VLSI Design Flow: RTL to GDS

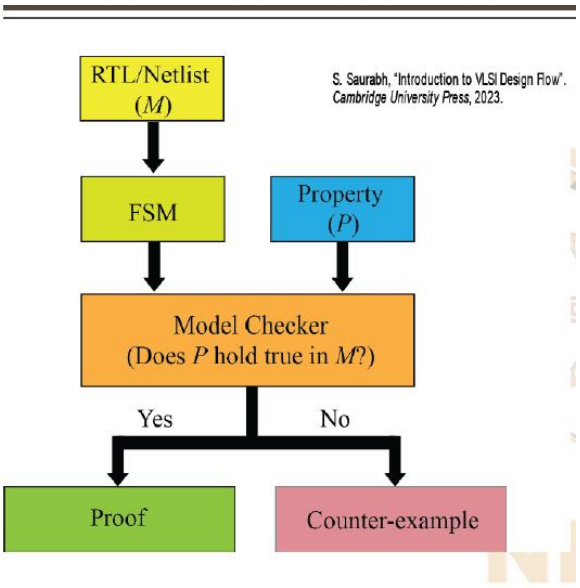
Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
IIIT-Delhi

Lecture 24
Formal Verification-III

Hello everybody, welcome to the course VLSI Design Flow RTL to GDS. This is the 19th lecture. In this lecture, we will be continuing with formal verification. In the earlier lectures, we had looked into two fundamental techniques that allow us to solve formal verification problems that are encountered in VLSI. And those techniques were related to representation of a Boolean function using BDDs and solving the satisfiability problem efficiently. Now, having looked at the fundamental techniques, now let us understand that what are the formal verification tasks that are performed in VLSI.

So typically, there are two fundamental formal verification tasks that are performed in VLSI design flow and these tasks are model checking and equivalence checking. Now, just to recap what is model checking? So, model checking basically verifies that for a given model or design whether the specification or a given set of properties are satisfied. And what is equivalence checking? So, the equivalence checking verifies that two representations of the same design exhibit exactly the same behavior or not. So, these are typically the two most popular most widely used formal verification task that are performed in VLSI design flow.

So, in this lecture, we will be looking at model checking and in the next lecture, we will be looking at equivalence checking. So, let us first start with the framework of model checking. So, in model checking what we do is that the inputs that we give to the tool is the given design either in the form of RTL or netlist and the tool internally models this design as an FSM or finite state machine. Additionally, we give a set of properties. So, the first input is the design, which is modeled as FSM and the second input is the set of properties that we want to verify, the property which should be valid or should hold for our given design.



So, the properties are not inferred by the tool. We need to provide the properties as an input to the model checking tool and given these two inputs what the model checker does is that it verifies that whether the given property let us say there is a property P, then whether this property P hold true in the model M or our design M. Now, if the property holds, then the proof of the correctness of properties established and if the property does not hold, then the model checking tool will give us an counter example and using this counter example, then we can debug our design and see that why the property which we were thinking should be valid for our design why it is not being valid. Now, what is the challenge in this task? So, the challenge in this task is that the model checker needs to verify that the given property is valid as the FSM or our design evolves through its various states. So, if given a design, the design is modeled as an FSM.

Now, FSM is finite state machines. There are lots of states in them and as we give input, then based on the input and the current state, the finite state machine goes to the next states and the finite state machine moves through various states during its execution. Now, the problem is that the model checking tool needs to verify that this property P should hold for that FSM for all the states. Now, given that our design contain n state elements or flip flops, the number of states that can be very huge it can be 2^n. It is exponential in the number of state elements.

And therefore, the model checking tool needs to verify that given a starting state, the property holds not only for the current state, but all the states that the finite state machine will go through and the number of states that are possible is 2^n. For example, you can consider say there are only 100 flip flops in our circuit, then the number of states that are possible will be 2^{100} and by going through each of the transition through each of the state,

it is simply impossible to verify that our given property is valid for our design or an FSM. So, this problem is known as state explosion problem meaning that there are lots of states possible for our circuit and we have to look for the validity of this property throughout the state space. Now, first let us look at how we give properties to the model checking tool. Now, to specify properties to the model checking tool, we need to specify them in what is known as temporal logic.

