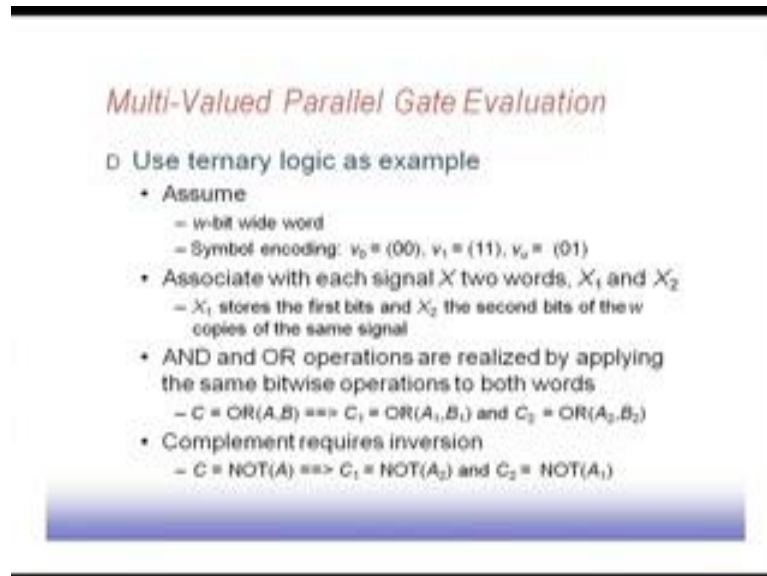


Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 12
Logic and Fault Simulation (Contd.)

(Refer Slide Time: 00:23)



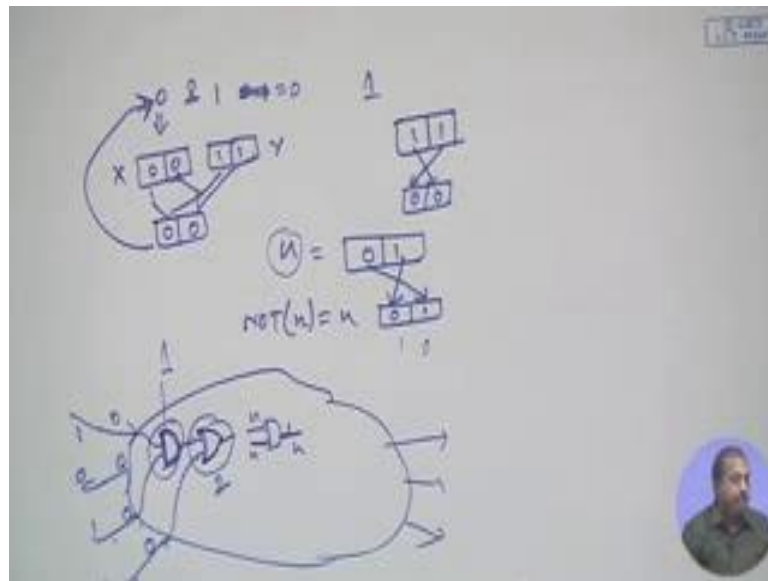
Multi-Valued Parallel Gate Evaluation

o Use ternary logic as example

- Assume
 - w-bit wide word
 - Symbol encoding: $v_0 = (00)$, $v_1 = (11)$, $v_u = (01)$
- Associate with each signal X two words, X_1 and X_2
 - X_1 stores the first bits and X_2 the second bits of the w copies of the same signal
- AND and OR operations are realized by applying the same bitwise operations to both words
 - $C = OR(A,B) \implies C_1 = OR(A_1,B_1)$ and $C_2 = OR(A_2,B_2)$
- Complement requires inversion
 - $C = NOT(A) \implies C_1 = NOT(A_2)$ and $C_2 = NOT(A_1)$

For multi valued logic also we can go for parallel gate evaluation. So, if you take ternary logic as an example and assuming that my computer has got w bit wide word now. So, v_0 , v_1 and v_u unknown suppose that that is 0, 1 and unknown. So, these are the 3 symbols that we have in the ternary logic, suppose their corresponding coding are 00, 11 and 01. So, what we do? With each signal X, we have 2 word; X_1 and X_2 ; X_1 stores the first bits and X_2 stores the second bits of the w copies of same signal. So, we have got this X having 2 word X_1 and X_2 , now this AND OR operation they are implemented like this the C equal to or of A B. So, for C_1 part is computed or of A_1 and B_1 and C_2 part is computed as or of A_2 , B_2 complement is reverse they C_1 is computed as not of A_2 and C_2 is computed as not of A_1 .

