Embedded Systems Design

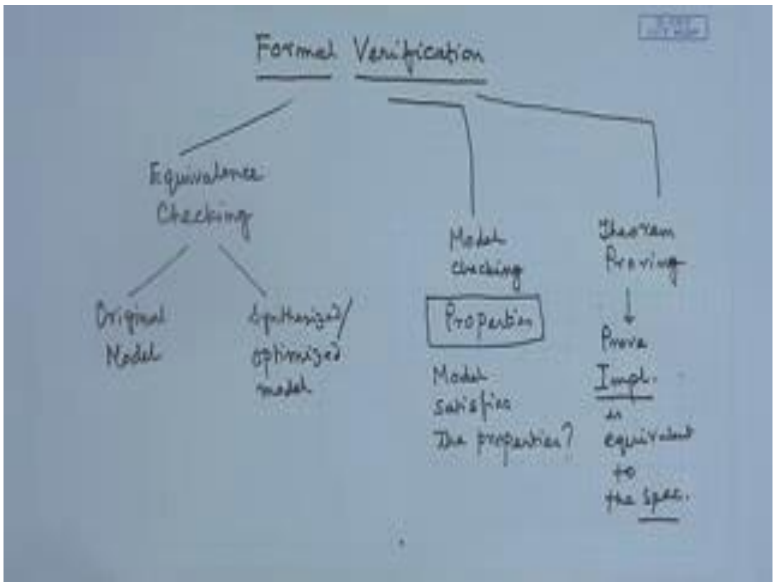
Prof. Anupam Basu
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 49
Formal Verification

So, in an earlier lecture, we have talked about simulation as a means of validating and verifying embedded systems. We have also seen what is mean by hardware-software co simulation where the hardware often represented in HDLs like verilog VHDL and the software represented in C or C++ can be simultaneously simulated and under two processes as two processes and who communicate among themselves is in inter process communication.

Now, simulation is of course very important and very much used in the industry. It has got its limitations because its coverage is doubtful I mean not doubtful the coverage is not 100 percent all the time and it is very expensive to get large amount of coverage it, takes more time. And still we will miss some errors depends on how accurate and how exhaustive we can make the input state vectors, based on that the coverage also varies.

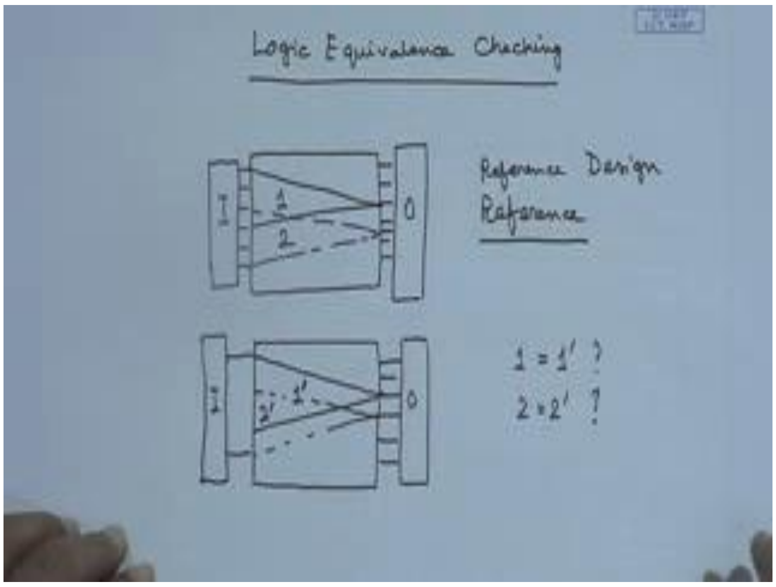
(Refer Slide Time: 01:46)


Formal Verification
Equivalence Checking
Original Model
Synthesized/ optimized model
Model checking
Properties
Model satisfies the properties?
Theorem Proving
Prove Impl. is equivalent to the spec.

