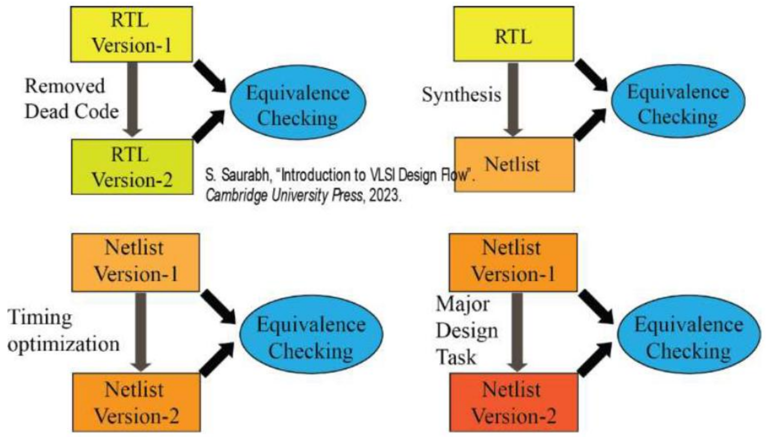
VLSI Design Flow: RTL to GDS

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

Lecture 25
Formal Verification-IV

Hello everybody, welcome to the course VLSI design flow RTL to GDS. This is the 20th lecture. In this lecture we will be continuing with formal verification. In the last lecture, we had seen that there are two types of task that are routinely performed or two types of formal verification task that are routinely perform in VLSI design flow and those are model checking and equivalence checking. So, in the last lecture we had looked into model checking in detail and in this lecture we will be looking into the equivalence checking. Now, let us first understand where to use this equivalence checking and why is it required.

So, we have in the earlier lectures looked into that during VLSI design flow, a design is transformed multiple times and during this transformation an error can be introduced in the functionality and these error can be because of human error or the error in using a tool or miscommunication and there are many other reasons why the functionality of a design can change during the transformations. But ideally what we want? We want that the functionality of a design as it was coded in RTL that remains intact throughout the VLSI design flow and to ensure that what we do is that we carry out equivalence checking. So, a few examples where we carry out equivalence checking is shown in this slide. So, if we have say an RTL which is our version 1 and we made some change in the RTL.



We said that this part of my RTL is not doing anything so this is dead code and we remove that code, we got another version of an RTL. Now, this change was done manually and maybe there was some very innocuous error introduced in this removal of a code. There may be some important statement which was there, which was doing some task and it was not really a dead code and we accidentally removed that. Now, how to catch that problem? So, what we do is that we take this RTL, the design which is the RTL version 1 and then RTL version 2 and do an equivalence checking in terms of functionality. And this equivalence checking that we are considering now is a formal method meaning that we are not applying any input and checking that these 2 models are equivalent functionally. But using formal methods we are checking that these 2 models are having the same functionality.

The another example is that we have got an RTL and we had discussed logic synthesis. At least 2 task we had discussed in the earlier lecture: RTL synthesis and logic optimization. We got a net list it may be in terms of generic logic gates or in terms of standard cells. So, after synthesis we got another representation of our design, before synthesis, we had an RTL. Now, are these 2 models equivalent. So, these 2 models ideally it should be equivalent.

So, we are not saying equal, it is equivalent. Equivalent why we are saying is that in RTL, there may be situation in which a variable y is assigned a value, a don't care x. And logic synthesis, what it might have done is that it has assigned a value of this as 0 or it might have assigned as 1 because it is free to do anything if the value of y was x. So, during synthesis some refinement might also be done. Refinement from refinement from x to 0 or 1. And therefore, these 2 models are not equal in that sense, but they are equivalent in meaning that the output response will be as per desired functionality.

So, during synthesis those kind of refinement and of course, logic optimizations are done. Now, what if the tool did not perform it correctly, there was a bug in the tool. We need to catch those situation and to catch that we need to do an equivalence checking between the RTL and the generated netlist. The third situation can be that we have got a netlist version 1, but it was having say timing problem meaning that the delay of some cells were much larger than expected. We change the logic to get a better delay.