(Refer Slide Time: 01:42)

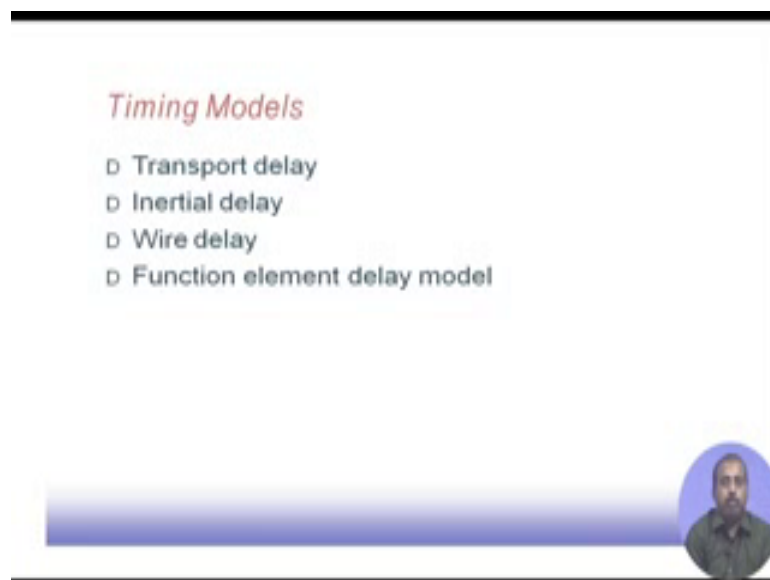


It is like this that suppose I have got same this one and operation of 0 and 1. So, the result should be 1. So, what is happening is that 0 is represented by 2 bits 00 and this 1 is represented by 2 bits 11. Now when I am doing an operation, I have to produce, this is your say X and this is my Y now this. So, this is now when I am doing and of these 2 say this 0 and 1. So, these 2 will be and it and then it says that 0 and 1. So, these 2 bits will be and it and accordingly this will produce a 0 similarly the 0 and 1, they will be and it and it will produce a 0. So, result is 00 which is equivalent to the multi with ternary logic 0. So, in ternary logic I am getting this 0 and 1 that is equal to 0, but in case of universal what happens is that this is suppose I have got some pattern suppose I had 1. So, 1 is represented as 11.

Now, when I am taking and in inversion, this if you look into the rule it says that C 1 is not of A 2 and C 2 is not of A 1. So, this is coming here. So, this is getting a 0 from here and this is getting a 0 from there. So, I am getting a 00. So, why is it done like this is because if one of them is u. So, u is if the symbol is u value is u which is represented by the symbol like this 01. Now when I take not over u the result should also be equal to u. So, as per this rule it says that for getting the not of this symbol this one value the second part it will be it should be with you should take the invert and take it here. So, in the first part you should take the invert and taking to this position you see by doing this I am getting the pattern 01 which is the representation of unknown.

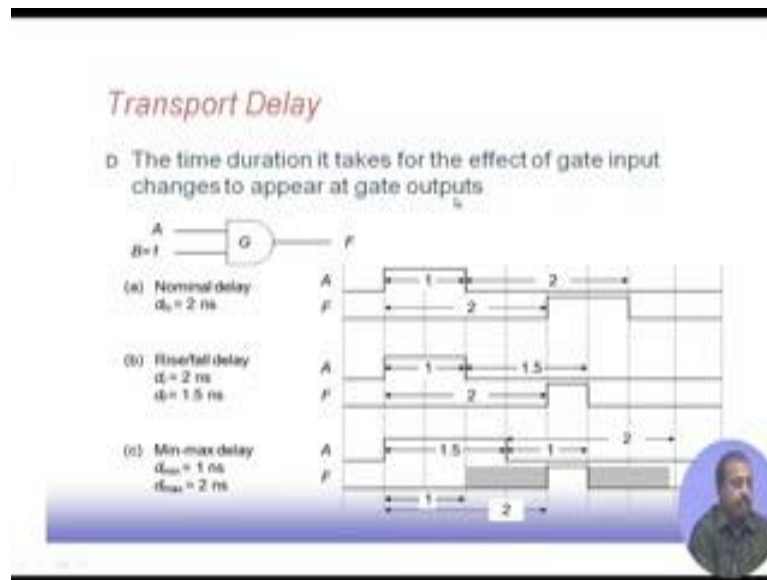
If I do it directly then there is problem because I will get 1 0 and 1 0 does not have any counterpart in the ternary logic representation that we are following here the symbol encoding that you are following here. So, this and or operations they can be realized by applying the same bitwise operations to both the words like if you are doing say OR of A and B then it is basically or of A 1, B 1, C 1 equal to OR of A 1, B 1, C 2 equal to OR of A 2, B 2, similarly if we are trying to do and so, C 1 equal to AND of A 1, B 1, C 2 equal to AND of A 2, B 2 and this 1. So, this way from this A symbol is if it is encoded like this. So, we can take help of this multi bit word of computer the host computer and do this operations in a single shot.

(Refer Slide Time: 04:41)



There are several timing models that we need to consider that are that transport delay inertial delay wire delay functional delay. So, basically when are doing this simulation. So, this delay part also has to be modeled.