Today we will be discussing about the other approach that is the formal verification approach. Now, formal verification actually comes in three different forms; one is equivalence checking, where we take two models that is the original spec model, I can say the original; let me say we take the original model, and the final model which can be the synthesized model or the optimized model. And try to find then equivalence between these two the original model can be the specification model, and then we want to see whether the specification model and the synthesized model are equivalent. Also it can sometimes to be done that are got some synthesized model and I run it when optimizer and I come up with optimized model. Now, I want to see whether the optimized model is functionally equivalent to the original model, therefore, there also will be carrying out the equivalence checking.

The other method is model checking, where we are given some properties some properties asserted as predicates in some logical form. And we want to see whether the model satisfies the properties whether the model satisfies the properties. And the third version is the theorem proving approach where we actually proves here we try to proves that the implementation I or let me write implementation is equivalent to the specification. Now, both of this the implementation here and the specification here both are represented in the form of some formal logic all right. So, these are the three different parts.

(Refer Slide Time: 05:46)



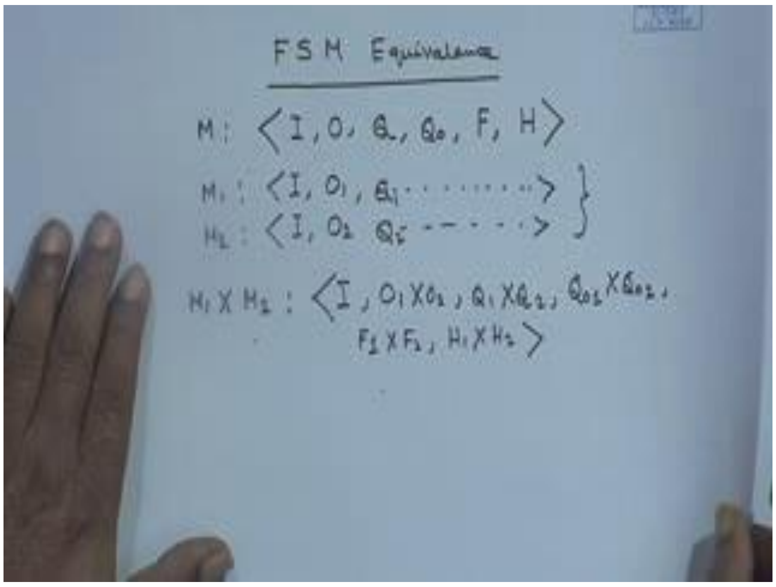
Now, let us first start with equivalence checking. Now, in equivalence checking we can try to find out the logical logic equivalence checking. Now, here I illustrate this using diagram. I have got the inputs I, and I have got some outputs coming out to this system, this is the outputs O. And say this output is generated by this logic cone. Now, what do we mean by this? Here this one I can consider that this is implementing some logic and that logic is built of different logical elements combination of sequential blocks. So, all these logical elements result in this output; and maybe another set of logical elements let me put another output here generate this output like that.

Now, this is the reference or the reference design, we often call it also the golden design, in case blocking here reference design. And here we can also have a synthesized design. So, this is a reference design, and which can be the specification also or it can be suppose the design has been achieved, now I want to optimize it; and by optimization, I have got another similar input and output scenario.

So, essentially again in the same way, I have got I am coming through two; now this contents of this logic cone, so let me call it 1 prime. And this one another logic cone, I call it 2 prime. This is might be different. You see here was some logic circuits using which are generate this I have optimize this, and again from the same set of inputs I am generating the output, so another logic cone.

Now, equivalence checking, essentially once to find out whether 1 is equivalent to 1 prime and 2 is equivalent to 2 prime all right that is my if these two are true then there equivalent. Now, there are different ways in which that is found out. Now, the content of this, so these are logic equivalent; here we are trying to find out just the logic that given this input this outputs should come, given this input this outputs should come. If that comes then this logic block this cone is equivalent to this cone that is as if I am doing just on some logic circuits.