Why do we want to specify property in terms of temporal logic? Because system properties needs time related specification that is why it is tempora,l in addition to logic related expression for example, AND, OR, and Boolean expressions and those kind of things. So, we need both kind of specific definitions to be allowed to be existing in our properties that we want to specify. So, let us see a few examples then it will become more clear. So, a few examples of temporal properties are 'whenever a correct password is entered the door opens eventually'. So, whenever and eventually these are a kind of time related specification in this property.

Similarly, let us take a look at another property 'for traffic lights at a given post one of the red, yellow or green light should be always on'. Now, one of the red, yellow or green light it is a kind of Boolean logic. So, it is a logic kind of thing, this expression kind of thing is Boolean logic expression and the always is a kind of a time related meaning that is this property should hold always in time. So, another property is 'whenever a request is made by multiple requester it is never granted to more than one requester simultaneously'. So, whenever and never these are kind of time related definition or specification in the property.

Now, so the ordering of events in these properties are implicit. For example, 'whenever', and then, 'eventually'. So, eventually follows whenever. So, we are not saying that at this time, this happens; this time, this happens, but it is the ordering of event is implicit in these definitions eventually, never, always, whenever, etc. Now, how do we model these kind of properties or how do we give these properties to the model checking tool.

We typically use System Verilog Assertion or SVAs to model the properties and give it to the tool. Now, what is System Verilog Assertion? So, System Verilog Assertion is a part of IEEE SystemVerilog language. So, we in the earlier part of this course we had looked into Verilog. Now, the System Verilog we had seen that it is a superset of Verilog and the System Verilog assertion, SVA is a part of System Verilog and we can specify temporal logic using the constructs of System Verilog language or System Verilog assertion constructs. So, there are constructs for example, there is a keyword assert.

So, using the keyword assert we can specify property in our RTL. So, in our RTL at

various points, we can put assert and within this some properties and based on its construct, we write some specification for what the assertion should be or what the property should be. So, we embed these assertions or properties in the RTL itself. Now, as a result what happens? So, this assertions are checked of course, by the model checker, it will also be checked during simulation meaning that whenever we have made our design and inside this design we have written some assertion and we have also written a test bench which runs on this or provides input stimulus to this design during our simulation based verification. Now, during simulation if any of these assertions or properties fail then the simulator will say that this assertion got failed and we can understand that the property was violated in our design for this particular test bench.

So, when we embed System Verilog assertion then those assertions are checked during simulation, but there is a catch. The catch is that here the test bench needs to provide the stimulus and the assertion will fail only when we have a stimulus which is actually failing that assertion and as we have seen we cannot write a test bench or a set of stimulus which covers our design exhaustively. So, it is likely that this assertion can fail and based on that we can understand that this property has violated, but we cannot totally rely on this. In contrast what model checker will do? Now model checker will also take these properties and will do a verification of these properties without applying the test stimulus and that is why it is a formal verification. So, in this lecture we will understand how formal verification of properties is done by the tool.

So, in that sense model checker is more complete in verifying our properties. Now while specifying the properties, we can also specify some assumptions and constraints which will help the model checker to perform the property checking more efficiently. Now there are keywords in System Verilog assertion: ‘assume’ and ‘restrict’ and using these keywords we can give information to that model checker that what assumptions can be made by the tool and these assumptions are treated then as axioms by the tool and based on that property verification is done or there are constraints which exist for the input in that case the space where the model checker needs to look into that gets restricted and as such the property checking can become easier. So, wherever possible if there are assumptions in our design, if there are constraints in our design, we should add them in our design because it will help the model checker to efficiently perform model checking. Now what is the technique of doing model checking like how the tool internally does the model checking let us understand that and let us understand why it is a formal method of of testing or checking our design, verifying our design.