(Refer Slide Time: 04:56)



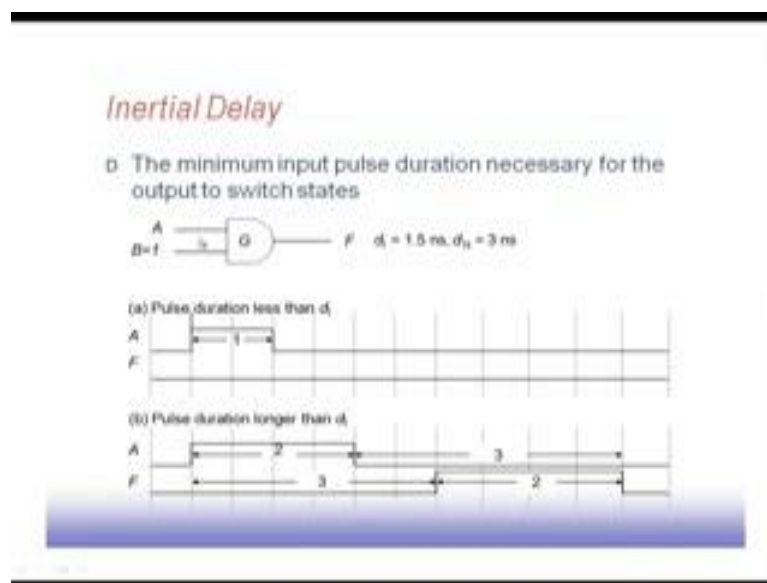
There are several types of delays like this suppose the transport delay. So, transport delay is the time taken it takes for the effect of gate input changes to appear at the gate output suppose I have got a gate G like this it has got a nominal delay of 2 nanosecond; that means, if A is equal to say B is permanently set to 1. So, B is not changing, only the value of A is changing. So, A was initially low and then A is becoming high after one time you it is again becoming low. So, if the nominal delay of this gate is 2 nanosecond. So, from this event to come to F, so it takes 2 nanosecond. So, I get the output becomes high at this point and again there is an event online A at this point. So, this will take another 2 nanosecond to propagate to the outputs. So, these takes another 2 nanosecond and comes back comes down to 0 after this stage. So, this is a nominal delay operation.

Now, there may be rise for rise and fall delay separately. So, the rise delay maybe 2 nanosecond and fall delay may be 1.5 nanosecond. So, in that case what will happen when it is rising? So, this rising effect or rising event to get reflected on to the output it takes 2 nanosecond this falling event, it is takes 1.5 nanosecond to get reflected. So, it is like this. So, this can happen because of this logic families that we have.

As a result we can have different types of this rising delay and falling delays may be different they need not always be same also there is another type of delay model which is known as min max delay. So, min max delay is there is a range of time that is specified d min is one nanosecond and d max is 2 nanosecond. So, it says that suppose this value

changes. So, add this point. So, the output will change between these 2 time minimum it will take one nanosecond and maximum it will take 2 nanosecond. So, between that time the output will change. So, output has changed somewhere here. So, between basically between one and 2 nanosecond it as change similarly when it has fallen here the input A has fallen here again between 1 and 2 nanosecond this has since. So, it can happen anytime. So, it can happen immediately or it can happen at the boundary or it can happen in between. So, between D_{min} and D_{max} . So, these type of delay modeling. So, they are known as transport delay modeling.

(Refer Slide Time: 07:33)

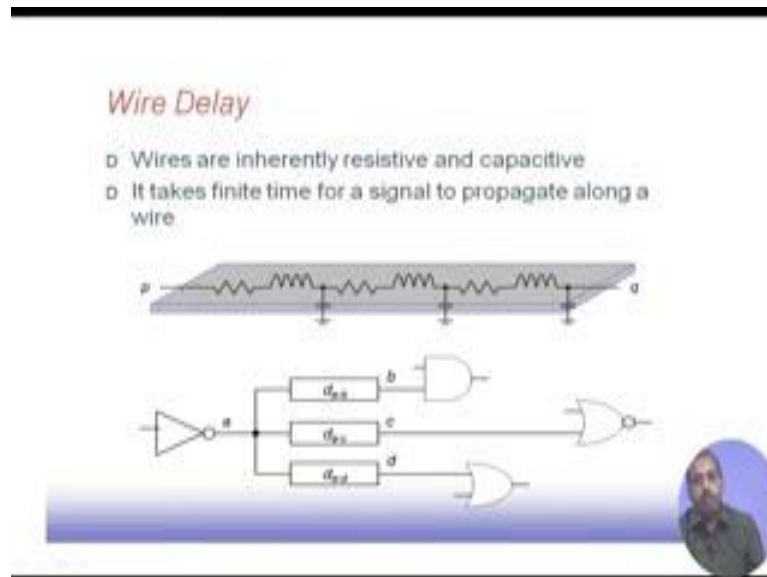


Then there is something called inertial delay. So, it says that the minimum input pulse duration necessary for the output to switch states. So, it may so happen that d_i is 1.5 nanosecond. So, if I have got pulse duration less than that. So, then it will not be affected. So, here it is safe if this pulse duration is a d_i 1 nanosecond that is less than d_i the inertial delay. So, this not did not get affected on to the output so it continued like that on the other hand, pulse duration it may be greater than d_i . So, it is it is greater than d_i .

It gets affected and it has got a nominal delay of 3 nanosecond d_n equal to 3 nanosecond. So, this affect comes to the output after 3 nanosecond. So, after 3 nanosecond we gate a pulse of 2 nanosecond width and then again this is falling down and then that is more than 1 and 1.5 nanosecond it is remaining low. So, it gets reflected

at the output. So, this is the inertial delay. So, the inertial delay is the amount of time that is needed the minimum duration for the input must be held at a particular value for the output to switch.

