had got a number of inputs. So, it had previously this value was found out to be 1, now in this method what will happen is that it will c for the next input that is coming to the NAND gate, how many changes are there like if this it will count how many c the controlling inputs are there how many unknown inputs are there. So, if there is a change if there is a change from one pattern. So, this is

suppose pattern 1 from there the next pattern, pattern 2 is applied. So, it will count how many controlling values are changed how many undefined unknown values have changed. So, that way it will be computing it.

If it changes from 0 to u then this modification will be done. So, at the end, I by just looking in to this c count value I can see that if the c count value is still greater than 0; that means, the output will be c XOR i and if I find that it is some the u count value is more than um c count u count value is more than 0 and this controlling input is 0 in that case this output will be u and if it is c count value is 0 and u count value is also 0 in that case output will be that c bar XOR i. So, that way the same set of rules will follow, but the effort needed in doing the simulation will be less because each from one pattern to another pattern when you are doing we are just looking into the changes that are occurring in the input set.

(Refer Slide Time: 25:56)



Another possible evaluation to speed up this simulation process is by parallel gate evaluation. So, it exploits the inherent concurrency of the host computer. So, a 32 bit computer it can perform thirty 2 logical operations in parallel. So, what we mean is suppose I have got 2 operands A and B.

In now this A and B can, if A is a 32 bit number both of them are 32 bit numbers, now if I do a logical ending of A and B. So, if I do a sorry, bitwise ending of A and B then what will happen? So, if this is my A register, 32 bits are there and this is the B register 32 bits

are there then when I am doing this bitwise ending. So, these 2 bits will be ended these 2 bits will be ended these 2 bits will be ended. So, bitwise this ending will be done. So, as a result it can evaluate 32 different logic operations simultaneously provided of course, the logic operations are similar then it can do all the 32 operations in a single shot. So, that is what is done here that suppose here I have got this assuming that this word size is 4 bit wide. So I have got say the first inputs is A equal to 1, B equal to 0, C equal to 0, A equal to 1, B equal to 0 and C equal to 0.

It can evaluate this one so this AND gate so, first of all this OR gate is evaluated this 0. So, this is 0 and then this 1 and 0. So, that evaluates to 0 simultaneously, this 0 the second bit is there, this one and this one. So, they are all to get this one and then I can do this thing, I can get this or here and then ending here. So, it can do 32 logic operations in parallel doing this bitwise operations. So, that way we can have much faster execution. So, we continue with the next class.