(Refer Slide Time: 09:54)



FSM Equivalence

$M : \langle I, O, Q, Q_0, F, H \rangle$

$M_1 : \langle I, O_1, Q_1 \cdots\cdots\cdots \rangle$

$M_2 : \langle I, O_2\ Q_2 - - - - - \rangle$

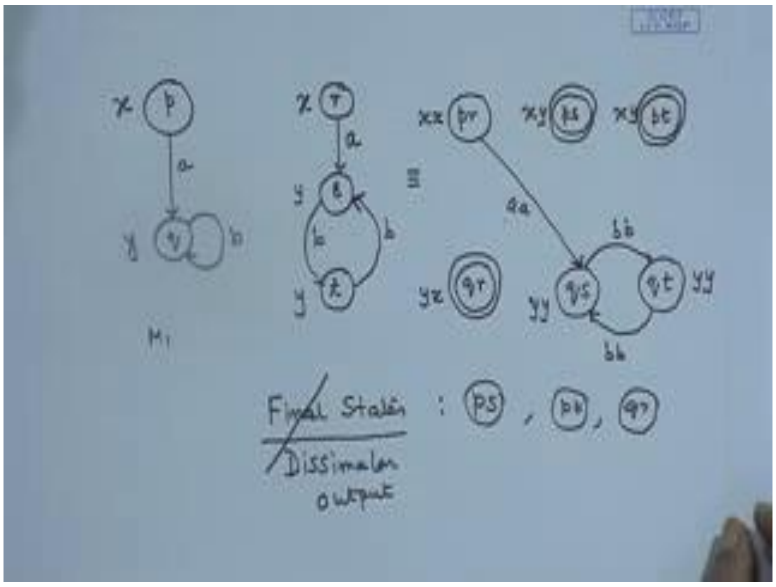
$M_1 \times M_2 : \langle I, O_1 \times O_2, Q_1 \times Q_2, Q_{01} \times Q_{02}, F_1 \times F_2, H_1 \times H_2 \rangle$

The other equivalence that we need to consider is the equivalence of FSM. Now, we know that we can define and FSM M to be a set of inputs, a set of outputs, Q is a set of states, there is some initial state Q 0, there is some functions F, which is a state transition function; and there is an output function, output function is mapping state to the output. So, that is a FSM. All of us know that a FSM is acts as a language acceptor, so I am not going into FSM in detail.

Now, suppose I have got two FSMs 1 M 1 and M 2. Now, these two FSMs, we will have since I am trying to find out the equivalence between the FSMs, the inputs of these FSM s should be same; otherwise how do I compare the equivalence right for the same inputs there should behave in the same the way. So, here there outputs sets maybe different. Similarly, all these functions and all these things can be different, but the key thing is that the input should be the same. Now, from these two, I want to make product FSM M 1 cross M 2 and what will that be, what will M 1 cross M 2 be, it will be the same input. The output set will be O 1 cross O 2. The state sets here it was Q 1, here it is Q 2, so it will be Q 1 cross Q 2; initial states Q 0 1 cross Q 0 2; similarly F 1 cross F 2 and H 1 cross H 2 that is my product FSM.

Now, so when we find out FSM equivalence, I am starting with two FSM say M 1 and M 2. And output, what is my output, what do I check for equivalence. What I want to check is does the same M 1 and M 2, do M 1 and M 2 give the same set of output for alls all the inputs which are same, all I mean for all the inputs that is what do you want to see. So, if effectively our point is if we have got to two FSMs, and we want to see their equivalence; that means that given the same input both this FSM should give me the same output. The reason why I came to this cross product is to explain one algorithm by which such equivalence can be tested.