(Refer Slide Time: 08:51)



Then we have got wire delay. So, wires are inherently resistive and capacitive. So, depending upon the range of frequency it in which we are operating most of the digital circuits. So, we have got this resistive and capacitive effect coming into picture. So, we do not have inductive effect much until and unless you go to the r f frequency range. So, it text a finite amount of time for the signal to propagate along the wire so, this is the if I have got this line a feeding these 3 gates, now you see that there will be some delay associated with this individual segment. So, d a b, d a to c and d a to d, they need not be same. So, that way there will be some delay associated with the wire.

(Refer Slide Time: 09:38)

Functional Element Delay Model

d. For more complicated functional elements like flip-flops

Table 3.3: The D flip-flop I/O delay model

| Input condition | | | | Present state | Outputs | | Delays (ns) | | Comments |
|-----------------|-------|----------|---------|---------------|---------|-----|-------------------|---------------------|---------------------|
| D | Clock | Preset/B | Clear/B | q | Q | Q/B | t _{in Q} | t _{in Q/B} | |
| X | X | ↓ | 0 | 0 | ↑ | ↓ | 1.6 | 1.8 | Asynchronous preset |
| X | X | 0 | ↓ | 1 | ↓ | ↑ | 1.8 | 1.6 | Asynchronous clear |
| 1 | ↑ | 0 | 0 | 0 | ↑ | ↓ | 2 | 3 | Q: 0→1 |
| 0 | ↑ | 0 | 0 | 1 | ↓ | ↑ | 3 | 2 | Q: 1→0 |

X: indicates don't care


Then there are some functional delay model so far so, it is for more complicated functional elements like flip flops etcetera. So, it may so, happen that for going for 1, the output 1 output state to another output state, it takes different amount of time for example,. So, if d n clock both are so if these are do not care this preset line is a preset bar is made high to low then Q output will be becoming 0. So, Q up output Q present value of Q is 0 then the output of Q will go from 0 to high low to high and Q bar will go from high to low. Now delay in nanosecond to Q is 1.6 nanosecond; Q is 1.6 nanosecond and to Q bar, it is 1.8 nanosecond similarly it may so happen that if you are asserting this clear bar line then previous output was one now it becomes 0 in Q and one in Q bar and these are the corresponding delays.

Asynchronous preset and asynchronous clear they have got this type of delays whereas, for a normal operation when this said d line is 1 and there is a clock signal going up the clock H comes then if the present bit is 0 it takes 2 nanoseconds for Q bar for Q output to change from 0 to 1 and 3 nanosecond for Q bar. So, that way Q 0 to 1 transition it takes 2 nanoseconds for Q line and 3 nanosecond for Q bar line and this 1 0 to 1 to 0 transition of Q line. So, it takes 3 nanosecond and it takes to nanosecond for the cube bar line. So, that way, if you have got more and more complex logic module, so they are we can have different types of delays associated with different parts in the circuit.

(Refer Slide Time: 11:44)

Logic and Fault Simulation


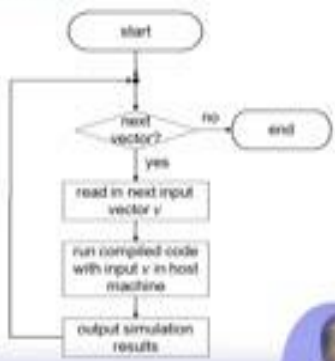
- Introduction
- Simulation models
- Logic simulation
- Fault simulation
- Concluding remarks



(Refer Slide Time: 11:46)

Compiled Code Simulation

□ Translate the logic network into a series of machine instructions that model the gate functions and interconnections.

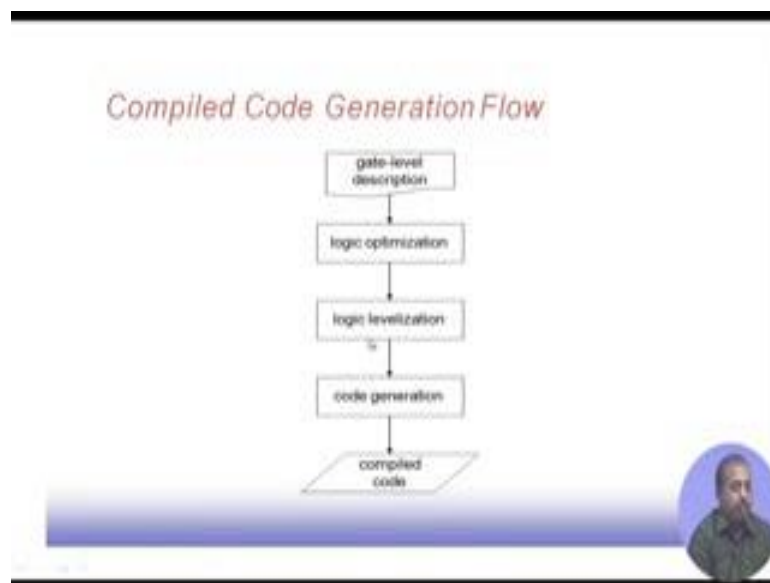


Next we look into this logic simulation. So, logic simulation the first approach is the compiled code simulation. So, what it does is that it translates the logic network into a series of machine instructions that model the gate functions and interconnections. In fact, in the way in the circuit what we have is basically we have got the logic elements like AND gate, OR gate, NAND gate, NOR gate, etcetera and if you look into any processor then this processor normally has these type of statements in the logic statements we have got those things now also if it is like a more complex like if it is a flip flop it etcetera. So, that can also be modeled by means of some if then else statement by and so and by