Now, if we have changed some logic or we have inserted or removed some cell then we have made another netlist and once we have got one netlist and we have got another netlist, functionally these 2 netlist should be exactly same. Though the delay profile can change by timing optimization, the logical functionality of these 2 representation should still be same. To ensure that we need to carry out equivalence checking. Similarly if in the physical design, we have got one netlist version 1 and then we carried out some steps like for example, placement or say floor planning or we did routing or maybe we have done some buffer insertion, changes in clock tree or some some kind of modification in physical design. If these are major design task then we need to ensure that these 2 representation are indeed equivalent by doing a thorough equivalence check and we carry out this thorough equivalence check using formal tool and how we do that we will just in this lecture. Now, the equivalence checking are basically of 2 types one is sequential equivalence checking or SEC and the second is combinational equivalence checking or CEC.

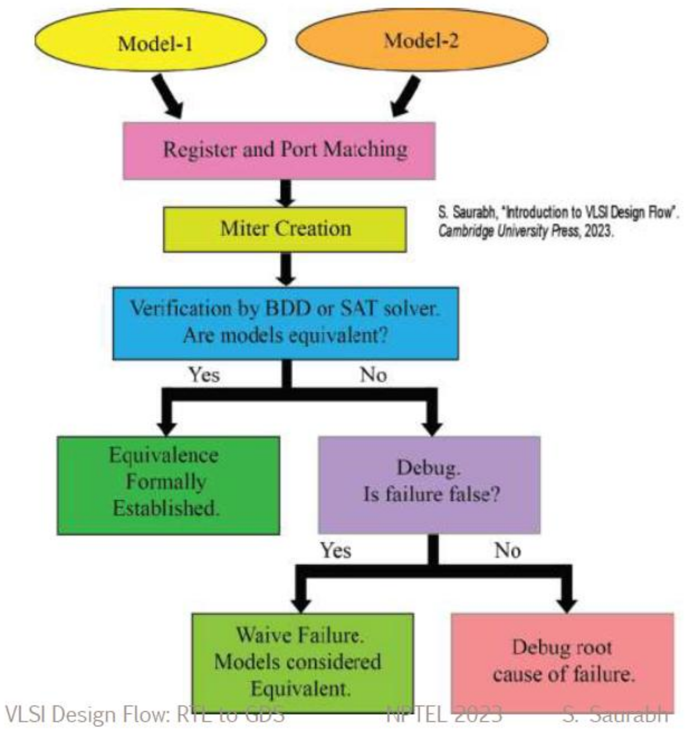
Now, what is sequential equivalence checking? So, sequential equivalence checking is a generalized approach to comparing 2 models meaning that it first converts given 2 models in terms of FSM and then what it does is that, it considers 2 initial states of these 2 FSM and then it checks whether the 2 FSM's produce matching output sequence for all input sequences. If there are 2 FSM's and if these 2 FSM's are or these 2 models are equivalent then given the same input patterns or input sequence, these 2 models should produce identical output pattern, output sequence. If that happens then these 2 models are equivalent otherwise it is not equal. So, this is a generalized method of equivalence checking and this is known as sequential equivalence checking. Now, let us look into what are the challenges of sequential equivalence checking.

So, we can very well understand that if we want to verify that the output sequence produced by 2 FSM's are identical for any given input sequence we need to do state space exploration meaning that from a given starting state what are the states are being traversed and ensure that the machine is or the FSM's are producing identical output for all the states starting from the state S0 that is the given starting state. And this process or this algorithm will have the challenges of state explosion problem that we had discussed in the previous lecture and therefore, it is a computationally difficult task. So, typically what we do in VLSI design flow is that we try to avoid performing sequential equivalence checking rather we do what is known as combinational equivalence checking which is a simpler problem. So, in combinational equivalence checking there is an important assumption and what is the assumption in combinational equivalence checking? The assumption is that there is a one-to-one correspondence among memory elements or flip flops and the ports of the two models. So, if we have two models then for each flip flops or memory elements there is a one-to-one correspondence and also there is a correspondence between the input and output ports.