(Refer Slide Time: 14:10)



Now, let me show that algorithm. This algorithm was by Devdas and Newton way back in 1987. Suppose, I have got an FSM p from state p, this is one FSM is coming to state q and that is happening on input a and it is producing the output x. Please note my notation, here what I am writing is the output x and as it comes here it is output y, and this transition is looping over here on b, this is let me call it M 1. And M 2, this was maybe this was my desire my specification. And ultimately what I got is something like this r, s and then t, this is on a, and the output is x; from s the output is y and as a o here I make a traverse along b, and I go back on b. So, here the output is y. Are these two equivalent, how do you find it out? The way is that suppose I form the M 1 cross M 2 I will now form M 1 cross M 2. So, each nodes, inputs will be the same outputs will be O 1 cross O 2 states will be Q 1 cross Q 2. So, here the states were p, q and here were r, s, t. So, my states will be p r, q s, p s, all those things right that is the Cartesian product.

Now, so my product one will be p r, q r. You can draw with me if you like, q s these two q s, q t and p s and p t. Now, I also put the outputs along with that. So, the outputs are for this the output is x x; for p s, the output is x y; for p t, the output is x y; for q r, q and r, it will be y x; here it will be y y q s; and this one will be y y 2. Now, where is the transition, transition is from p to q, there is a transition. So, my transition set between this is given by my transition function F 1 plus F 2 that means, it is a a same input passes this transition. Agree? Now, and there passing the same output, for same input a all right; now let me first completed q s to q t will there be an edge, q s to q t will there be an edge, yes.

Student: Yes.

Because of b, if I have b then we have go to q to q and; for b I also go from s to t, so b b. And from q, t to s, I have got b, so again by b b, I come here. Now and I have got no transitions here now what do you find from this.

Student: Sir, why do we do have other transition?

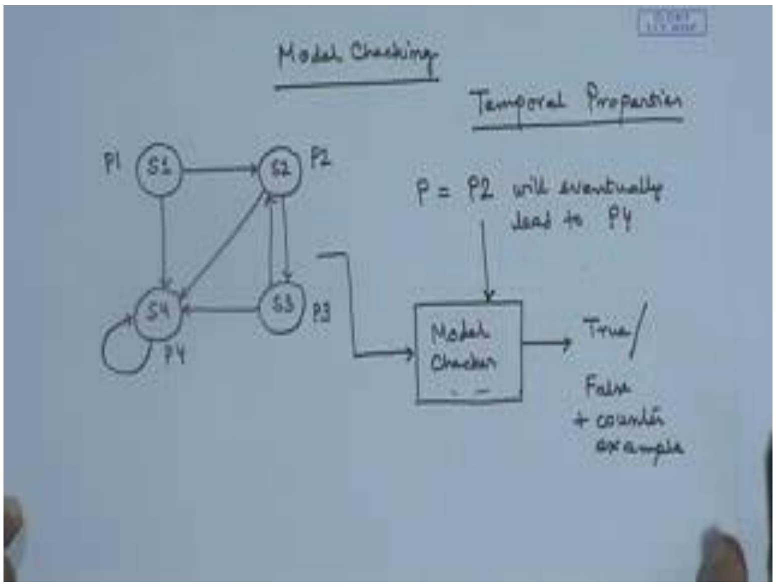
Show me the other transitions, because there are no transition between there is r s, between p to s there is no transition p s; p r is there, there r to s there is, so there are no transitions. If you look at this, we will not find any transitions. Now, the point is here we defined the outputs is a; declare different way of defining it, we define the outputs as the once which have got different outputs, sorry I am I am sorry here I will say call it the final states I will call the final states to be the states which have got different outputs. So, which are my final states, differing outputs, this one is the differing outputs one, this is another one, and this is another one. The name is the bit peculiar you can call it dissimilar output states all right dissimilar output states. So, they are this p s, p t and q r.

Student: (Refer Time: 20:48).

Exactly now these are the; now I would rather prefer my name instead of calling them final state, let us call it dissimilar output state peculiar.