checking some signal values and all that. So, it can be converted into a piece of code. So, any circuit that we have, we can convert it into a piece of code that we can execute on the on the host computer.

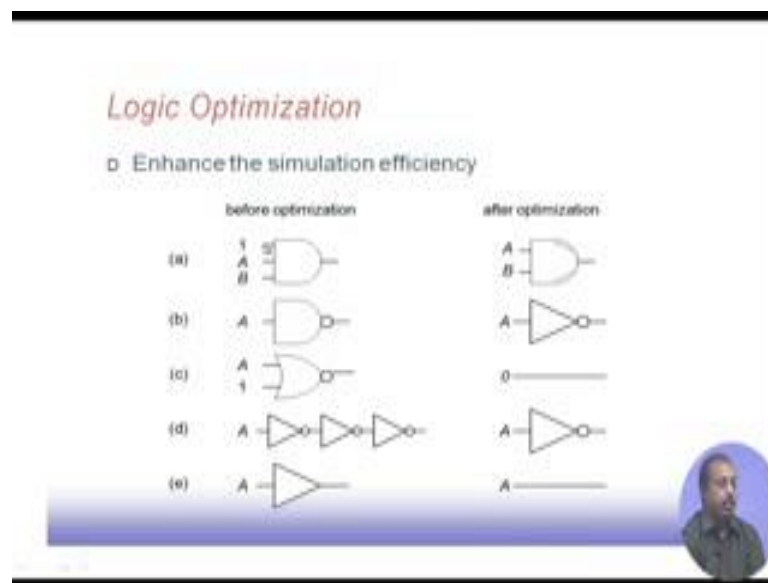
Now, the way this compile code simulation will work is that it will take the next vector. So, if the next sector is existing then it will read the next input vector from the file. So, in a file that is vectors maybe there. So, it take the next test vector v from the file and the it will run the compiled code with the input v in the host machine. So, it will v is taken as input and with that the code will run and whatever be the output of the program that is the simulation result then again it will take the next input next test vector and again do the same thing till all the vectors have been simulated. So, this is possible because most of the machines they have got instructions similar to the logic gates and elements that we have.

(Refer Slide Time: 13:31)



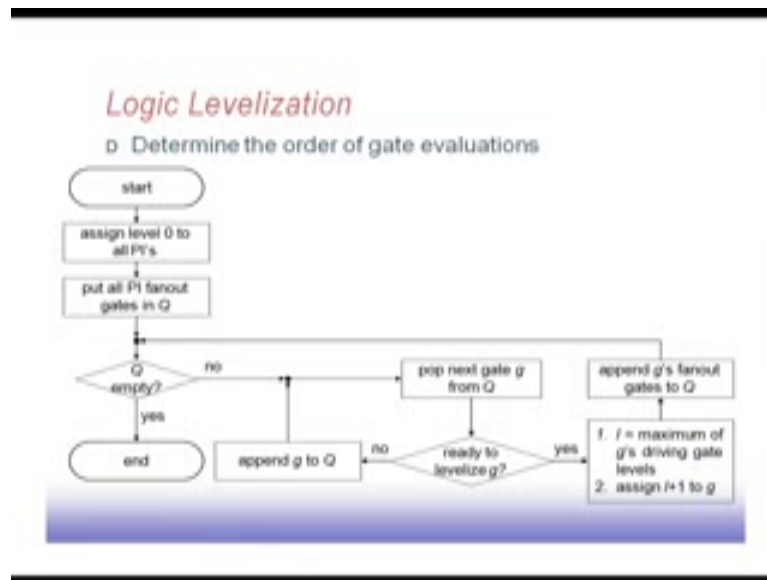
In a flow graph, it is like this the date as the gate level description it goes through logic optimization, it generates then there is a logic levelization. So, logic levelization is actually telling us like what up like which gate should be what say simulated at which point of time after that we have got code generation and that gives us the compiled code.

(Refer Slide Time: 13:55)



A logic optimization is like this may be before optimization we have got a situation like this. So, there are there is a 3 input AND gate out of that one input is permanently 1. So, that can be simplified to have it 2 input and gate where this only 2 inputs are there. So, for the simulation purpose, I do not need to simulate the first one the, simulating 2 input and get is definitely simpler than 3 simulative 3 input AND gate. So, that way it can be done similarly there may be and gate with only 1 input. So, that can be converted into an inverter we can have they were 1 NOR gate where one input is permanently 1. So, we can convert it to a line 0, similarly there are there with 3 inverters is a series. So, that can be converted to one inverter similarly a buffer can be converted to a single line. So, this type of optimizations can be done because for functional simulation. So, we do not need to consider this type of we do not need to consider that that situation that we have shown on the left hand side.