So, the primary difficulty in model checking is because of state space explosion that we just saw. So, earlier what researchers or VLSI designers tried was, we enlisted various states and and using graph traversal, given a starting state, various states were traversed

and then the properties were verified based on if a state is reached where this property is not valid and so on. So, this was the earlier approach, of course this approach of enumerating states and representing in graph and then verifying this does not work meaning that it was not scalable to the level that it could be useful for VLSI design application. But in early 1990s what happened is that there was a breakthrough. So, the the breakthrough came when we started solving this property checking method using a technique which is known as symbolic state space exploration.

So, earlier this technique was using BDDs and later these techniques were further improved by using the SAT based techniques. So, we will be looking at how BDDs can be used to traverse the state and verify the properties. Now, BDD based model checking uses characteristics function to represent states. So, let us understand what is a characteristics function. Now, consider an FSM and it has got a finite set of states and let us represent the set of states as Q.

So, the Q is the set of states for a finite state machine and consider a subset of states. Suppose there were say 1000 states out of that we took say 20 states. Now, these 20 states, the subset of states out of all states is represented by A. So, now, what we can do is that we can represent this A by a Boolean function f and how do we represent this set A. We represent it using Boolean function f such that if any state x which is an element of Q, if it is a member of the set A then f(x) should take a value of 1. So, if we define a function f which is known as the characteristics function for the subset of states A and we define such that f(x)=1 if and only if x is an element of A meaning that if this is the set Q, it represent all the states, we took a subset of state, A.

Now, for the states existing in A, f(x) will be 1 and for all states which do not belong to A, their f(x) will be 0. So, this is the definition of characteristics function. So, let us take an example and see how we can formulate or write a Boolean function for characteristics function. Now, consider an FSM of 5 states and let us call these 5 states as s0, s1, s2, s3, and s4. Now, let us for example, let us first represent these set of states using 3 bits since there are 5 states will need at least 3 bits to represent these states.

Let us call these 3 bits as x2, x1, and x0. The bits that are used to represent states those are known as state bits. So, in this case there are 3 state bits and we denote this 3 state bits as x2,x1,x0. Now, we will do encoding of these 5 states using these 3 bits, there can be multiple ways to do the encoding. We take one example just for the sake of illustration. Let us say that we have encoded s1 to s4 as 000 for s0, 001 for s1, 010 for s2, 011 for s3, and 100 for s4.

So, this is how we have encoded the given set of states. Now, in this representation, we

can represent the subset of states A. Let us consider a subset of states, let us consider that there is a subset A, which contains only 3 states out of 5 states s0, s2, and s4. Then how can we represent this in terms of characteristics function of A? We just take the min term corresponding to each of the state. s0 for that we can take if the value is 0, bit is 0, we take it as in the complemented form and if it is 1, then we take uncomplemented form.

So, for s0, all are 0. So, we take x2'x1'x0'. So, this is the min term corresponding to this state. Similarly the min term corresponding to this. Now, similarly min term corresponding to s4.

So, s4 is this. So, here we have taken as uncomplemented because this is 1 and then 00 these are complemented. So, we take x2x1'x0'. So, this is how we represent the subset of states using the characteristics function. Now, here this is the simple representation of the characteristics function, it is in terms of min terms and we have seen that in the earlier lectures that min term representation of the function is similar to a truth table and the size requirement is exponential meaning that it is not compact representation. But for any Boolean function, we can represent that Boolean function in terms of BDDs also and what we have seen in earlier lecture is that BDD is more compact and also canonical representation of a Boolean function.

So, we can represent the characteristics function of even very large set, in which there are say lots of elements maybe $2^{n,}$ in that case also, we can represent all of the states using very compact BDDs. So, we can represent a large set using its characteristics function with the help of compact BDDs. So, this is a very important tool. So, the number of states can be very large, but its representation in terms of Boolean function can be very very small and that we can represent for example, if we take the representation in terms of BDDs. Now we can also compute the transition from a set of states to another set of states very efficiently using BDDs.

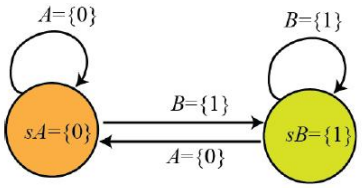
So, now how can we do that? So, we need to use what is known as a transition relation. So, consider an FSM with set of states Q. Let us take an FSM which has got set of states which we represent as Q and let us denote the set of inputs as I. Then for this FSM, there will be a next state function. Let us represent it using delta ($\delta(x,i)$).