So, if these correspondences are there, then what happens is that the problem of equivalence checking is simplified to establishing the equivalence of pairs of combinational circuits, not one circuit, but multiple pairs of combinational circuits. So, now, we have to not worry about the states and the traversal of states. But an important thing is that we have to consider whether this assumption that there is a correspondence between the state elements and the ports in the two models, whether that assumption is correct or not. What it turns out is that this assumption is valid for most of the cases. Why? This assumption is valid, this assumption is valid because as VLSI design flow is carried out we typically do not make much changes in the state elements and the ports.

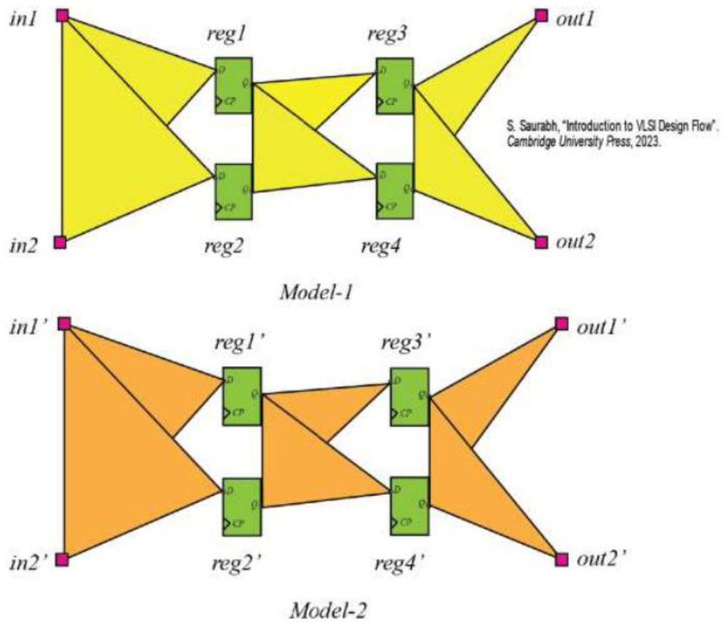
So, the ports that are there at the RTL that remains throughout the design flow, there are not major changes in the ports and also there are not major changes in the state elements. If we are not doing sequential optimization, the state elements do not change. So, the task in VLSI design flow are typically incremental and therefore, if the task are incremental, the changes in the sequential circuit elements like flip flops, those are not very disruptive. Small changes may happen because of say sequential optimization or so, those can be taken care of by special techniques, but major part of our design and in the major design flow, the correspondence between state elements and the ports, that remains intact and therefore, combinational equivalence checking is widely used and now it has become an integral part of design. So, in this lecture we will be looking at the combinational equivalence checking.

So, first let us look into the framework of combinational equivalence equivalence checking. So, for combinational equivalence checking, we give two models: one may be RTL, the other may be net list or these two may be two different net list or so on. Now, two models are there. Now the first task that is done is that establish the correspondence between flip flops or the state elements and the ports and that is what is represented as register and port matching. So, we establish a one-to-one correspondence of the state elements and ports for these two models. Then what is done is that we create multiple miters for our combinational circuit elements in our circuit.


Model-1
Model-2
Register and Port Matching
Miter Creation
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.
Verification by BDD or SAT solver. Are models equivalent?
Yes
No
Equivalence Formally Established.
Debug. Is failure false?
Yes
No
Waive Failure. Models considered Equivalent.
Debug root cause of failure.
VLSI Design Flow: RTL to GDS
NPTEL 2023
S. Saurabh

Now, how we do it, we will see in subsequent part of this lecture and then we invoke BDD and SAT solver to check whether the miters or the combinational logic block between sequential elements, those are equivalent or not. Now, if using these BDD and SAT solver, we can establish the equivalence of these two models if all the miters that we have built and created and verified and found to be equivalent, then we say that our these two models are indeed equivalent. If not then the tool will produce a set of input vectors or stimulus for which these two models are producing different results and then we can use those test pattern to debug and if those test vectors or input stimulus are not valid input stimulus, why it can be invalid, we will just see later. If those are invalid then we can simply waive those failures and we say that these failures are a false failures, we need not worry about it and our designs or these two models are still equivalent or in other case indeed there has been some discrepancy between model 1 and model 2 then we need to debug the root cause and then fix the problem. So, this is the framework of combinational equivalence checking.