Now, let us go back a second. What is my basic idea of equivalence between two state machines? For the same input the output should be same wherever, when I compose them wherever I get the outputs to be different there are my danger points, but as you can see there are not reachable. So, those will never be arrive that. I will simply arrive at only in the valid states which are got the same output. Also now you check that for the same input I am coming to this same output; for a, from x, I am coming to y; for b, from y I am coming to y, so that is it. So, these two are I mean, so this is proper equivalent use to FSMs equivalence. This is one way, there are other algorithms too, but this just to illustrate the idea of what is mean by FSM equivalence, this is what we present.

(Refer Slide Time: 22:19)



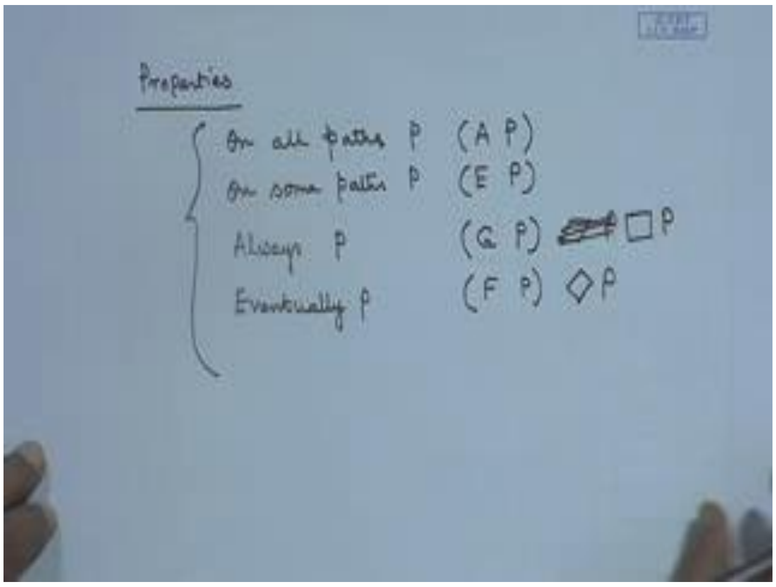
Next, we come to model checking. We have seen that we have got three different ways of formal verification - equivalence checking, and model checking and theorem proving. So, let us talk about model checking. So, suppose we have got machine consisting of this four states. So, we have got states transitions, here I am not showing in the labels, but these are there. Now, along with that something that we will introduce now are the temporal properties. Temporal properties are properties which are associated maybe with each of this states, or transitions, or overall I can present some temporal properties.

For example, if I considered, suppose here I have got some temporal property P 1 some temporal property P 2, P 3, P 4. And say overall I make a temporal property P, which says thus phenomenon P 2 will eventually lead to P 4; that means, some P 1 is some properties. If this properties satisfied sorry P 2, if P 2 is satisfied at any point of time then ultimately in this machine P 4 will also be satisfied. What is P 4, P 4 is another property.

Now, these properties are presented to the model checker. So, I have got the model checker here, where I am feeding this properties, and I am feeding this model and it is giving me the outputs which can be true that is this property is satisfied or it will say false plus a counter example. And that is very beautiful, because if it is satisfied I am told that ok that is satisfied that this circuit. So, here what are we checking, till now in equivalence checking we were given a model other specification and their design and we have trying to find out the equivalence between these two.

Here, what we are doing, we are doing something different here we are given some properties and the models. And our job is to see whether the properties are satisfied. For example, typically a property can be that in this circuit say the output of the JK flip flop will give a sequence 1 0 1 ultimately at some point of time it will give a sequence 1 0 1. The D-flip flop will always remain in the state of one for example, very in that case why do I put at D-flip flop, but just for the sake of example I am saying always it will be in the state of one. This property is it being satisfied by the other properties and this entire model that is being given.