BDD-based Model Checking: Transition Relation

- Consider an FSM with set of states Q.
- Let us denote the set of input values as I.
- Let us denote the next-state function as $x' = \delta(x, i)$ for $x \in Q$ and $i \in I$.

Transition relation for an FSM:

- The transition can be defined using a transition relation $T(x, i, x')$ such that $T(x, i, x') = 1$ if and only if $\delta(x, i) = x'$.



S. Saurabh, "Introduction to VLSI Design Flow", *Cambridge University Press*, 2023.

x	i	x'	$T(x,i,x')$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- We can represent a transition relation $T(x, i, x')$ compactly using BDD
- Subsequently, we use the transition relation in BDD-based model checking

So, this delta, δ(x,i) is a function which takes the current state x and the current input i and it gives us the next state. Now we can represent this FSM transition also in terms of transition relation and how can we do? So, the transition of the states in an FSM can be defined using a transition relation T(x,i,x') which takes input as the current state, input and the next state. Such that if the transition is there, this transition exists in the FSM for a given x, i and the next state then the value of that transition relation is 1 and if does not, then its value is 0. So, T(x,i,x')=1, that is,transition relation takes a value 1 if and only if there is a transition in the next state function from the current state x to the current input i to the next state x'.

So, let us take an example then it will become more clear. Suppose this is our given FSM. So, this FSM has got two states sA and sB we represent it as 0 and 1 and suppose there are two inputs A and B, we denote A as 0 and B as 1. Now for the sake of clarity we have not shown the output function, we are not showing any output, we are just showing the transition, given in current state and the input, how the next state is reached. Now in this FSM there are four transition 1, 2, 3, 4 represented by the arrows.

So, if we draw the transition relation for these four transitions, will have 1 and for all other transitions, the transition relation will have a value 0. Now this is the transition relation now what we said that transition relation takes input as the current state. So, this is the current state, this is the current input, which can take a value 0 and 1, current state also can take a value 0 or 1 in this case because these states are represented by only one bit. So, this first state sA is represented as 0 and sB as 1 and x' is the next state. Now we can see in this state diagram that if the current state is 0 and the current input is 0 that is A, the next state is the same state sA which is 0 and so this transition relation exists in the

FSM therefore it takes a value 1.

Similarly if the current state is sA represented by 0 and the current input is 1 meaning that we have applied B then it goes to sB or the next state is 1 and therefore this row has got the value of T as 1. Similarly for the other transition, this transition and we have this this entry and this entry. Now the other zeros are for the transition that do not exist in the table in the FSM. For example, from the state 0 and input 0, it never goes to next state B and therefore this entry is 0 and this is how we build the transition relation for a given FSM. Now we can represent a transition relation also very compactly using BDD.

So this is what we have shown is a truth table of the transition relation. Now for this truth table, we can always represent this in terms of BDDs and it can be represented many a times compactly and subsequently we use the transition relation in BDD based model checking. So once we have the transition relation, then we use the transition relation which captures the transition of the FSMs for the subsequent processing in model checking. How that is used we will just see. So the key or the crux of BDD based model checking is the computation of image and preimage.

So let us understand what is image and what is a preimage. Now image of a given set of state S is the set of state S' that can be reached in one step from S. So suppose this were the set of states S. Now from this state, this is not one state, this is a set of state, there are multiple states here. Now from this state in one transition meaning that we go from one node to the other node in the graph, in the FSM representation, there is only one edge from one node to the other that is what we mean from one step from S, we can get to another set of states. So those set of states are known as the image of this node that these two sets S and S' may overlap also meaning that from S then in one step we can again reach to the same states that we started with. So there can be overlap. So this is the set of state S that is given and S' is the image of S meaning that S' or the set of states which exist in S', those can be reached in one step from any state that is an element of S. So image of a given set of state S, so we are talking about set of states, note that we are not talking about one state.