Now, let us look into each of these steps that we just mentioned in more detail. So, given two models the first thing that is done by the tool is register and port matching. So, what it means? So, suppose the first model is given as it has say two input ports and say two output ports and these are the registers and between these registers we have combinational circuit elements meaning that it can be a huge combinational block meaning that it is not a simple one or simple logic gate, it can be a huge logic cone or so on, but these are only combinational circuit elements. Similarly these are combinational circuit elements. Now, this is for the first model. So, we have input ports, we have flip flops and then we have the output port.



Now, this is the second model in which we have input ports, output ports and then the flip flops or registers or state elements and these are another cone of combinational logic blocks. Now, what we do is, we first establish the correspondence between the ports meaning that which port corresponds to which. So, in this case we can say that this one corresponds to this one and this one corresponds to this one. So, this is the matching of the input port. Similarly this one matches to this one and this one matches to this and then we have to match the flip flops and we can say that this matches to this one, this matches to this one, this matches to this one and this matches to this one. So, a one-to-one correspondence between the state elements meaning the flip flops and the input and output ports is first established.

Now, how to establish this correspondence. Typically there is no algorithm to or no exact algorithm to establish the correspondence between various circuit elements in two models. What we take help of is some heuristic based algorithm and what is the most popular heuristic. The most popular heuristic is name based matching, why? So, why do we do name based matching because typically say if the name of the port in RTL is say reset then the name in the model or the netlist of the same port will be reset, it will not change. So, name of the ports typically do not change in our design flow and therefore, just by looking at the names we can establish the correspondence. So, this technique, name based matching works very well for the ports.

Now, what about the flip flops or the state elements. There also it works good where a model has not undergone sequential optimization and register correspondence is kept intact. If the registers are there and sequential optimizations have not been done in our design then there will exist the correspondence between the state elements or flip flops and we can find those correspondence using names, most of the time that works. And why it works because RTL synthesis keeps names of registers based on some rules or conventions. So, it does not give arbitrary name to the instances of the flip flops and state elements. Typically the logic synthesis tool follow some conventions or rules in giving names to this flip flops instances. For example, it can add suffix to the name of the signals in the RTL code.

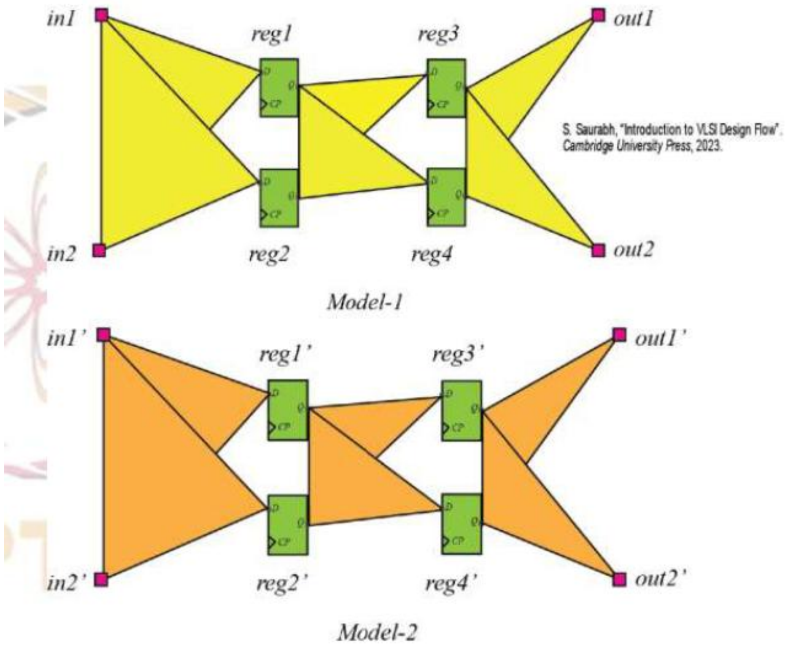
Now, let us take a look at what I mean by adding suffix to the name of the signal in the RTL code. So, suppose our module is written as shown in this case. Now, for this always block we can very well understand that since there is a posedge clock and q is assigned the value of d, it will lead to the inference of a flip flop. So, we will have in our synthesized design as a flip flop in which we have d at the d input and q at the q output of the flip flop and here we will have flop. So, now the question is that what will be the name of this flip flop that will be generated by the synthesis tool.