(Refer Slide Time: 28:28)



Properties

On all paths P (A P)

On some paths P (E P)

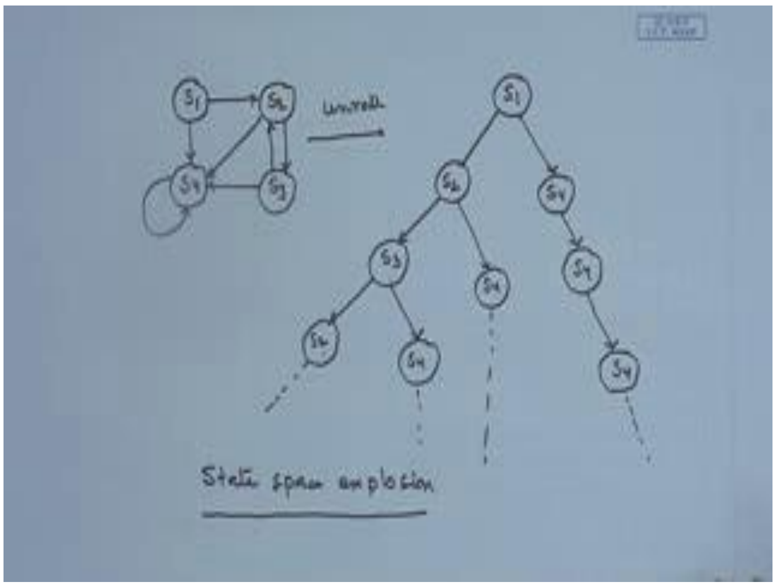
Always P (G P) $\Box P$

Eventually P (F P) $\Diamond P$

The typical properties are say for example, let us look at this diagram, and typical properties that are given in under this temporal properties are something like this. Say some property is true on all paths P that means let me denote as say P on all paths O will be true say P is some proposition or some paths P will be true. Now, this you can find those of you have done proposition logic, proposition predicate logic, you can immediately see this is the same as our universal and existential quantifiers. For all x P is true for all X there it was X some variable some here it is for all time P is true there exist some path or sorry here its x all right for all paths P is true there exist some paths for which P is true.

Similarly, we can have always P or this also told as globally. There is globally always, there are different notations there are some notations like this it is a very popular temporal logical notation. Eventually P that is shown the here the notation is F P typically in temporal logic will find diamond notation I am sorry this is a looking the same. So, these are some typical temporal properties which are specified over this.

(Refer Slide Time: 31:10)



Now, for example, how do we take that let us illustrate this with the same example that we have shown here. Say if I just quickly rework this. Now, this sorry there was another one if I unroll this it will be an infinite computation tree. What sort of tree it will be say start with S 1 any one I start as a route. So, from S 1 I can go to S 2 all right, or I could have gone to S 4. So, I want to making it a binary tree sort of thing. From S 2, I can go to S 3; and from S 3, I can go to S 2 or I could have gone to S 4; from S 3, I could have gone to S 4. So, it is coming to the same scenario. And from S 4, I can go ahead from S 2, I could have come to S 4 that was the possibility. And from S 4 I could have I mean that should be continues similarly S 1 to S 4, S 4 to S 4 in that way it can go on right something of this are.

So, in that way, now let us look at some property states. If I say on all paths S 2, on all paths S 2 let us see on all paths S 2 is not true right there is a the path there is S 2 is not true. So, immediately it can say that here is a path there is S 2 is not true. So, here we are start is no way if there was an edge from S 4 to S 2 then on all paths say could be S 2 possible. On some paths S 2, true; on some paths S 2; always something always if I say always S 4 succeeds S 2, is it true, no. So, which one is always true, which one is always true?

Student: S 4 succeed S 4 always.

S 4 succeed S 4 always, yes, anything else?

Student: (Refer Time: 34:41).

S 3 succeeds S 3 succeed S 2, but here what happened yeah this is true. Eventually, S 4, no it is not because ultimately if I follow this path, S 4 will be tree; if I follow this path now here also after S 2, S 4 will ultimately.