Given a set of states in one transition, the set of states that can be reached those are known as the image of a given set. So we denote the image computation as S'=image(S,T), where S' is the image. So which will be the set of states that will be reached in one step from a given set of state, it will depend of course on the transition table or transition function. So T comes as an input to the image computation. So given a transition relation T and the set of states, S' is what is determined by the image computation and that S' is the set of states which can be reached in one step from a given set of states.

Similarly we can define preimage of a set of state S', so we are given S' as the input. Now from this, for this set of states, what are the set of states S from which we can reach S' in one state. So the preimage is the opposite part meaning that we are given a set of states now from which all state we can come to this set of states, those are preimage of the given set of states. So the preimage of a set of state S', so we are given S', is the set of state S from which we can reach S' in one step and we represent this set of state S the preimage as S=preimage(S', T) because it again depends on the transition relation. Now for a given set of states we can compute image and preimage very efficiently using BDD.

So this is the crux or this is the reason why we can use BDD for symbolic state traversal and property verification. Why? Because the image and preimage can be computed using BDD for a given state transition relation T very efficiently. So BDD based model checking relies on this computation. Now let us look at how we can use the image computation or efficient image computation to compute the set of reachable states, given a starting set of states and what is meant by reachable set of states. So suppose we are given a set of states S0, then from a given set of states, the reachable set of states are those states which can be reached in any number of traversal. So image was related to only one step, in image computation we were allowed to traverse only one step but when we are computing reachable set of states, we are removing that constraint.

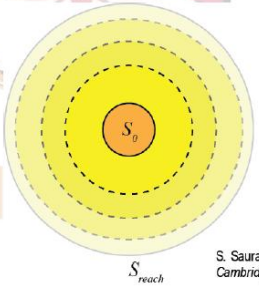
So reachable set of states are those states which can be reached from a starting set of states in any number of traversals of the FSM. So let us look into a pseudo code of how the reachable set of states can be computed from a given starting set of states S0 and the given transition relation T. So this pseudo code that we will just see will return the reachable set of states S_{reach}. So we start initially, we say that all the states that we start with, S0, those are reachable from itself. So S_{reach} is initially assigned S0 and we say that all the new set of states that are being reached is same as S0 initially and we keep a counter k to see that how the iteration progresses or how the set of states are expanded.

Input:
- Given starting set of states S_0
- Transition relation $T(x, i, x')$

Output:
- Returns reachable set of states S_{reach}

1: $S_{reach} \leftarrow S_0$
2: $S_{new} \leftarrow S_0$
3: $k = 0$
4: while ($S_{new} \neq \{\}$) do
5: $k \leftarrow k + 1$
6: $S_k \leftarrow Image(S_{new}, T)$
7: $S_{new} \leftarrow S_k - S_{reach}$
8: $S_{reach} \leftarrow S_{reach} \cup S_{new}$
9: end while
10: return S_{reach}

- The set of reachable states S_{reach} until no more new states are discovered
  - Algorithm is said to have attained a fixed point.



- States represented compactly as characteristic functions using BDDs
- Canonicity of BDDs eases manipulation
- Pre-image computation: set of all states from which S_0 can be reached.

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

So initially we keep the value of k=0 and then what we do is that from starting set of states S0, we go on increasing the set of states which are reachable from S0. So this reachable set of states goes on expanding until no more new set of states are found and that is why we have a while loop which continues while S_{new} is non-empty. When it becomes empty then this loop will terminate and we will get the set of reachable states. So initially, we increment the counter and then we compute given the new set of states what are the next set of states that can be reached S_k we can compute it using the image computation and the transition relations. Now this S_k will also contain some of the previously reached state.

So from this S_k, we remove the already reached state that will give us the set of new states that were now discovered in this iteration and we assign it to the set S_{new}. And then to understand this process that how the set of states are expanded we can understand this figure. So we start with S0 those that contains set of states that we are given. And in first iteration we say that from S0, we compute the image meaning that the set of states which are reached in one step from this S0 and suppose these are the set of states which are reached from S0. Then we say that out of these reachable state, we remove already reached state that was S0 and these are the set of new states, this is S_{new}.