(Refer Slide Time: 15:02)



Next part is to do well levelization. So, the objective of this stage is to determine the order in which the gate evaluations will take place. The problem is that if we have a logic circuit say like this and, so, it has got a number of inputs in the number of outputs now if I am trying to simulate this logic circuit and all of a sudden I start I want to check what is the output of this and gate. So, that is not possible because these values are still unknown. So, if I do a simulation here I will get and u at this point. So, that most of the values will remain as u, but since this is primary inputs are there. So, this test vector that we have. So, that has specified, what are the values of these primary inputs? So, if I have a gate here which is fed from these primary inputs only then I can simulate these gate first.

Now, after that if this gate feeds another gate like this then I can simulate this gate. So, there is an order in which I should do the simulation. So, I should first simulate this gate then this gate and then level by level we should proceed. So, this is actually known as a topological ordering of gates or topological ordering of nodes in the circuit. So, we can do it like this we can have a very simple algorithm that can assign this type of level. So, assign level 0 to all primary inputs and you put all primary input fanout gates in a Q.

Now, if the Q is not empty then we pick up the next Q next gate G from Q. So, since this gate G, since its inputs are all coming from the primary inputs only. So, these must have been their values are known. So, we can we can assign some level to this. So, is it

ready to levelize because it is we can because all the inputs are all the inputs are known to it. So, I put this 1 to be equal to maximum of g's driving gate level. So, whatever be the levels of it is previous level previous inputs or the levels of its input. So from them whatever has the maximum level with that one has to be implemented and that has to be assigned as the level of G.

What will essentially mean is that say if I apply this algorithm on to this process then for first this gate will be picked up and these are actually leveled with 0s. So, this time see these are primary input. So, they are at level 0. Now for this gate I know that all in all it is inputs are at level 0. So, it will be leveled as one because as that 0 plus one that is one now when you are trying to levelize this node then what will happen. So, there actually it has got 2 input, one of them has got a level 1, another one has got a level 0. So, we take the maximum of these 2 that is equal to 1 AND plus 1. So 2, so 2 is made the level of this or gate. So, this way this leveling is done. So, any point of time when it is ready to level g. So, we know that the nodes inputs are already leveled. So, this we can now find out the level for this node if it is not then we just keep that node in the q. So, later on it will be added now this process continues till this Q becomes empty the Q becomes empty means all nodes are been leveled. So, we have got the complete levelization of the nodes.

(Refer Slide Time: 18:47)

Exmpl

Table 1.4: The levelization process of circuit

| step | A | B | C | G ₁ | G ₂ | G ₃ | G ₄ | Q |
|------|---|---|---|----------------|----------------|----------------|----------------|------------------------------------|
| 0 | 0 | 0 | 0 | | | | | <G ₁ , G ₂ > |
| 1 | 0 | 0 | 0 | | | | | <G ₁ , G ₂ > |
| 2 | 0 | 0 | 0 | 1 | | | | <G ₁ , G ₂ > |
| 3 | 0 | 0 | 0 | 1 | 2 | | | <G ₁ , G ₂ > |
| 4 | 0 | 0 | 0 | 1 | 2 | 2 | | <G ₃ > |
| 5 | 0 | 0 | 0 | 1 | 2 | 2 | 3 | <> |

o The following orders are produced

- G₁ => G₂ => G₃ => G₄
- G₁ => G₃ => G₂ => G₄

Now, suppose say this node this graph. So, in the first step this A, B and C, they are assigned level 0 and in the Q, I have got these 2 nodes G 2 and G 1 because their inputs are there at the fun out of this A, B and C then at the step 1. So, I have got this 0, 0, 0. So, this A, B, c, they all these nodes are there then the G 1, G 2 remain, now G 1 will be picked up. So, if G 1 is picked up then at the next level. So, G 1 will be assigned a level 1 and G 1s fanout is G 2 and G 3. So, G 2 is already there. So, G 3 will be picked up. So, you see that here G 2 was there in the Q and since G 2 is listed earlier in the Q. So, it might have s, happened that G 2 is picked up before G 1 for leveling, but when you pick up G 2, we find that it has got some node G 1 which is not yet leveled. So, G 2 will be put back on to the Q, it is appended at the end of the Q. So, G 1 comes to the top of the Q. So, in the next iteration, G 1 is picked up and then G 1 gets leveled.

That way it operates the algorithm that we have noted there it is if you follow the algorithm thoroughly then you will find that at step number five all the gates will get leveled and will have the level like 1 2, 2 and 3. So, the orders that are produced one possible is that G 1, G 2, G 3, G 4, since G 2 and G 3 both have level 2. So, they can be done in any order it may be G 1, G 2, G 3, G 4 or G 1, G 3, G 2, G 4 that does not matter and in fact, if you look into the circuit. So, this G 2 and G 3 they are not dependent on each other. So, they can be simulated in either order.