So, synthesis tool is free to give any name to this instance, but typically it will put the name of this flip flop instance as q_reg. So, _reg will be just the suffix which will be added to the instance name and it will be derived from the name of the signal or which exist in the RTL queue. So, using this name of the instance, the correspondence between flip flops can be established between RTL constructs and the net list and from one net list to another. So, in some cases this name based matching may fail. In that case the tool can use some structural or functional analysis also to do a kind of matching between flip flops and ports. Additionally, this combinational equivalence checking tools have some mechanism through which we can say that this flip flop is being matched to this.

So, suppose there are say 1000 of flip flops in our design for 900 flip flops it may or 990 flip flops, the matching happened based on the name, everything was fine. Now, 10 flip flops were mismatched. So, for them we can debug and based on that we can say that this flip flop matched to this in the other model and so on. So, we can give regular expression also and there are more better ways in which we can specify that this flip flop in one model corresponds to this flip flop in the other model. So, when tool is not able to establish the register and port correspondence then we can directly instruct the tool that the matching is based in a given manner that we know.

Now, once we have got the correspondence between the ports and the state elements, the next step is the creation of the miters. Now, why do we want to create miters because after getting a correspondence between state elements say this and this, what we need to do is that, we need to ensure that the logic cone in the input of this flip flop and in the input of this flip flop, both are functionally equivalent. So, note that these are combinational logic and this is also combinational logic, but their representations can be quite different. Just for an example, in this representation, there can be a gate here which is an AND gate and in this case there are two gates NAND gate followed by an inverter. I am giving just for the sake of an illustration that even though the logic gates can be different in this logic cone and this logic cone, but still they can be equivalent. So, after establishing the correspondence of the state elements and ports, the next task is to ensure that the logic cones in the fan in of the registers and the output ports, those are functionally equivalent. So, we need to ensure that the corresponding combinational circuits lying between the matched registers are equivalent.

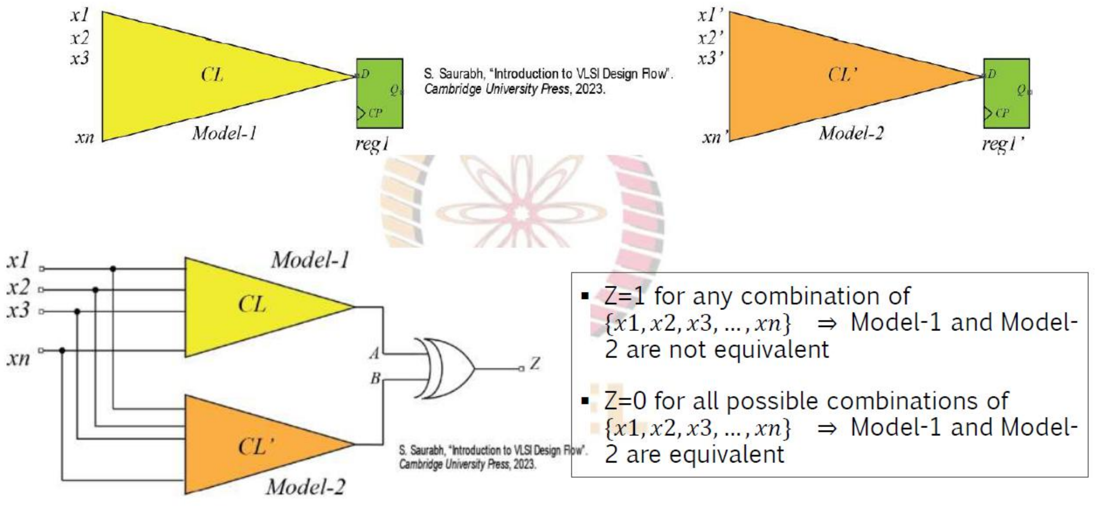
To do this what we do is that we create small combinational circuits which are known as miters for the two models. So, using this original circuit we create multiple miters for the two models and we create separate miters for each register. So, in this case there are 1 2 3 4 registers. So, for all 4, we will have separate miters and also we create miters for the output port.