Student: (Refer Time: 35:09).

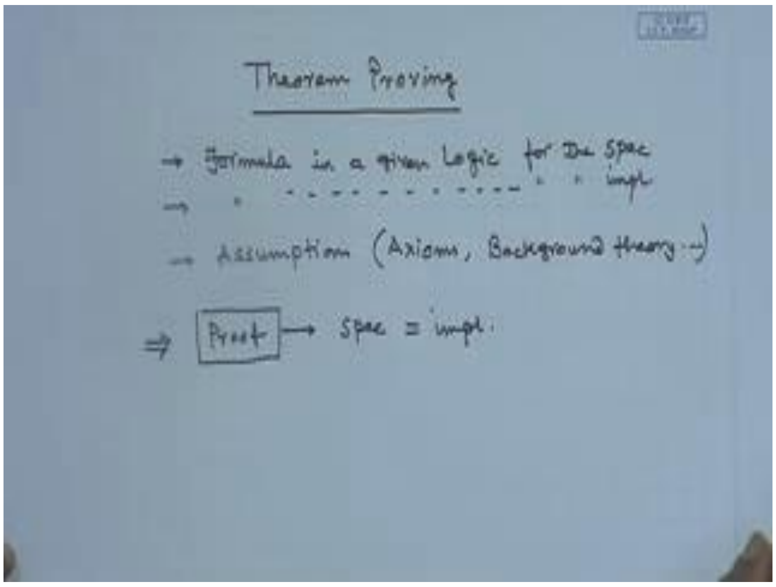
Yeah, it can where is my diagram no S 4 if we ever reaches S 4 it will be continue over there S 4 to S 3 it cannot go S 4 is becoming a sink here.

Student: (Refer Time: 35:26).

No, no that so I am saying this everything is not being reflected in this diagram in this example, but that is how we can say these properties and we will see whether this property is holding, and how do you see that by expanding this machine in to a computation tree. But what is the problem, the problem is it will lead to a state space explosion. So, it can lead to a; we cannot say eventually S 4, because as you can see here S 2, S 3 it can just cycle over here. Ultimately depending on whether S 3 to S 4 transition will take place or not that we do not know right, so that is why it will leads to some sort of state space explosion.

So, for that this is one approach in which model checking is done. And we look at different properties and analyze whether this properties are available or not. And since there is a state space explosion, we take recourse to several heuristics to proven that tree and see which are the properties, which part of the tree needs to be looked at for which properties etcetera.

(Refer Slide Time: 37:03)



Theorem Proving

→ Formula in a given Logic for the spec

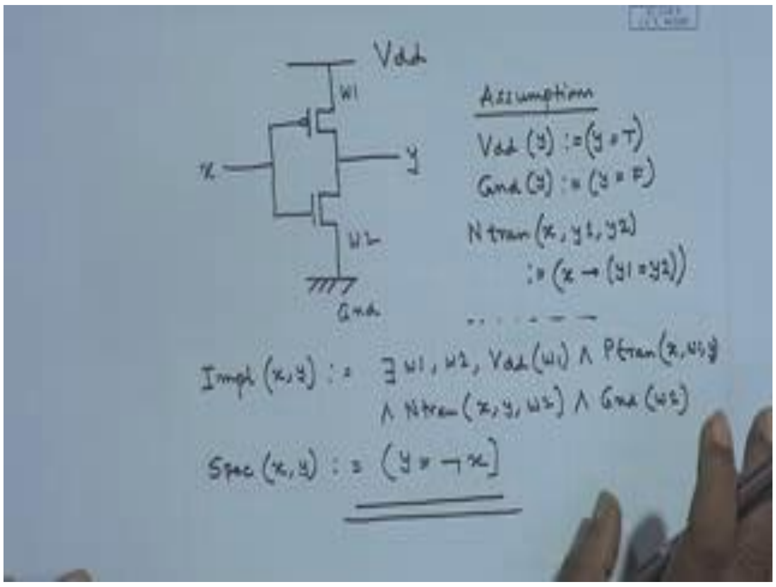
→ " - - - - - - - - - - - " " impl

→ Assumption (Axiom, Background theory ...)