(Refer Slide Time: 20:39)

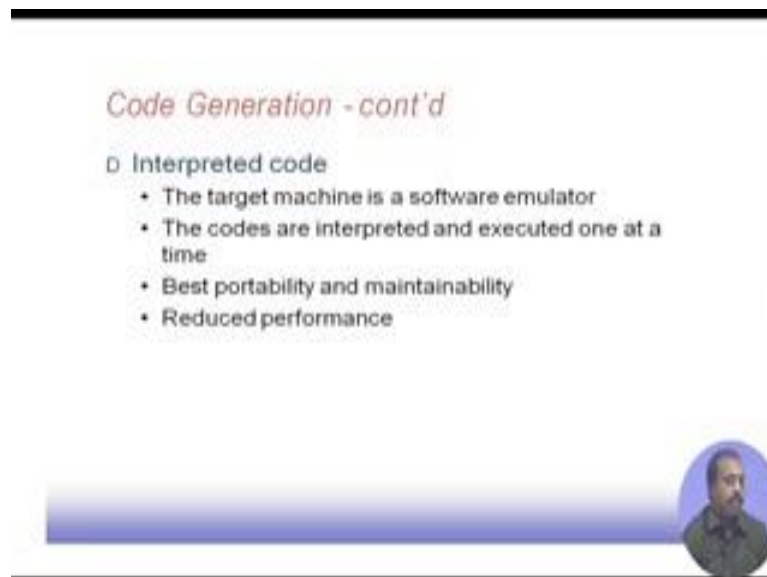
Code Generation

- High-level programming language source code
 - Easier to debug
 - Can be ported to any target machine that has the compiler
 - Limited in applications due to long compilation times
- Native machine code
 - Generate the target machine code directly
 - Higher simulation efficiency
 - Not as portable

Now, for the code generation part, high level programming language source code may be used like we can write down this piece of circuit code corresponding to the piece of circuit into some high level programming language that advantage is that it will be easier to debug it can be ported to any other target machine that has the particular compiler; however, limited in applications due to long compilation time. So, maybe it will take quite some time to do the compilation. So, that way it may be restricted.

Other possibility is that we do it in the machine code directly. So, we generate the target machine code directly. So, in that case, the simulation becomes a efficient because now we are not depending on any intermediate compiler. So, we can we know that if there is an AND gate. So, you have to use the AND instruction of the processor. So, we can do this we can generate this target code directly that is not very difficult. In fact, so that can be done. So, simulation efficiency will be better, but it is not that much portable because if my system on which I am running this simulator. So, if it has got say one particular processor. So, if I take it to some other machine then other computer then it may not have that processor as a result the same code may not run there. So, portability becomes a problem whereas, high level programs they have the portability.


(Refer Slide Time: 22:11)



Code Generation - cont'd

D Interpreted code

- The target machine is a software emulator
- The codes are interpreted and executed one at a time
- Best portability and maintainability
- Reduced performance

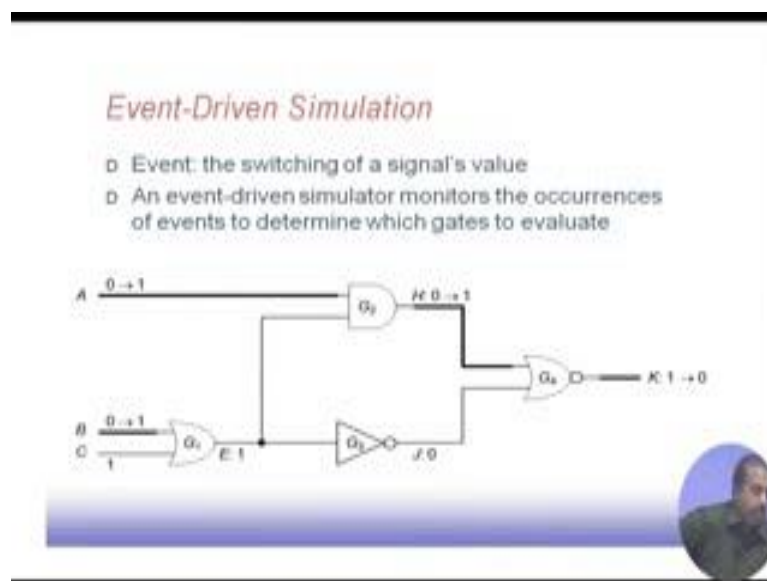


Other possibility is to have interpreted code, interpreted code like say a say languages that has got interpreted in them. So, the target machine is a software emulator. So, it is the codes are interpreter and executed one at a time. So, this is we know that the AND

gate, OR gate, NAND gate, NOR gate, etcetera. So, I can have the corresponding procedures written in that in the machine instructions of the target processor. So, that it can emulate the behavior of these individual gates; individual logic gates and flip flops etcetera. So, that can be that it becomes a software emulator and the codes are interpreted and executed one at a time. So, this is best for portability and maintainability because all the if you are taking on to some other system.