So, we create miters. So, in this case we will be having miters 1 2 3 4 5 6. So, these miters are created for each comparison points. So, these are known as comparison points, the d pin of the flip flops and the output ports are considered as comparison point for the combinational equivalence checking. So, now given a comparison point what the tool will do. So, this is the comparison point, it will traverse in the fan-in and stop whenever it reaches either the input port or another flip flop, it will stop there.

So, it means that in the fan-in cone of the d pin will have only the combinational circuit elements. Similarly for the other model given. So, this flip flop 1 and this flip flop 1' ,these are 2 flip flops which we have already established correspondence in the previous step. Now we are considering 2 such matched flip flops and their fan-in cone. So, for the other matched flip flop also we traverse in the fan-in cone and stop when we reach input port or the flip flop or q pin of the flip flop, we stop at that point.

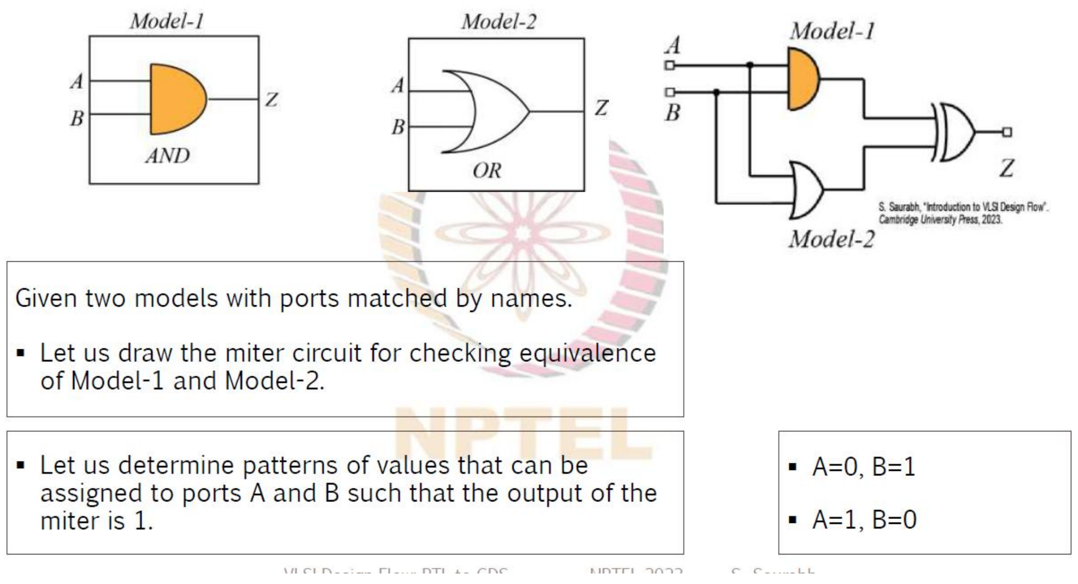
So, we get 2 logic cones. Now we want to establish the equivalence of these 2 logic cones. So, what we do is that we create a miter. So, this is a miter circuit. So, we make the variables that were hit or we make the input port and the q pin of the flip flop that were hit in the fan-in cone as independent variables x1, x2, x3, xn.