$\Rightarrow$ $\boxed{\text{Proof}}$ $\rightarrow$ spec = impl.

Now, just to conclude, we will mention about I will just touch upon the theorem proving approach just for the sake of completion, as a third approach of formal verification. So, say for example, some formulas, we start with some formulas for the specification. Formula means well formed formula in a given logic, some given logic typical logic can be first order predicate logic, higher order logic or it can be temporal logic. And similarly formula in a given logic for the spec; and formula in a given logic for the implementation and there are assumptions about the particular domain which are typically the axioms of the background theory etcetera, all those things. The output that we want is a proof we want to find a proof as the output that proves that the spec is same as the implementation that is the theorem proving thing.

(Refer Slide Time: 39:00)



Vdd
w1
x
y
w2
Gnd

Assumption

$Vdd(y) := (y = T)$

$Gnd(y) := (y = F)$

$Ntran(x, y1, y2)$
$:= (x \rightarrow (y1 = y2))$

. . . . – –

$Impl(x, y) := \exists w1, w2, Vdd(w1) \wedge Ptran(x, w1, y)$
$\wedge Ntran(x, y, w2) \wedge Gnd(w2)$

$Spec(x, y) := (y = \neg x)$

Just quickly I will give you an example and that is again from electronics. So, we need not bother too much, but I know some of you are afraid whenever you see CMOS circuit, but say here is a circuit and I am giving some input y, and taking some sorry x some input x and I am taking some input y and there is w 1, w 2, and there is ground. So, this is a typical CMOS inverter. Whatever x is given, y will be a inversion of that.

Now, we can say that there are some assumption, this assumption can be say V d d y that means, it is y is true. Ground y, y is the output ground y is y is equal to false; that means, ground is there is zero level. N transistor, a typical N transistor has got x, y 1, y 2 and it leads to x is to y 1 is equal to y 2. In that way, we can similarly I can make something about the p transistor. Now, the implementation will be so I have got some assumption implementation of x y is something like w 1 and w 2, these are the paths V d d whatever is there in w 1 and P transistor x, w 1, y; and n transistor, you need not go inside this and just trying to show you how it looks like and ground w 2 that is my implementations. So, this entire thing this entire thing describes this implementation this circuit and the specification can be x, y is looks so simple y is not x. So, what we have to do given this assumption and this implementation, I have to do that derivation and prove that this is holding.

Now; obviously, is just to give your feeling I did not explain all these things. So, much, but just to give your feeling that even for an inverter description in this logical form is very complex turning out to be very complex and thus pure improving representations become quite complex. And therefore, the drawbacks of formal methods of course, there are many advantages, the drawbacks of the formal methods are for equivalence checking, you have to do a comparison between these two. Theorem proving is not finding so much space in the industrial yet because of too many reasons one of them is that most of the designers do not have enough background in formal logic, high order logic.

The models must be expressed in logic formulas as we have could see the simply turned out to be (Refer Time: 43:48) and there is very limited scope of automation that given this how do automatically generate this logical formula. For model checking the problem that we saw was the state space explosion problem and for equivalence checking you see that equivalence checking is much more accepted in the industry and we have got the algorithms to find out the equivalence.

So, there are lot of work going on in formal methods. And when we go for formal methods for hardware-software co design I mean then the thing becomes formal methods serving adopted for software verification as well. So, software verification hardware verification are gradually coming to at the same level, where the same setup tools can be apply to both. So, as the embedded systems are becoming more and more complex day-by-day, we need to have better verification tools, so that we can first verify the circuits about their real time properties, about their critical natures, before you go to the market and for manufacture.