And then this loop goes on and then in the next iteration from this S_{new}, using image computation we again find next set of states which is reached from S_{new} that is done in this state. We get next set of states which is found for S_{new}. Now out of them some of them will be already reachable so we remove that and then the other that will be left will be the new set of states. So these were already reached and in the next iteration these are now S_{new}. And then what we do is that now we say that now reachable set of state has

become this much.

And then in the next iteration using image computation we will get next new set of states and so on. Now if we go on expanding the set of reachable states what will finally happen? Now we are expanding set of states for an FSM. FSM means that the number of states in this machine is finite and therefore, we cannot continue infinitely. Finally what will happen is that the set of reachable states will be reaching a point in which no new states are being discovered and when that happens, we say that the algorithm has reached a fixed point. And once the algorithm reaches a fixed point then no more new no more new set of states are being reached and then this S_{new} will become an empty set and the algorithm will terminate.

So in this manner from a given set of state S0 and using efficient image computation we can determine all the reachable states from a given set of states. Now states are represented compactly as characteristics function using BDDs. So note what we have done we have not said that from this state let us traverse to the next state and so on. We are not doing a graph traversal state wise. What we have done is that we are symbolically traversing set of states and set of states are being represented using characteristics function and all these operations are being performed on the characteristics function of sets.

And these characteristics functions can be, though the number of element in this set can be very huge, the characteristics function will be compact and those can be represented as BDDs. And also since we are using BDD, comparing set and performing operations or comparing two functions becomes very easy. Why, because BDD is a canonical representation if two functions are equal their representation or their pointers will also be and that is why this state traversal or set of state traversal using symbolic traversal of set of states using BDD is efficient. Now in this algorithm instead of image if we put preimage then what will happen? Then what we can say is that we can compute the set of states from which a given set of states can be reached. Now having understood the reachability analysis or how the set of reachable states are computed using BDDs let us understand that how it is employed in model checking.

So, suppose it is required to check whether a state satisfying a Boolean function P is reachable from a given initial state s0 we are given a initial state s0 and we are asked the question or a model checker is asked the question that whether a state satisfying a Boolean function P or set of states that satisfies a Boolean function P is reachable from a given initial state. So, let both the states and the Boolean function P be represented in terms of state bits. So, we are saying that P is also a function of state bits x0, x1 and so on and we are representing the set of states also in terms of these state bits. Now a model

checker considers a set of states S_P for which P holds. Now we are given an initial state S0, this is one state. Now we are asked the question that whether a state is reached in which P holds.

So, model checker considers a set of states S_P for which P holds and then it tries to find that whether this set of states can be reached. Let us represent the set S_P using a characteristics function, CF_{SP}. So, let us represent this set using a characteristics function and we are representing as characteristic function S_P and for this state, P holds for all states in this set, the property P holds. Now but what we find that the characteristics function CF_{SP} is nothing but P, it has to be P. Why that is the case we will see. So, what we are saying is that the characteristics function of the set of states for which P holds is nothing but P. Note that we have represented both P and the set of states using the same state bits.

So, why this is that the characteristics function of the set of state that is CF_{SP} will be equal to P let us understand that. So, consider state x, for which P holds that means that P(x)=1.If P holds then for a given state x that means that P(x)=1 by definition of what we mean by holding true for a given state. Now x if it holds then it should also belong to S_P, this set S_P. So, x should be here. Hence $CF_{SP}(x)$ must be equal to 1 for this x.

So, for the given x, this characteristics function must evaluate to 1. So, what this means is that if P(x)=1, it implies that characteristics function $CF_{SP}(x)$ is also 1, this is one way implication. Now consider another state y, for which the characteristics function gives us a value 1. So, $CF_{SP}(y)$ evaluates to 1 therefore, y belongs to S_P. For a state y, if characteristics function gives us a value 1 what it means is that y belongs to the set and therefore, y belongs to S_P and if it belongs to S_P then P should also hold for it and therefore, P(y) must be equal to 1. So, what it means essentially is that $CF_{SP}(y)=1$ implies that P(y)=1. Now using these two statements, we can say that these two are equal, CF_{SP} it must be equal to P.