So, what you have to do is just need to have the corresponding procedures written. So, in that system how this AND, OR, NAND, NOR are implemented so that that particular interpreter if it is written properly then it will be working there also. So, these interpreted codes are better, but of course, performance will be slow because interpreter runs slower than the compiled code.

(Refer Slide Time: 23:28)

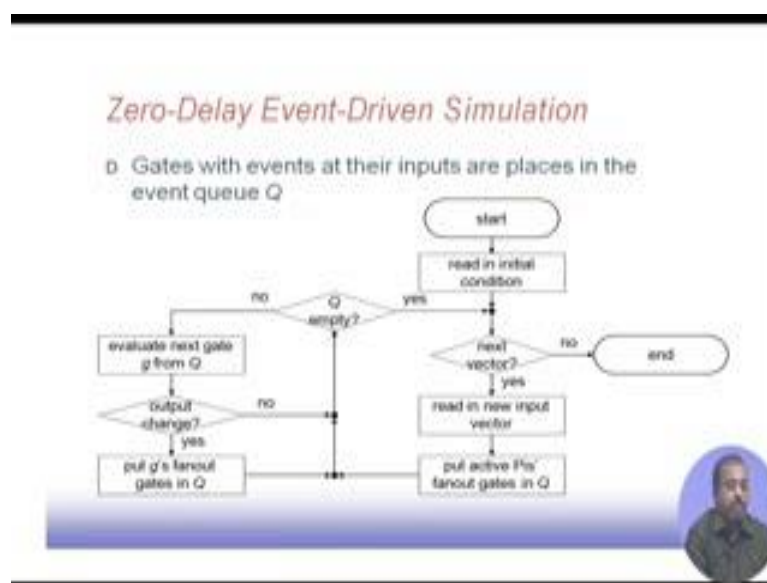


Another way of doing is the event driven simulation. So event driven simulation means that if suppose some event occurs then we will simulate the behavior and we do not simulate the entire we do not simulate the entire circuit. So, we simulate only the portion that is related to the event. So, here actually the example, like if you have got say this situation where previously we had the input 0, 0 and 1 So, this H, so, 0, 0, 1, this E was equal to 1, as a result this G 2 output H was equal to 0, this was 0 and this K output was equal to 1.

Now, suppose there is event so, event in the sense that some of the inputs do change say A changes to 1 and B changes to 1. So, first, this case since there is an event and say G 1s, G 1s input has changed so, G 1 will be simulated. So, G 1 simulated and we see that this E value remains unaltered. So, if E value remains unaltered; that means, wherever it goes. So, that need not be simulated. So, this E value comes to G 3. So, since G 3 does not have any other input. So, G 3 need not be simulated again, but this G 2, it has got 2 lines in it A and E, out of that E does not create any event, but this A creates an event. So, this node G 2 has to be this simulated. So, this is simulated and as a result H becomes 0 1, H becomes equal to 1 from it is value of 0.

There is an event on the line H. So, for the gate G 4 again there is an event on line H. So, this gate has to be simulated and ultimately is a simulated and k becomes equal to 0. So, what E essentially show here is that there may be a portion of the circuit that need not be simulated when you are going in this event driven fashion like in this case, the gate G 3 need not be simulated because it is input is not changing. So, this is a very small circuit. So, there we see that a very small portion it can be left out from simulation, but if you have got more complex circuits then naturally this simulation it may be able to get rid of a larger portion of code from simulation larger portion of the circuitry from simulation. So, that is the event driven simulation. So, this is often used to speed up the simulation process because you do not need to simulate each and every gate in the circuit.

(Refer Slide Time: 25:59)



Now, there are the question of delay like if it may so happen that these gates have got delays and these delays are to be taken care of. So, if there is a 0 delay, if we assume that the gates with events are there inputs are they are they whenever some events occurs at the input immediately the output will be computed. So, in this case, it is simple because we do not have to wait for anything. So, is the flowchart maybe like this, we start read the initial condition if that initial after the initial condition has been read then we check whether there is a any test vector pending if it is vector is pending we read the next input vector put active primary input fanouts gates in Q. So, primary inputs are modified after reading this input vector so whichever primary inputs have changed so, their fanout gates are put into Q.

Now, we need to simulate those gates only. So, we again do the same thing that the all those gates; all those gates fanout, all the put G fanout gates in Q, suppose you have taken out taken the gate G so that for simulation so that gate will be that gates fanout nodes will be put into Q, again if the sorry, I am sorry, so, this is coming here if that it is if it is put into Q. So, now, it checks whether the Q is empty or not, if Q is not empty then it will evaluate the next gate G from Q and if it is equal to if it is correct, if it is evaluated then the output changes then only it will be it will the fanout nodes of G will put into Q if output does not change then the fanout nodes are not put.

That is the basically that event driven portion and you see that there is no we have not considered any delay. So, as soon as G's input changes, we are evaluating it to see whether it is output going to change or not. So, that is why it is a 0 delay event driven simulation ultimately Q becomes empty that when we have simulated all the gates using this event driven policy for the particular test vector and when it is over then it will check the next vector and if it will pick up; take up the next vector try to simulate it and finally, when all the test vectors are over, it will come to the simulation will end and it will come to an end. So, we continue in the next class.