x1
x2
x3
xn
CL
Model-1
D
Q
CP
reg1
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.
x1'
x2'
x3'
xn'
CL'
Model-2
reg1'
x1
x2
x3
xn
Model-1
CL
A
B
Z
CL'
Model-2
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

- Z=1 for any combination of $\{x1, x2, x3, \ldots, xn\}$ $\Rightarrow$ Model-1 and Model-2 are not equivalent
- Z=0 for all possible combinations of $\{x1, x2, x3, \ldots, xn\}$ $\Rightarrow$ Model-1 and Model-2 are equivalent

So, these are and this corresponds to this. So, corresponding to those input ports and flip flops, we create inputs in the miter and connect this input to the combinational logic. This is the first combinational logic cone and this is the second one. And then at the output of this combinational logic cone, what we do is that we we add an xor gate we write we have an xor gate. Now if these models are equivalent meaning this model this combinational logic cone CL and CL', these 2 are equivalent. Suppose this CL and CL' are equivalent then what will happen? So, in that case this output will always be 0 and if there is some combination of input for which this model produces say 0 and this model or this combinational logic produces 1 in that case we will get the output z is equal to 1. So, if z is equal to 1 for any combination of inputs, model 1 and model 2 are not equal and if z is equal to 0 for all possible combination of x1, x2, x3, xn, then we can say that model 1 and model 2 are equivalent in terms of this logic cone.

We are only worried about this logic cone at this point of time and subsequently other flip flops will also be picked. So, first we have picked for example, the reg 1 and reg 1'. Similarly we will pick reg 2, reg 3 and reg 4 all 4 registers 1 by 1 and create a miter for that and verify the equivalence. Similarly we will do it for the output port. Now, let us look into an example to illustrate that how the miter works. So, suppose we had 2 models: in one model we had A, B and the output is Z and we are using AND gate and in another model there is an OR gate we have the input same input as A and B and the output Z.


Model-1
A
B
Z
AND
Model-2
A
B
Z
OR
A
B
Model-1
Model-2
Z
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

Given two models with ports matched by names.

- Let us draw the miter circuit for checking equivalence of Model-1 and Model-2.

- Let us determine patterns of values that can be assigned to ports A and B such that the output of the miter is 1.

- A=0, B=1
- A=1, B=0

So, port matching will happen by name, no problem at all and we can easily see that these 2 models are not equivalent why because this is an AND gate and here we have an OR gate. Now, how will the miter be created for this kind of trivial model. So, given 2 models with port match by name and let us draw the miter circuit for the checking of equivalence of model 1 and model 2. So, the miter circuit will be like this. So, at the input we will have A and B. At the output we will have, this is z, this is also z and then we put this XOR gate at the output.

Now, what will be the value of z, what can be the possible values of z? As we vary A and B, the z can vary and let us determine the patterns of values that can be assigned to port A and B such that the output of the miter is 1, when can this output be become 1. When these point and this point has different value either this is 0 1 or this is 1 0 and we can easily see that if A=0, B=1, this will produce 0 and this will produce 1 and the miter will produce an output as 1. So, A=0, B=1 is the input pattern at which these 2 models will give different result and in this circuit we can easily see that if A=0 and B=1, then this will produce output as 0 and if here you have 0 and 1 this will produce output as 1. So, A=0, B=1 is a failing pattern similarly A=1, B=0 is also a failing pattern. So, now how does the CEC tool establish equivalence? So it can use either BDD or the SAT solver.

So, if it is using BDD what it can do is that it creates a BDD for the given miter. So, the miter we had created. Now, for each miter it will create a BDD. Now if the models are equivalent then the miter will produce always a value of 0 and if a function produces always of 0 then the BDD for that will contains only 1 vertex with the value of 0. The leaf node only, it will contain no other node. So, if the models are equivalent the miter function as represented in BDD can contain only 1 vertex and if it is non equivalent then it will yield some other vertex. So, what we do is that we create miter for each of the flip flops and the output port and represent those miters using BDD and if any of them is producing a non-zero value or containing vertex other than the leaf vertex with a value 0 then we say that these 2 models are not equal.