So, what we have proven now is that the characteristics function of the set of states for which P holds is nothing but P. So, now we have got the set of states. Or given a property P, which is represented in terms of state bits, we know that the set of states for which it holds is nothing but P. Now using preimage computation we can determine that what are the states from which we can reach the set of states P. So, given the property P, we know that this is the characteristics function for the set of states for which P holds is nothing but P and we are given a starting state s0. Now if we ask the question that give me the set of states from which the P is reachable it will expand this and we get this set.

Now if s0 is within this set then we can say that this property holds and if it is outside

this set we can say that it does not. So, using preimage computation we can determine the set of states S_{reach}'. So, this is S_{reach}', this big set is the set of all reachable states from which S_P can be reached. Now if S_{reach}' includes initial state s0, given property holds and if S_{reach}' does not include the initial state s0, the property does not hold. So, this is what is basically the mechanism of property checking or model checking using BDDs.

Now what could be the limitations of this method of checking or checking the property. So, in the worst case the size of BDD can be exponential. We had seen this in the earlier lectures when we were discussing BDDs we saw that for some function it will always be exponential for some function it depends on the variable order. So, what it means is that when we are trying to represent this characteristics function and the transition relation using BDDs, sometimes it may happen that the size becomes very big and the BDD explodes and in that case we will not be able to verify our property. So, a BDD based representation of transition relation can blow up with an increase in the number of state bits and and if the number of state bits increases or number of variables in the Boolean function representation that is growing and therefore, the chances of BDD exploding or becoming out of memory becomes greater as the number of state bits increases.

So, in that case what we can do we can try different variable order. So, in the tool the model checker internally what it can do? If for this kind of variable order, some problem is there, the BDD is growing exponentially or it becomes very big then probably reorder the variable order and probably the problem can be solved. So, that is one way to deal with this problem. The other way is to give manual intervention. So, when as a designer we find that our tool is not or the model checker tool is not able to solve or saying that in case the property is not being verified by a tool, typically the tool will say that the property P is not verified or it is a case of unsolved problem. So, the tool cannot say whether the property holds or it is not even able to refute the property. In that case the tool will report that this property, the tool cannot say anything about or cannot infer anything about this property and it is an unsolved problem. When that situation arises then as a designer what we can do? So, we can apply some constraints to our problem. If we know that some variables cannot take some values and so on, then we can provide those constraints or if there are some assumptions or similar kind of thing we can provide some hints to the tool which can help solve the problem.

So, we simplify the problem for the model checker and make it solvable. That is one way as a work around we can try this by looking. So, we as a designer, we know that some constraints are there or the tool can make some assumptions about the inputs and so on we should give that because it helps model checking tool to simplify its internal problem formulation and as such it may not go out of memory or it may give you the answer in lesser run time. The other way to do a model checking is SAT based model check. So, it

in some sense avoids the problem of memory blow up of BDDs. So, one of the way in which SAT based model checking is done is finding the counter example for a given property.

So, the approach that is used by the model checking tool is to obtain a counter example of a finite length. Length here means the number of clock cycles from the initial state say in 10 clock cycle can I get a counter example. So, what we mean by counter example is that suppose we are giving a property that given a wrong password the door should remain locked. So, suppose this is the property then what would be a counter example? The counter example could be that given the password is wrong and the door opens.

So, that is a kind of counter example. So, the tool will try to formulate a Boolean function Φ_n, n is the number of clock cycle or the length for which the tool is trying to check. So, that the model checking tool will formulate a Boolean function Φn using the given circuit or given FSM and the given property such that the function Φn is satisfiable if and only if a counter example of length n exists. So, it will first try to formulate a Boolean function Φn where n is the length or the number of clock cycle. That function Φn will be true or that will be satisfiable only if there exist a counter example of length n and by then invoking the SAT solver it can see that whether this function Φn is satisfiable or not and based on that it can come to a conclusion that if it is satisfiable then we have found a counter example and that will be reported to a designer and then designer can understand that why a given property failed. So, this type of model checking is known as bounded model checking because here in this case, we are allowing the clock cycle to be bounded by n, it is not more than n, it is less than n. So, therefore, this is a bounded model checking. It is a simpler problem in that sense than what we solve in using BDD based model checking.