Similarly we can use SAT solver also. Now for a miter what will be returned by the SAT solver. Now if we invoke SAT solver on the Boolean expression represented by a miter, for equivalent model, it should always give an answer as unsat why because there will be no input combination for which the miter output will have a value of 1 and therefore, miter will be unsat and for non equivalent model the miter will give an answer as a SAT. So, it will give us an input combination for which 2 models are producing different answers. So, what we do in CEC is that we create miters for all flip flops and output ports in the model for equivalent model all the miters should be equivalent. If there is any failure means that there is some combinational logic cone which are not equivalent.

Now we should also understand that if we have said that combinational equivalence checking can sometimes give us false failures. When it can give us a false failure. So, note that what we have done is that we are in combinational equivalence checking, we are trying to establish the equivalence of combinational logic cone and we are treating the flip flops q pins as an independent variable meaning that any flip flop can take any value. So, suppose there was a case in which the miter was failing and there were 4 flip flops in its fan in cone and the combination that it produced that led to the failure was for example, 0 1 0 1. Now this 0 1 0 1 is the output of the flip flop meaning that there are 4 flip flops 1 2 3 4 and there is a combinational logic cone which is driven by these 4 flip flops and the values at the output of this flip flops are 0 1 0 1. That is what we are saying that that the combinational equivalence checking tool will give us as the failing pattern.

Now what we have done in combinational equivalence checking we have said that all these flip flops can take any value simultaneously meaning that for these 4 flip flops in effect 2^4 combinations are possible meaning that any flip flop can take any value independent of other flip flops. But what actually happens in a circuit is that these flip flops are elements of a finite state machine and it may be possible that the state 0 1 0 1 is never reached in ournFSM and therefore, this failing pattern which was discovered by the combinational equivalence checking tool will never exist or never come in our circuit. For example, suppose this 4 flip flops were a part of a FSM which was 1 hot encoded in that case only 1 bit can have a value 1 and therefore, 0 1 0 1 will never exist, but by assuming that state elements can take any value irrespective of value of other state elements, we have given much more freedom to or or we are allowing states which are not possible in a circuit to exist and therefore, the combinational equivalence checking tool can sometime gives us a false failure also. If that is the case then we need to debug the case that is reported by the tool and then see that whether that case if it is not possible then we can simply waive off that violation and move ahead and declare that our 2 models are equivalent. But understand that even though the combinational equivalence checking tool can give us false failures, but it will never miss a failure meaning that if there is indeed a situation in which the 2 models are different it will always be reporting that. It will not miss any failures and that is why this combinational equivalence checking is a safe method. Though it can give us some false failures it is still a safe method and that is why combinational equivalence checking is widely used in our design flows.

So if you want to know more about combinational equivalence checking you can go to this reference. Now to summarize what we have done in last 4 lectures is that we have looked into formal verification. We looked into 2 fundamental techniques for formal verification or the fundamental tools or methods that are employed inside formal verification tools and those were BDD based and the second one was was SAT solver based. And then we saw that how these tools can be used for 2 major tasks that are performed in VLSI design flow: the first is model checking and second is equivalence checking. Now using these techniques we can discover functional problems in our circuit.

So this is one aspect of our design but what about the timing of our design meaning that we have got a circuit but the delay between 2 flip flops is very large and the synchronous behavior of our circuit is being lost. So that is a timing problem. So besides the functional behavior of our circuit the other important aspect of our circuit is the timing safety or temporal safety of our circuit. To ensure this what we do is that we carry out static timing analysis. So, in next few lectures we will be looking at static timing analysis.

We will start with looking at library because library contains vital information of the delays of the standard cells and cells that we use in our design and then use these libraries for static timing analysis and for other purposes. So we will be looking into libraries and followed by the static timing analysis in the next couple of lectures. Thank you very much.