So, typically we carry out bounded model checking or BMC iteratively by incrementing n. We start with n=1, 2, 3, 4 and so on. So, we go on doing it. So, it continues until we have found a counter example say we went from 1 to say 2, 3, 4, 5 and at the fifth clock cycle we found a counter example. We are done, we got a counter example, now we have to fix the problem in our design or we have to at least analyze what is there in our design. Now if we go on increasing n what will happen is that we will reach some stage where the problem, this Φn function becomes so complicated, the number of variables and clauses in this function are so many that the SAT solver, even though it is efficient, is not able to solve it. So, the problem becomes too complicated to be handled by the SAT solver in that case we cannot do anything, we cannot progress further with increasing the number of clock cycle.

So, the mechanism of this bounded model checking is to first derive this Φn, we unfold

the behavior of the system one clock cycle at a time. So, we start with 0, then clock cycle 1, then what happens in the clock cycle 2 and so on. So, we unfold the behavior of the system one cycle at a time using the next state function of the FSM until it reaches the n clock cycle. So, for one clock cycle, two clock cycle, three and so on up to n, it will unfold the behavior of the system. And then it will find Φn which is nothing but a logical conjunction of clauses obtained from three different entities.

First is the initial state from where we are starting, the second is the system behavior obtained from the next state function. So, this is what we had said, we unfolded the behavior of the system, from there we will get the system behavior obtained from the next state function and then the Boolean expression that evaluates to one for a counter example. So, it will be derived from the given property. So, given the state and the evolution of states using the next state function, evolved up to n clock cycle and the given Boolean function, it will formulate Φn and then invoke the SAT solver to see whether it is satisfiable. If it is satisfiable we have found the counter example otherwise we can say that till clock cycle n, there does not exist any input combination that will refute this property P.

Beyond n clock cycle we cannot say anything, but till n clock cycle, we can say that there is no input sequence which will actually refute this property P. In that sense it is still a formal verification technique because it is not taking the sequence of inputs from the designer or we are not giving as input, it is considering all inputs, the complete/ entire input space, but it is only unfolding up to n clock cycle and therefore, in that sense it is incomplete. So, the merits of model checking using this SAT technique or bounded model checking is that it avoids the problem of memory blow up that may happen in the BDD based model checking. We are not representing transition relation and doing those kind of image computation and those kind of thing and therefore, memory blow up problem can be avoided.

So, next state function grows linearly as the BMC traverses the next state in each cycle. So, if we go from say clock cycle 10 to 11, the number of variables that will increase in the evolution of states that will grow linearly rather than exponentially and therefore, the chances of memory blow up is much lesser in bounded model checking. But the problem for the SAT solver that increases exponentially. If the number of variables is n and if we increase to n+1, then the solution space where the function can be satisfiable that is increasing exponentially and therefore, the problem becomes more complicated for SAT solver and will take more run time. So, it exploits the power of the SAT solver since the SAT solver has becomes very efficient over last few years. The power of SAT solver is employed efficiently in bounded model checking. However as we discussed, it lacks completeness meaning that it gives an answer only up to n clock cycle beyond that we

cannot conclude anything using bounded model checking.

So, where is basically the application of this bounded model checking. So, the application is in practice to quickly find bugs which we cannot as a designer find easily, for bug hunting. So, why it will be useful because, see if we say that we have done BMC up to say 5 clock cycle. Till 5 clock cycle it is actually considering all input space for a given property. So, if there is a failure in 5 clock cycle we will get a failure. So, as a designer, we need not give a stimulus to the tool, to the simulator to give an answer for that. So, a bounded model checking can help a lot in finding bugs which are not easy for us to comprehend or come up using our intelligence.

So, in that sense BMC is a very good tool for verifying property of our design. So, if you want to look further into the topics that we discussed, these are some of very good references. Now, to summarize what we have done in this lecture is that we understood that the application of formal verification technique in VLSI is in 2 major areas. The first is model checking or property checking and the second is equivalence checking. So, we saw the model checking in detail in this lecture and in the next lecture we will be looking at the equivalence checking in detail. So, thank you very much.